BSc (Hons) Computer Science

City, University of London

Individual Project

UniMeetUp: A dynamic web application for university students to meet new people

Proposed by: Rosen Georgiev
Student e-mail address: Rosen.Georgiev@city.ac.uk

Consultant: Laure Daviaud
Consultant e-mail address: Laure.Daviaud@city.ac.uk

No arrangements for proprietary interests

Word count of main body: 9215

Table of Contents

Chapter 1: Introduction ... 5

1.1 Problem To Be Solved ... 5

1.2 Project Objectives ... 5

1.2.1 Main objective ... 5

1.2.2 Sub objectives ... 5

1.3 Beneficiaries ... 6

1.4 Assumptions and Project Scope ... 6

Chapter 2: Output Summary ... 7

Chapter 3: Literature Review ... 9

3.1 Programming language and frameworks ... 9

3.1.1 JavaScript ... 9

3.2 Solution architecture ... 10

3.2.1 IDE ... 10

3.2.2 Database ... 11

3.3 Authentication ... 12

3.3.1 JWT (JSON Web Tokens) ... 12

3.3.2 Amazon SES (Simple Email Service) ... 12

Chapter 4: Method ... 13

4.1 Software engineering methodology ... 13

4.2 Management ... 13

4.2.1 Version Control ... 13

4.3 Analysis ... 14

4.3.1 Requirements gathering ... 14

4.3.2 Use Case Diagram ... 14

4.4 Design ... 14

4.4.1 System architecture ... 14

4.4.2 Logo and user interface ... 15

4.5 Implementation ... 15

4.5.1 Development environment ... 15

4.5.2 Programming language ... 15

4.5.3 Database connection and operations ... 16

4.5.4 Features ... 17

4.5.5 Software reuse ... 19

4.6 Web hosting ... 19

4.7 Testing ... 19

Chapter 5: Results 20
5.1 Software engineering methodology 20
5.2 Management 20
5.2.1 Version Control 20
5.3 Analysis 21
5.3.1 Requirements gathering 21
5.3.2 Use Case Diagram 22
5.4 Design 22
5.4.1 System architecture 22
5.4.2 Logo and user interface 25
5.5 Implementation 25
5.5.1 Development environment 25
5.5.2 Programming language 26
5.5.3 Database connection and operations 26
5.5.4 Features 30
5.5.5 Software reuse 40
5.6 Web hosting 41
5.7 Testing 42
Chapter 5: Conclusions and Discussion 44
6. Reference List 45
7. Appendices 46
7.1 Appendix A Project Definition Document 46
7.2 Appendix B Reuse Summary 53
7.2.1 Software libraries 53
7.2.2 Reused code 54
7.3 Appendix C Requirements 56
7.3.1 Volere requirements 56
7.3.2 Use case diagram 61
7.4 Appendix D Design 62
7.4.1 Logo 62
7.4.2 Design documentation: webpages 62
7.5 Appendix E Test Report 69
7.6 Appendix F Source Code 72
7.7 Appendix G Deployment Guide 73
7.8 Appendix H Video 75

Abstract

The project goal was to address the problem of meeting new people at university. Many students find it hard to make friends, especially in their first year of higher education. Having a hard time socializing at university could also lead to academic difficulties. Therefore, students could benefit from having software built for them to connect.

The aim of the project was achieved by building a web application that enables university students to meet new people from their university or other UK universities. Students can find new people and view a summary of their profile including university, course, year of study, and interests and connect with them on a mutual-like basis achieved through a matching algorithm. Furthermore, they can communicate via instant messaging. This report describes the research, design, and implementation as part of the development process of the web application.

Chapter 1: Introduction

1.1 Problem To Be Solved

University life is a unique time in every student’s journey. However, there are students who struggle to meet people at university. It can be intimidating to socialize when you are adjusting to a new environment. This problem is also relevant for international students who have been separated from important people in their life such as family and friends when moving to a new country for their higher education. The proposed web application addressed the problem by providing a platform for students to find others with similar university experience or interests and communicate.

1.2 Project Objectives

1.2.1 Main objective

The main objective is to connect students who are looking to meet new people and potentially date by architecting and building a fast, responsive, and secure web application. The platform should allow students to easily find and communicate with other students who share similar interests and preferences.

1.2.2 Sub objectives

- Provide a high quality user experience by focusing on usability, simplicity, and intuitive design to ensure the users can easily navigate the web application and use its functionalities
- Optimize web application performance by minimizing HTTP requests, compressing data, and caching data when possible
- Follow security guidelines and implement secure coding practices such as encrypting data, hashing passwords, requiring input validation, and avoiding security misconfigurations
- Ensure user safety: implementing safety features such as a functionality to block users and prevent them from sending messages is essential for promoting user safety, trust, responsible behavior, and legal compliance in the software
- Make the software maintainable by writing clean, well-document code and using a version control system
- Perform testing to catch bugs and ensure that functional and non-functional requirements are met

1.3 Beneficiaries

As stated in the Project Definition Document available in **Appendix A**, the anticipated beneficiaries of the project are university students. Attending in-person events is a great way to meet people, but students can also benefit from utilizing a web application built for them to meet other students online. There are other similar software systems on the market that offer features such as mutual-like, match, and instant messaging, but their users are not solely students.

1.4 Assumptions and Project Scope

Assumptions made:

- Target audience: the author assumed who the website is for and what their needs and preferences would be
- Content: it was assumed what kind of information will be presented, and how it will be structured and organized
- User flow: the author also assumed the path which a user will follow on the website to accomplish their goals. This includes assumptions about navigation bar, buttons, and other components
- Technologies and tools: the author assumed what technical stack will be used to build the site, as well as what security measures will be required
- Design: assumptions were also made about design, including layout, colors, and images
- Timeline: assumption about the time required to complete the project, including planning, design, build, testing, report writing

The author decided to concentrate on building the main functionalities required for this type of web application first. Focusing on achieving strong building blocks in the implementation process established a solid base. It was assumed that more ideas will be implemented in the future, improving the website experience.

Chapter 2: Output Summary

The complete software product is built as 16 JavaScript files and 2 package.json files, amounting to 2040 lines of code (not counting node_modules folders that include library code generated by the package manager), of which 1523 lines were written by the author and the rest were re-used from the sources indicated in the code files, the Results section, and Appendix B Reuse Summary.

Overview description	React.js front-end application, the client side of the project.
Output type	Software code
Recipients	Assessors
How recipients will use the output	Recipients can use this output for testing
Link to Results section	5.5 Implementation
Link to Appendix	Appendix F

Overview description	Node.js back-end application, the server side of the project.
Output type	Software code
Recipients	Assessors
How recipients will use the output	Recipients can use this output for testing
Link to Results section	5.5 Implementation
Link to Appendix	Appendix F

Overview description	Project demonstration video: demonstrating key features. Includes code walkthrough
Output type	Video
Recipients	Assessors, Users
How recipients will use the output	Assessors can see a demo of main features and a walkthrough of a code section. Users can learn how to use the software.
Link to Results section	5.5.4 Features
Link to Appendix	Appendix H

Overview description	Deployment guide
Output type	PDF document
Recipients	Assessors
How recipients will use the output	To install the software
Link to Results section	5.5.1 Development environment
Link to Appendix	Appendix G

Overview description	An online hosted version of the software is available at: http://16.16.81.184:3000/
Output type	URL
Recipients	Assessors, Users
How recipients will use the output	Assessors can use the output for testing. Users can use the output to benefit from the features
Link to Results section	5.6 Web hosting
Link to Appendix	Appendix G

Chapter 3: Literature Review

This project required familiarity with application development technologies and concepts, as such extensive reading was required. This section covers the research undertaken during the initial development of the project, including a review of programming languages, frameworks, tools, and deployment systems.

Software Development, Design and Coding (2017) by John F. Dooley provided more detailed information about the software development cycle. The book covers relevant topics such as Software Architecture, Design Patterns, and Object-Oriented Principles. It describes how every program has a life cycle, despite how large or small the program is. The process of architecting a software system is described to start by drawing pictures/diagrams, because they allow people to see the structure and framework of the program much more easily than text allows. The book also covers different software process models, modern programming languages, and useful advice about the implementation stage. It was also interesting that one of the chapters was focused on project management essentials. "Project management? Isn't this a software development book?" (Dooley, 2017). It goes to show that project planning, scheduling, resource management, and project oversight are also important to be considered alongside the technical aspect of the work.

3.1 Programming language and frameworks

3.1.1 JavaScript

"JavaScript is a high-level, often just-in-time compiled language that conforms to the ECMAScript standard. It has dynamic typing, prototype-based object-orientation, and first-class functions. It is multi-paradigm, supporting event-driven, functional, and imperative programming styles." (Wikipedia, n.d.)

Used alongside HTML & CSS, JavaScript provides webpage behavior and client-side functionality. JavaScript libraries/frameworks used:

- React.js: used for building user interfaces and front-end components.
- Node.js: used for server-side development of main functionality. Node is a JavaScript runtime environment built upon event-driven programming. It provides functionalities for building real-time chats, which was useful for developing communication features of the web application. Furthermore, Node was utilized to implement server-side events.

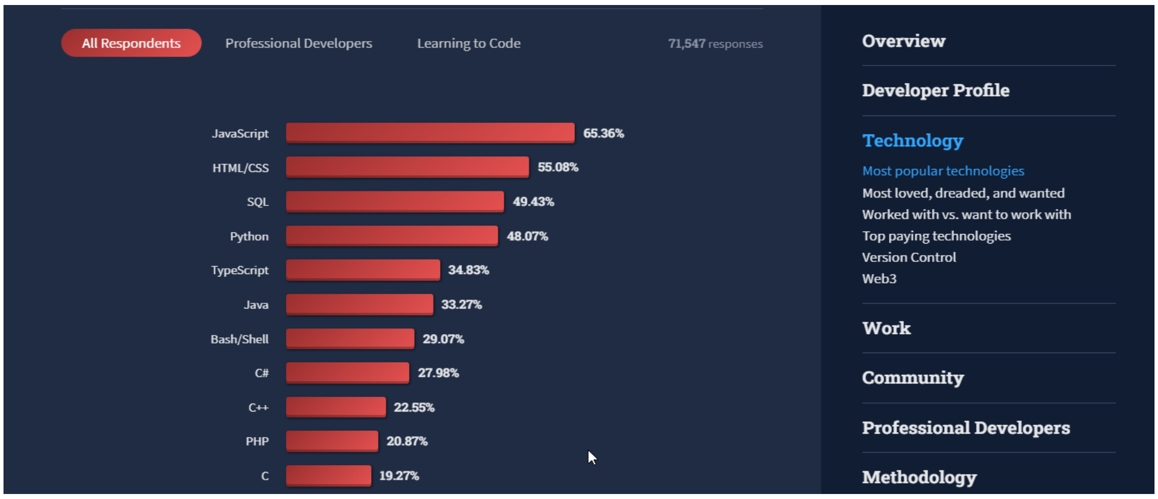


FIGURE 1: IN 2022, STACK OVERFLOW SURVEYED MORE THAN 70,000 DEVELOPERS TO LEARN ABOUT THEIR FAVORITE AND MOST USED PROGRAMMING LANGUAGES, WITH JAVASCRIPT BEING ONE OF THE MOST POPULAR TOOLS (STACK OVERFLOW, 2022)

3.2 Solution architecture

3.2.1 IDE

3.2.1.1 WebStorm by JetBrains

"WebStorm is an integrated development environment for JavaScript and related technologies. Like other JetBrains IDEs, it makes development experience more enjoyable, automating routine work and helping handle complex tasks with ease." (JetBrains, n.d.)

The main integrated development environment (IDE) used to develop the web application was WebStorm. It provides a good user interface, robust customization options, and support for multiple programming languages and software packages. WebStorm was chosen because it is specialized in JavaScript and has native support for tools required for the project such as React, Node, HTML, and CSS.

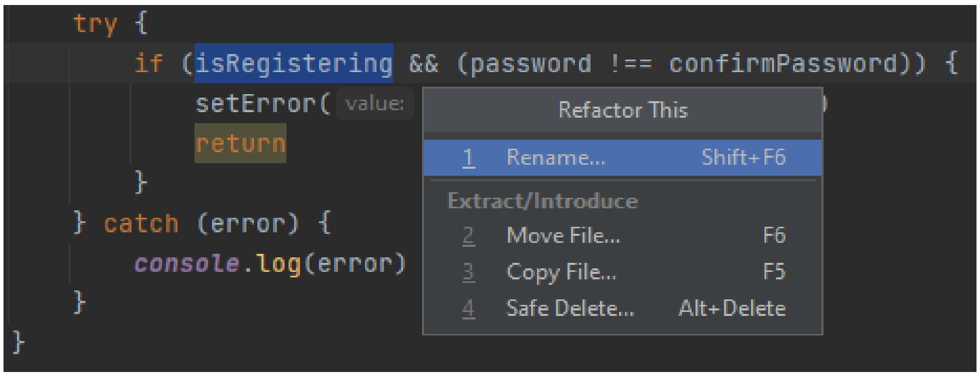
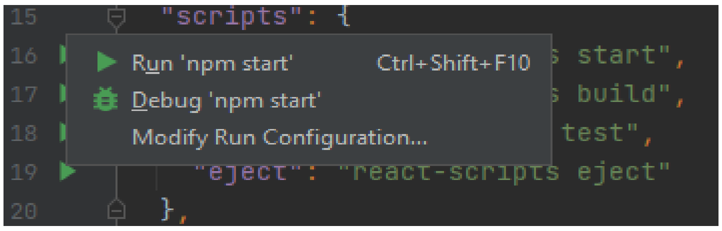

```
    try {
        if (isRegistering && (password !== confirmPassword)) {
            setError( value:
            return
        }
    } catch (error) {
        console.log(error)
    }
}
```

FIGURE 2: WEBSTORM DEEPLY UNDERSTANDS THE PROJECT STRUCTURE AND CAN ASSIST WITH ASPECTS OF WRITING CODE. IT CAN DETECT AND SUGGEST FIXES FOR ERRORS AND REDUNDANCIES, AND HELP REFACTOR CODE SAFELY



```
"scripts": {
                                   s start",
                                   s build",
                                  test",
    "eject": "react-scripts eject"
},
```

FIGURE 3: WEBSTORM HAS BUILT IN DEVELOPMENT TOOLS SUCH AS LINTERS, TERMINAL, AND HTTP CLIENT, ALL INTEGRATED WITHIN THE IDE.

3.2.2 Database

When it comes to databases, the author had to choose between a SQL or NoSQL one. SQL databases are table-based while NoSQL databases are generally document-based or graph-based. SQL databases have predefined schema while NoSQL databases are more flexible and have dynamic schema.

3.2.2.1 MongoDB

"MongoDB is an open-source document database built on a horizontal scale-out architecture that uses a flexible schema for storing data. Instead of storing data in tables of rows or columns like SQL databases, each record in a MongoDB database is a document described in BSON, a binary representation of the data. Applications can then retrieve this information in a JSON format." (MongoDB, 2019)

After weighing the pros and cons, the author decided to use a NoSQL database and utilized MongoDB as it is accessible, flexible, and can be integrated with JavaScript. MongoDB is a document-oriented database, which means that it stores data in a document format (typically JSON or BSON). This makes it possible to work with data structures that are more complex than simple tables and rows. MongoDB is designed to run on a distributed architecture, which means that it can be deployed across multiple nodes and clusters. This provides high availability and fault tolerance, and ensures that the database is always accessible to users. The database does not require a predefined schema, which means that the data can be changed and modified easily as the application evolves. This makes it easier to work with semi-structured and unstructured data, and allows for more flexibility in data modeling. It is designed to handle large amounts of data with high read and write throughput. It provides fast response times and low latency, making it ideal for applications that require real-time data processing.

3.3 Authentication

The main options considered for an authentication mechanism were OAuth (Open Authorization) and JWT (JSON Web Tokens). The author decided to utilize JWT after identifying several pros.

3.3.1 JWT (JSON Web Tokens)

JSON Web Tokens contain JSON objects which have the user information that needs to be shared between the web application front-end and the server for authentication purposes. Each JWT is also signed using cryptography (hashing) to ensure that the JSON contents (also known as JWT claims) cannot be altered by the client or a malicious party. JWTs are self-contained, meaning that all the necessary information is contained within the token itself. This allows for decentralized authentication, since the token can be passed between different services or applications without requiring a central authority.

3.3.2 Amazon SES (Simple Email Service)

Amazon SES (Simple Email Service) is a cloud-based email sending service provided by Amazon Web Services (AWS). SES provides a cost-effective way to send and receive emails using email addresses and domains. SES can be used to send verification emails to users upon registration. It supports various email sending scenarios such as reputation monitoring to help maintain a high deliverability rate and avoid sent emails being marked as spam.

Chapter 4: Method

4.1 Software engineering methodology

With limited experience in using software development methodologies other than the Waterfall method, researching alternative approaches and weighing their pros and cons in the context of the project was required. Given the limited available time for development and strict deadlines, the author decided to stick with Agile, because it is usually faster than Waterfall and allows for more flexibility. An Agile approach enabled the author to split the project into smaller pieces (known as sprints) throughout the different project stages (requirements gathering, design, development, and testing).

"The ultimate value in Agile development is that it enables developers to deliver value faster, with greater quality and predictability, and greater aptitude to respond to change." (Cprime, 2023).

4.2 Management

The author used several strategies to stay on track and increase productivity. He created a centralized folder to store all documentation required and used throughout the project. This allowed for a good organization and easy access to resources for references. A personal diary (created with pen and paper) was utilized to write the project's findings, problems and their solutions. This proved very useful in setting clear, short-term goals for the specific project stage. Furthermore, the Author used an app called Calendar to explicitly schedule time to work on project analysis, design, implementation and report writing. As the saying goes, "If you fail to plan, you are planning to fail". Therefore, the author made sure to plan and prioritize tasks well in advance. Scheduling enabled him to prioritize important tasks, identify potential time wasters, and ensure progress.

4.2.1 Version Control

GitHub was used as a version control system. It allowed the author to keep track of the work in progress and helped explore the changes made throughout the process when it comes to data, code, and notes. *"GitHub is a for-profit company that offers a cloud-based Git repository hosting service. Essentially, it makes it a lot easier for individuals and teams to use Git for version control and collaboration."* (Kinsta, 2022)

4.3 Analysis

4.3.1 Requirements gathering

As part of the analysis stage, the author researched existing social networking applications on the market and made a list of essential functionalities, ranking them by priority. Gathering requirements also consisted of searching for reviews and discussions about online networking/dating platforms. A user-friendly interface was a must for the software. Engaging user experiences are built on a foundation of solid interface design. Before the author started the implementation, he researched fundamental design concepts for building clean, efficient interfaces for a broad set of users. While brainstorming for how the project should differentiate from the competition, it was decided to build the application with a focus on a niche user base (students) in order to provide a more tailored and unique experience.

4.3.2 Use Case Diagram

Designing a use case diagram was an appropriate way to understand user interactions with the system and display relationships between actors and use cases. "*A key concept of use case modeling is that it helps us design a system from the end user's perspective. It is an effective technique for communicating system behavior in the user's terms by specifying all externally visible system behavior. A use case diagram should be simple and contains only a few shapes." (Visual Paradigm, 2019).*

Throughout the author's academic career, he has learnt that describing the system from a user's perspective by designing a use case diagram is one of the best practices when starting a software development project. Visual Paradigm was the obvious choice for this purpose. It provided useful features such as generalisations and include/extend relationships between use cases.

Visual Paradigm is a sophisticated application suitable for designing object-oriented software with a massive set of useful features for designers (Visual Paradigm, 2019).

4.4 Design

4.4.1 System architecture

The author divided the system into two parts using the client/server approach. The client is the part of the system that requests services or resources from the server. It consists of user-facing components, such as a web pages, that allow users to interact with the system. The server, on the other hand, is responsible for providing the requested services or resources to the client. It consists of backend components / APIs. In this approach, the client and server communicate with each other using a standardized set of protocols, which define how data is transmitted and received. This allowed for scalability, as multiple clients could connect to a single server, and also enabled more efficient resource utilization, as the server could manage resources more effectively than if they were distributed across multiple clients.

4.4.2 Logo and user interface

The author designed the web application logo using *https://app.logo.com/*. The logo consists of the website name and a graduate cap symbolizing the website objective and the fact that is tailored towards university students. The author aimed to choose a consistent color scheme that works with the application’s branding, target audience, and user experience and decided to use green and white colors for the web pages. Green, the main color, was a good option because it is associated with growth, harmony, and balance. Furthermore, green is often used to symbolize hope, which is a good fit for an application that helps people find meaningful relationships.

4.5 Implementation

4.5.1 Development environment

WebStorm was used as the IDE for implementing the front-end user interface and server-side logic. WebStorm provided advanced support for JavaScript, including error checking and refactoring tools, which helped the author write high-quality code faster and with fewer errors. It also provided specific features for React.js development, such as code completion and a built-in debugger. WebStorm offered support for Node.js, allowing the author to write and test server-side JavaScript code. The IDE integrated well with other tools used in the project, such as Git and npm, making it easy to manage project dependencies and maintain version control. The author installed additional packages required for the implementation and followed fundamentals of object-oriented programming: encapsulation, abstraction, and inheritance.

4.5.2 Programming language

Planning the implementation stage well in advance, the author researched the possible programming languages available on the market which would be suitable for achieving the project objectives and provide optimal results. Throughout the research, JavaScript stood out with its speed and versatility. There are many different ways to integrate it into a website. React.js was the main framework used to develop the front-end. Its main advantage is the fact that it automatically updates the user interface based on modifications and updates within a particular component. Since Node.js integrates effectively with MongoDB, it was a good option to handle the back-end. JavaScript offers a variety of interfaces to build engaging websites. Furthermore, JavaScript seamlessly integrates with other programming languages, which would be beneficial for future improvements of the web application if there are features to be implemented with a different programming language.

Furthermore, using JavaScript leads to an increase in the speed and performance of an application. For example, PayPal replaced Java with JavaScript as a server-side platform for their application and published a report by Nick Heath in 2013 on how Node.js helped them increase performance levels and reduce the amount of code significantly. According to the report, the JavaScript application was built twice as fast as the one written in Java with fewer people in the team. It was also stated that the JavaScript application consisted of 33 percent fewer lines of code and 40 percent fewer files with both applications without impacting the functionality. The report is available at: https://www.zdnet.com/article/how-replacing-java-with-javascript-is-paying-off-for-paypal/

4.5.3 Database connection and operations

4.5.3.1 Registering new users: db.collection.insertOne()

When a new user creates an account, they need to fill out a registration form with their email and password (including a confirm password field) and click a submit button. The aim in this development step was to create a functionality to send this information to the back-end upon clicking the submit button. For this purpose, the **insertOne()** method inserts a single document into a collection. The method works both with or without specifying an **id** field. If a document is inserted in the collection without an **id** field, then MongoDB will automatically add an **id** field and assign it with a unique **ObjectId**.

4.5.3.2 Signing in as an already existing user: db.collection.findOne()

The **findOne()** method finds and returns one document that matches the given selection criteria (in this case, searching for a user by their e-mail address in the collection of users). If no document matches the selection criteria, then this method returns null. It takes two parameters: the first one is the query criteria and the other is optional. As part of this function, the author also implemented a step to check if the password matches the user's hashed password in the database once the user has been found by e-mail in the previous step.

4.5.3.3 Filling out profile details form: db.collection.updateOne()

The **updateOne()** method updates a first matched document within the collection based on the given query. When a document is updated, the **id** field remains unchanged as it is a unique value. This method updates one document at a time and can also add new fields in the given document. Used to update user's details in the user collection once a user has filled out the onboarding form after signing up. Also used to implement a functionality which allows profile editing.

4.5.4 Features

4.5.4.1 Authentication

An authentication mechanism was one of the fundamental components as part of the project. When a new user registers, they need to provide their student e-mail address (@university.ac.uk) to sign up. After a user registers, the backend server generates a JSON web token. The browser saves the token in the cookies and it will be attached to every API request from that user. After registration and email verification, the user is able to fill in an onboarding form with their profile details.

4.5.4.2 Email verification

The author implemented an email verification mechanism using Amazon SES (Simple Email Service) to ensure that the person registering for the website is the owner of the email address provided and confirm their student status. This helped prevent fake registrations and ensured that the website user base is made up of legitimate users.

4.5.4.3 User Profiles

The user profiles function allows users of the web application to update their personal details such as their name, course, year of study, and bio. This feature was incorporated for several reasons:

- User engagement: allowing users to personalize their profiles and share more about themselves can increase engagement with the app and help users find more meaningful matches
- Accuracy: by giving users the ability to update their personal details, the application ensures that the information displayed on their profiles is accurate and up-to-date

- Flexibility: as users' personal circumstances change, they may want to update their profile to reflect those changes. Allowing users to easily edit their profile information adds flexibility to the application and can help users feel more in control of their online dating experience

The ability to edit profile is an important feature of the application that helped create a better user experience and ensure accuracy of the information displayed on users' profiles.

4.5.4.4 Explore student profiles

A user has a profile page with information they have filled out previously in an onboarding form. By default, every user has a profile page which displays their name, university, course, and year of study. They can also write a short bio about themselves. A user can explore other users' profile pages and "swipe" left or right depending on whether the user wants to have an interaction. Users are matched based on a mutual-like system (if they both swipe right).

4.5.4.5 Real-time chat

When there is a match between two users, they have the opportunity to communicate via chat. The chat functionality consists of a message text editing field with keyboard, conversation windows with sent and received messages clearly distinguished from each other and ordered chronologically, ability to receive, interpret, and display a message, storage of past messages, and a list of matches you can message and receive messages from. The author's aim was to provide a simple and intuitive communication experience where end users do not necessarily notice individual features and just open a keyboard below a message window, so each feature feels like a part of the whole.

4.5.4.6 Block a user

It was essential to take measures against potential risks in the online world. Application users are able to block other users if they find them suspicious or do not want to interact with them at all. This functionality is important for a number of reasons. It allows users to feel more in control of their experience on the website. If a user encounters another user who is making them uncomfortable or unsafe, being able to block that user can help the first user feel safer and more secure. Moreover, having a block user functionality can help prevent harassment and abuse on the platform. If a user is repeatedly harassing or bothering others, those users can choose to block the offending user, which prevents the user from interacting with them in any way.

4.5.4.7 Icebreaker questions

The icebreaker questions feature was implemented with the goal of promoting more meaningful and engaging conversations on the platform and encouraging users to connect beyond the initial match. Students can generate a conversation starter from a pre-defined list of questions. By providing a structured way for users to initiate conversations, the icebreaker questions feature helps to overcome the awkwardness or uncertainty that can sometimes arise when starting a conversation with a stranger. It also helps to ensure that users have something to talk about, and encourages them to engage in interesting conversations.

4.5.5 Software reuse

Reusing software was incorporated to increase consistency, reduce errors, and encourage best practices throughout the project. The full list of reused libraries can be found in **Appendix B.** The author installed and utilized already existing software packages such as mongodb, dotenv, bcrypt, cors, axios and others which were useful for different purposes. The author also reused code in order to configure the AWS SDK (Software Development Kit) which was required for implementing the email verification functionality with Amazon SES (Simple Email Service). Furthermore, the author reused code from a demo of the react-tinder-card library and adapted it for the project needs in order to implement user profiles as card elements that can be swiped on. More details, including the reused code and original sources can be found in **Appendix B**.

4.6 Web hosting

The author hosted a prototype of the software on Amazon Web Services (AWS) to enable users to access it online.

4.7 Testing

The author utilized testing methodologies to make sure the software can successfully operate in multiple environments and across different platforms. These were broken down between functional and non-functional testing. Functional testing involved testing the application against the objectives defined in the Project Definition Document. It incorporated different test types designed to guarantee each part of software behaves as expected by using previously created uses cases. Non-functional testing methods incorporated test types focused on the operational aspects of the program, including security, performance, and usability.

Chapter 5: Results

This chapter covers more details about the information discussed in **Chapter 4: Method** and the results achieved from the methodology procedures. It follows the same structure as **Chapter 4: Method** to make the report readable and easy to follow.

5.1 Software engineering methodology

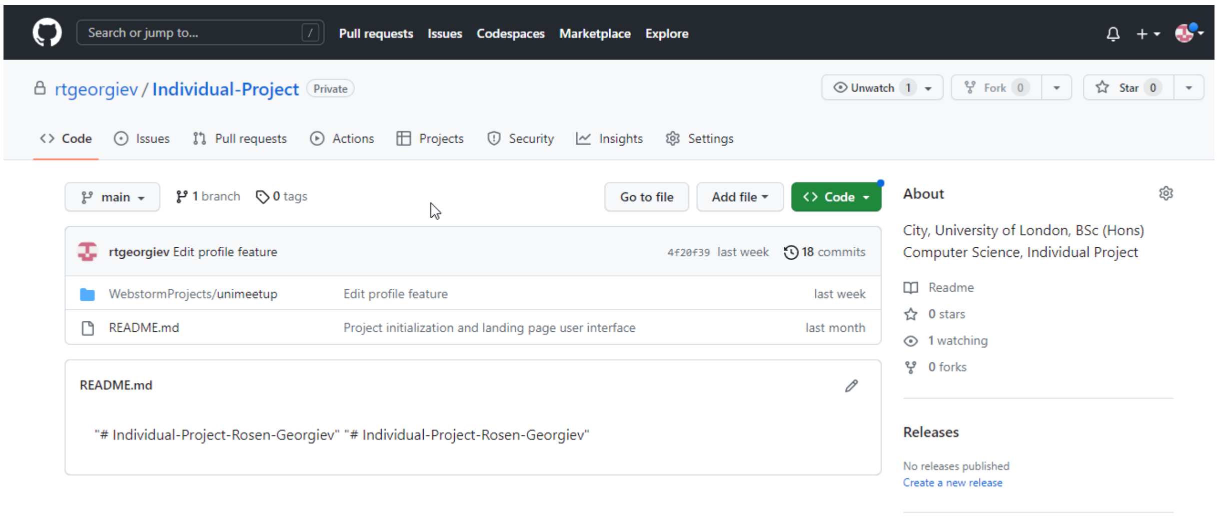
As previously mentioned, an Agile development method was followed throughout the project lifecycle. Benefits which the Agile approach resulted in included increased visibility, adaptability, increased value provided to the beneficiaries, and decreased risk to deliverables.

5.2 Management

In **Chapter 4.2**, the author discussed project management techniques such as planning well in advance, keeping records of relevant documentation, and scheduling time in the calendar to work on explicitly defined tasks. Utilizing those methods throughout the project resulted in having clear and measurable goals, as well as ensuring consistent progress and meeting deliverables. Using a calendar tool for scheduling resulted in balancing high-effort and high-priority tasks, better workload management, preserving work-life balance and helped meet deadlines.

5.2.1 Version Control

When a specific development task had been completed as stated in the work plan, the changes were committed and pushed to GitHub: the chosen version control system for this project. The project code base as well as assets, components, and configuration files were stored within a private GitHub repository, with the ability to view old versions, revert files to a previous version and view changes.



rtgeorgiev / Individual-Project Private

rtgeorgiev Edit profile feature		4f20f39 last week 18 commits
WebstormProjects/unimeetup	Edit profile feature	last week
README.md	Project initialization and landing page user interface	last month

README.md

"# Individual-Project-Rosen-Georgiev" "# Individual-Project-Rosen-Georgiev"

About

City, University of London, BSc (Hons) Computer Science, Individual Project

Readme
0 stars
1 watching
0 forks

Releases

No releases published
Create a new release

FIGURE 4: GITHUB REPOSITORY

5.3 Analysis

5.3.1 Requirements gathering

The following section highlights the functional and non-functional requirements. The author researched reviews and comments about existing software as mentioned in **Chapter 4.3** to note what features are mandatory for any online social networking platform. This helped established the overall project objectives.

Requirement ID: 1	**Requirement Type: Functional**
Description: The website shall allow its visitors to create accounts.	
Rationale: Individual accounts are required for a personalized user experience.	
Source: Analysis of existing software	
Fit Criteria: Users will receive a validation email to confirm their registration on the application	
Customer Satisfaction: 5	**Customer Dissatisfaction: 4**
Priority: Essential	**Conflicts:** None
Supporting Material: None	Volere
History: New requirement	Source: Atlantic Systems Guild

FIGURE 5: FUNCTIONAL REQUIREMENT EXAMPLE

Requirement ID: 9	**Requirement Type: Non-functional**
Description: The system must adhere to good software security practices in order to protect user data.	
Rationale: The website stores personal information.	
Source: Author	
Fit Criteria: Security best practices will be followed. Every security conflict will be investigated to prevent future incidents.	
Customer Satisfaction: 4	**Customer Dissatisfaction: 5**
Priority: Essential	**Conflicts:** None
Supporting Material: None	Volere
History: New requirement	Source: Atlantic Systems Guild

FIGURE 6: NON-FUNCTIONAL REQUIREMENT EXAMPLE

Full list of gathered requirements (functional and non-functional) is included in **Appendix C.**

5.3.2 Use Case Diagram

After gathering requirements, the author worked to achieve a better overview of the entire system and have a clear visual representation. Visualising a precise and correct system, the author produced a use case diagram for the project. The use case diagram clearly demonstrated functional requirements of the system. The use case diagram created also encapsulated the possible interactions between the actors and the system. The use case's main actor was the student. The use case diagram can be found in **Appendix C**.

5.4 Design

The design for the project is based on a plain and elegant approach, minimising the amount of content in the background to ensure the app remains effective and stylish. Before commencing development within the IDE, it was preferred that the application had an initial design process completed. To achieve the design, the author produced prototypes for the user interface on pen and paper which were then implemented as user interface design of the webpages. Webpage design can be found in **Appendix D**.

5.4.1 System architecture

The architecture diagram was created with draw.io. The visual representation of components showed the web pages of the client side and communication between different elements.

Having a well-designed architecture diagram to provide a high-level overview of the system components proved to be very useful in the implementation stage.

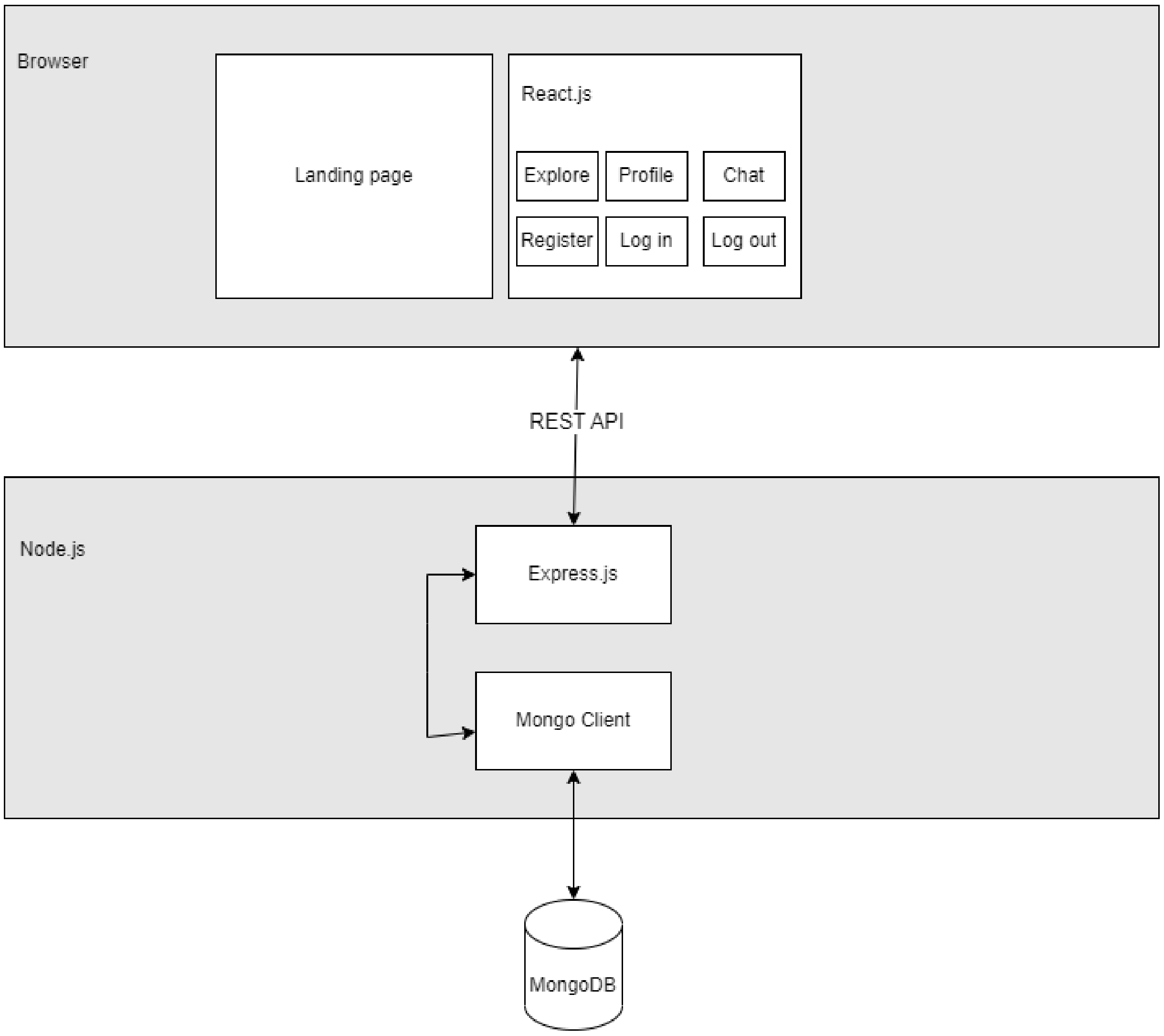
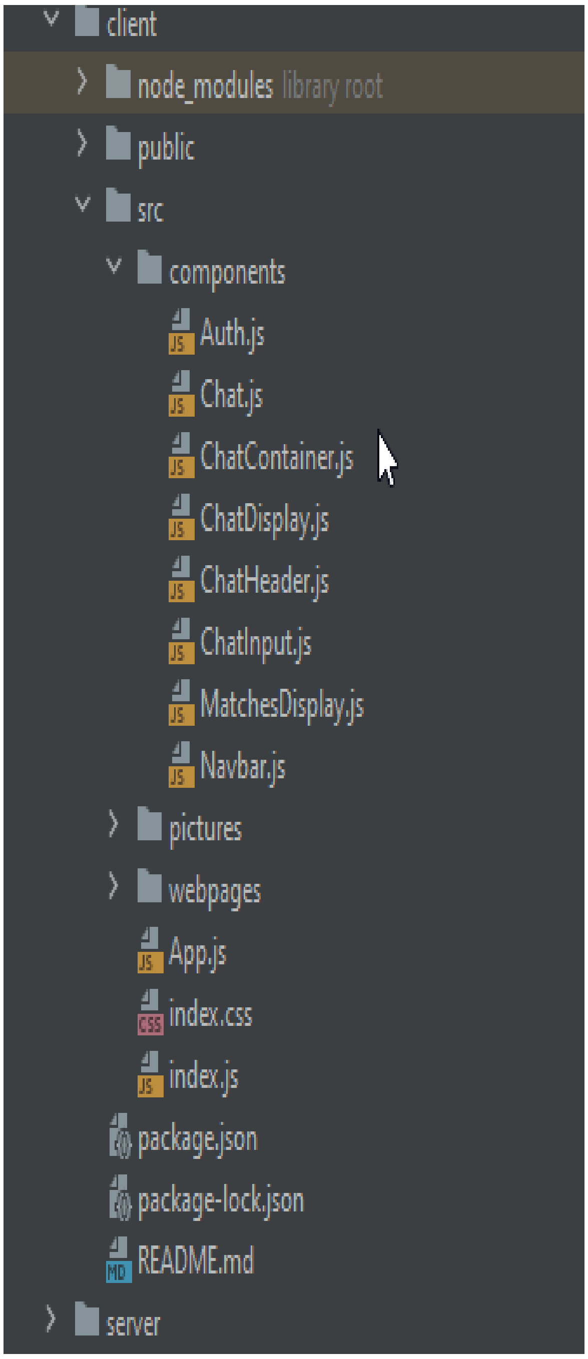


FIGURE 7: ARCHITECTURE DIAGRAM

Figure 7 illustrates two parts: front-end and back-end. The back-end runs in a Node.js environment and MongoDB is used as a data storage. On the front-end part, there is the client side: a React.js application, which is retrieving and storing data on the back-end using REST APIs. Mongo client is used to establish a connection to the database. Express.js framework is used for building the API for external services (such as a browser).

React.js was the main framework used to develop the client-side of the project and build an interactive user interface based on components.

- client/src/components stores the main layout components such as the navigation bar, chat UI, and other components of the dashboard. Components are the basic building blocks of the application. Every component file stores code to carry out custom functionalities.
- client/src/pictures is used for static contents storage such as website logos and other assets
- client/src/webpages includes the home page, verify email, edit profile, onboarding form, and dashboard.
- App.js is a a concatenation of all of the user interface files. It also includes web page routes, linking URLs with their corresponding components
- index.css is the main CSS file used to apply style and define layout
- package.json: records important metadata about the project which is required before publishing to npm, and also defines functional attributes of the project that npm uses to install dependencies, run scripts, and identify the package entry points.



```
client
  node_modules library root
  public
  src
    components
      Auth.js
      Chat.js
      ChatContainer.js
      ChatDisplay.js
      ChatHeader.js
      ChatInput.js
      MatchesDisplay.js
      Navbar.js
    pictures
    webpages
    App.js
    index.css
    index.js
  package.json
  package-lock.json
  README.md
server
```

FIGURE 8: FROND-END FOLDER STRUCTURE

The /server directory contains the files for the application logic. The back-end was implemented with Node.js: a JavaScript runtime primarily used for creating both server-side and dynamic web applications.

5.4.2 Logo and user interface

The logo of UniMeetUp is also included in **Appendix D** alongside the design of web pages.

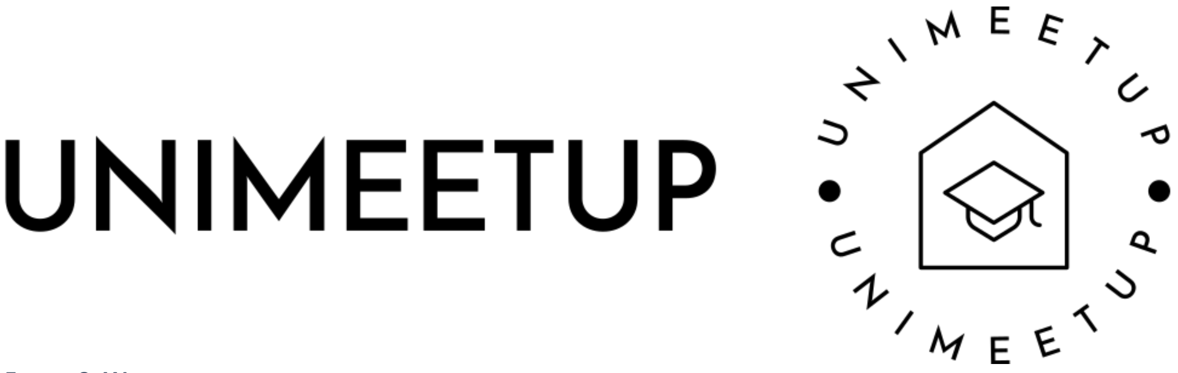
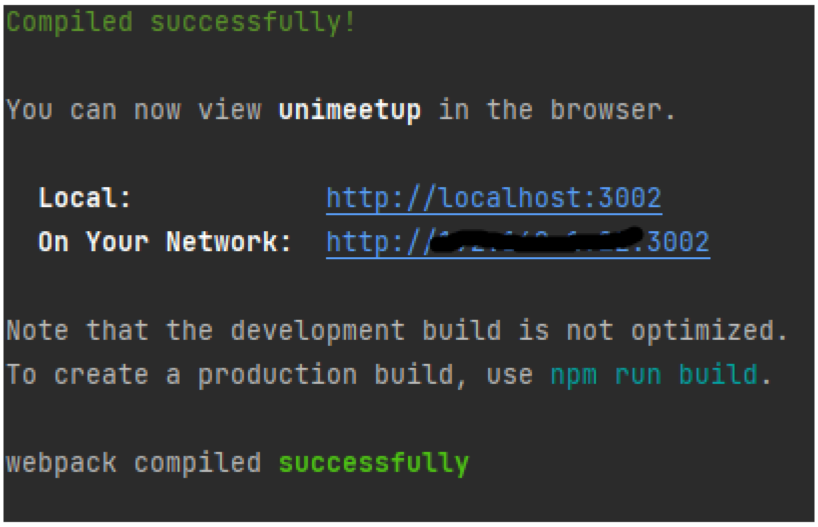


FIGURE 9: WEBSITE LOGO

5.5 Implementation

5.5.1 Development environment

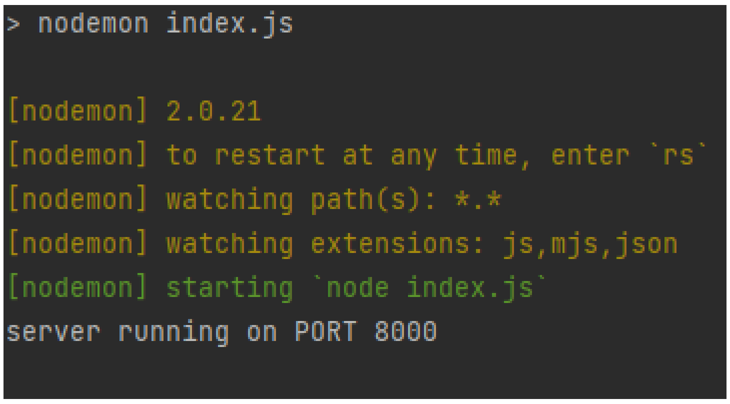
First, the author installed Node.js and Node Version Manager (NVM). The project was initialized by navigating to a chosen directory and executing **npx create-react-app unimeetup** in the WebStorm terminal. This command set up a development environment allowing for the use of latest JavaScript features and populating the project repository with all necessary files for a React project. Those files were moved to a /client directory for a better organization. Node Package Manager (NPM) helped installing required JavaScript packages and managing application dependencies. Npm is the default Node.js package manager.



```
Compiled successfully!

You can now view unimeetup in the browser.

  Local:            http://localhost:3002
  On Your Network:  http://[redacted].3002

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

FIGURE 10: NPM RUN START:FRONTEND IS THE COMMAND TO BE EXECUTED IN THE /CLIENT DIRECTORY. IT STARTS THE REACT.JS FRONT-END.

The author also created a /server directory to store the back-end files. An index.js file was created to write the main application logic. In the **package.json** file, a **start:backend** script was added. A package called **nodemon** was used to listen for changes in the index.js file. **Nodemon** automatically monitors for any changes in the source code and automatically restarts the server.



```
> nodemon index.js

[nodemon] 2.0.21
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
server running on PORT 8000
```

FIGURE 11: NPM RUN START:BACKEND SHOULD BE EXECUTED IN THE /SERVER DIRECTORY TO START THE NODE.JS APPLICATION

5.5.2 Programming language

As discussed in **Chapter 4.4.2**, JavaScript was the main programming language used for the implementation, with React and Node used as the primary frameworks. The author did not have much experience with JavaScript as he had used mainly Java and Python before so it was challenging to get familiar with a new programming language syntax in a timely manner. However, this was also an opportunity to learn a new skill.

5.5.3 Database connection and operations

The author used MongoDB Atlas. An account was created on the MongoDB website: *https://cloud.mongodb.com/user#/atlas/register/accountProfile*. A new free MongoDB cluster was set up after creating a new project. A user with read and write privileges to the database was created and used in the Node.js application to read and write the data from and to the database. Additionally, the current IP address of the author's machine was added to the IP whitelist so that the application can communicate with the server. Finally, MongoDB driver was installed on the application using the command **npm install mongodb** in the WebStorm terminal. MongoDB Atlas provided a secure way of storing data in the cloud, addressing the risk of a database attack. Each cluster in MongoDB Atlas has users with specific roles. These users must be configured while connecting to the database from the server. Only authenticated users with appropriate access to the database are able to perform corresponding database operations. In addition, Atlas only allows the connection to the database from the IP addresses that are listed in the project's whitelist. The IP address of the machine the project is deployed on was added to ensure a successful connection.

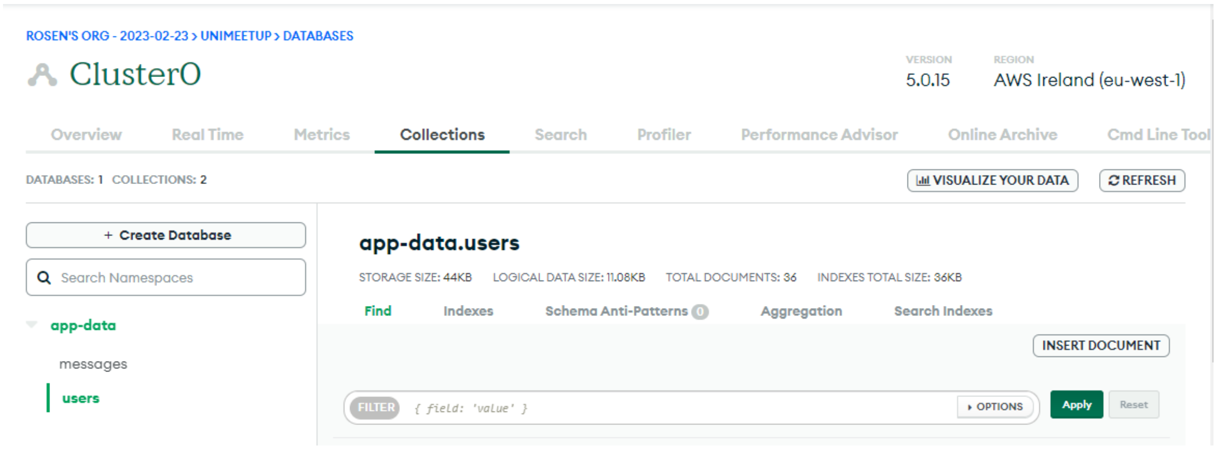
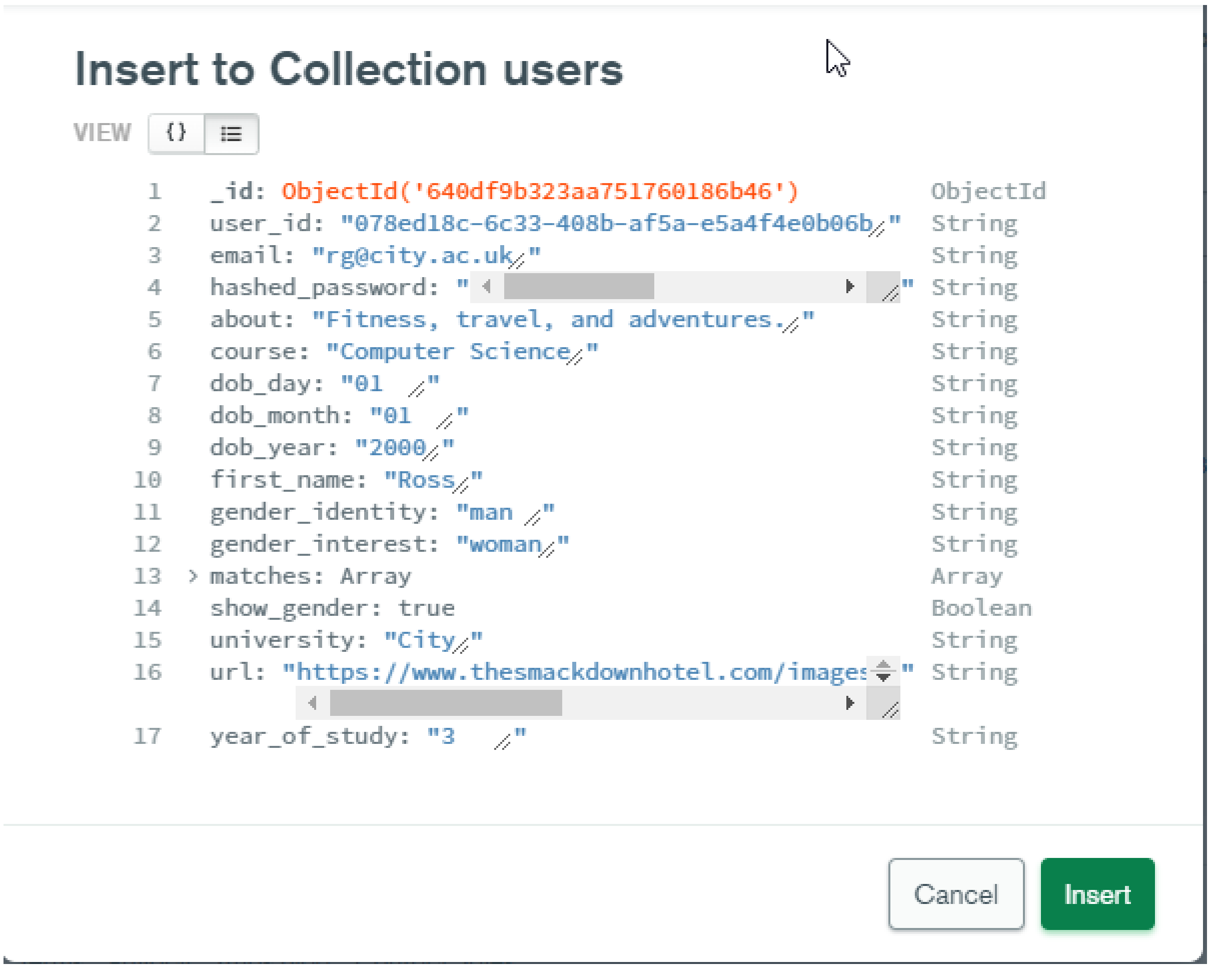


FIGURE 12: DATABASE

In the cluster, the author created a database with three main collections: user data, blocked users, and chat messages data. Every document in these collections has a unique object ID that can be used to identify objects.



Insert to Collection users

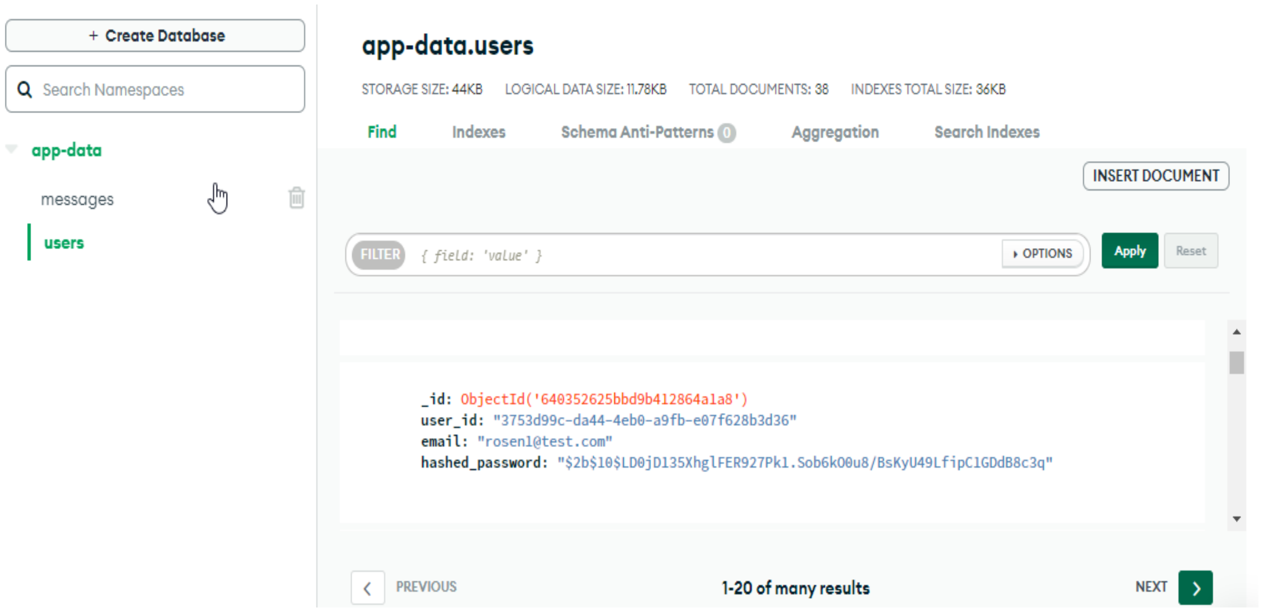
```
_id: ObjectId('640df9b323aa751760186b46')                  ObjectId
user_id: "078ed18c-6c33-408b-af5a-e5a4f4e0b06b"             String
email: "rg@city.ac.uk"                                      String
hashed_password: "                                          String
about: "Fitness, travel, and adventures."                   String
course: "Computer Science"                                  String
dob_day: "01   "                                            String
dob_month: "01   "                                          String
dob_year: "2000"                                            String
first_name: "Ross"                                          String
gender_identity: "man "                                     String
gender_interest: "woman"                                    String
> matches: Array                                            Array
show_gender: true                                           Boolean
university: "City"                                          String
url: "https://www.thesmackdownhotel.com/images              String
year_of_study: "3    "                                      String
```

FIGURE 13: AN EXAMPLE INSTANCE IN THE USERS COLLECTION CONTAINING USER DATA IN MONGODB

5.5.3.1 Registering new users: db.collection.insertOne()

```
const data = {
    user_id: generatedUserId,
    email: sanitizedEmail,
    hashed_password: hashedPassword
}

const insertedUser = await users.insertOne(data)
```

CODE SNIPPET 1: USER.INSERTONE(DATA)



```
_id: ObjectId('640352625bbd9b412864a1a8')
user_id: "3753d99c-da44-4eb0-a9fb-e07f628b3d36"
email: "rosen1@test.com"
hashed_password: "$2b$10$LD0jD135XhglFER927Pk1.Sob6kO0u8/BsKyU49LfipC1GDdB8c3q"
```

FIGURE 14: AN EXAMPLE USER INSERTED INTO THE COLLECTION AFTER COMPLETING THE REGISTRATION FORM

The **insertOne()** method in MongoDB made it possible to insert a single document into a collection. A document in MongoDB is equivalent to a row in a relational database. Collections are similar to tables in a relational database. **Code snippet 1** shows the logic behind inserting a new document into the user's collection. The documents that are inserted have three fields: **user_id, email, and hashed_password.** When a user submits a registration form on the website, MongoDB creates a new document in the users collection with the specified field-value pairs. If a collection does not exist, MongoDB will create it automatically upon insertion of a document.

The web application stores a hashed password in the database. Hashing passwords is an important security measure that helps protect user data. The library **bcrypt** was used to hash passwords.

```
const hashedPassword = await bcrypt.hash(password, 10)
```

CODE SNIPPET 2: BCRYPT.HASH FUNCTION

bcrypt.hash is a function used in cryptography to securely hash passwords or other sensitive information. It is designed to be slow and computationally expensive, which reduces risks from attackers using brute-force or dictionary attacks to exploit user credentials. The bcrypt.hash function takes a plaintext string (a password, as shown in **code snippet 2**) and a salt value as inputs, and outputs a hashed value that is difficult to reverse engineer. The salt value (in this case: 10) is a random string that is added to the input before hashing, which ensures that even if two users have the same password, their hashed values will be different. bcrypt.hash uses a "work factor" parameter, which determines the number of iterations of the hashing algorithm to perform. This work factor can be adjusted to make the hashing process slower and more resource-intensive, which increases the difficulty of cracking the password.

```
const correctPassword = await bcrypt.compare(password,
user.hashed_password)
```

CODE SNIPPET 3: COMPARE PASSWORDS. WHEN A USER LOGS IN, THE PASSWORD THEY ENTERED IS COMPARED WITH THE HASHED PASSWORD STORED IN THE DATABASE

5.5.3.2 Signing in as an already existing user: db.collection.findOne()

This method was used in the server logic as part of the sign in process for existing users. On the back-end, it allowed for the retrieval of the first document in a collection that matches a specified set of criteria. The criteria in this case were user's email and password which were provided as part of a form input when signing in.

5.5.3.3 Filling out profile details form: db.collection.updateOne()

After a successful registration, the database stores user's ID, email, and hashed password. The next step for a user after registering is to complete an onboarding form and fill in additional details about them. Therefore, the **updateOne()** method in MongoDB was very useful to update existing documents in the users collection (adding more information to a user's profile based on provided data). The **query** parameter is a document that specifies the criteria of the search (in this case, the search is performed by user id). Compared to a SQL database, it is similar to a WHERE clause. The **updateDocument** parameter is a document that specified how to update a matching document. The **$set** operator replaces the value of a field with the specified value. If the field does not exist, **$set** will add a new field with the specified value, provided that the new field does not violate a type constraint. As shown in **code snippet 4**, the author is providing multiple field-value pairs. **$set** will update/create each field.

```
const query = {user_id: formData.user_id}

const updateDocument = {
    $set: {
        first_name: formData.first_name,
        university: formData.university,
        course: formData.course,
        year_of_study: formData.year_of_study,
        dob_day: formData.dob_day,
        dob_month: formData.dob_month,
        dob_year: formData.dob_year,
        show_gender: formData.show_gender,
        gender_identity: formData.gender_identity,
        gender_interest: formData.gender_interest,
        url: formData.url,
        about: formData.about,
        matches: formData.matches
    },
}

const insertedUser = await users.updateOne(query, updateDocument)
```
CODE SNIPPET 4: UPDATEONE()

5.5.4 Features

5.5.4.1 Authentication

The authentication system was one of the main priorities in the implementation stage in order to verify the identity of a user accessing the web application. It was an essential element because no content of the application could be accessed without a user being logged in. Users register with a university e-mail address (one with .ac.uk suffix) and a password (including a confirm password field). If a valid e-mail address is provided and both passwords match, the back-end server sends a message to the provided e-mail for its validation.

```
<form onSubmit={handleSubmit}>
    <section>
        <label htmlFor="first_name">First Name</label>
        <input
            id="first_name"
            type='text'
            name="first_name"
            placeholder="First Name"
            required={true}
            value={formData.first_name}
            onChange={handleChange}
        />

        <label htmlFor="university">University</label>
        <input
            id="university"
            type='text'
            name="university"
            placeholder="University"
            required={true}
            value={formData.university}
            onChange={handleChange}
        />

        <label htmlFor="year_of_study">Year of Study</label>
        <input
            id="year_of_study"
            type='text'
            name="year_of_study"
            placeholder="Year of Study"
            required={true}
            value={formData.year_of_study}
            onChange={handleChange}
        />
```
CODE SNIPPET 5: REGISTRATION FORM IMPLEMENTATION

The Auth component uses a reusable function that sends asynchronous requests to authenticate users' credentials. An error message appears after a failed login attempt where there may have been issues with the credentials. If the credentials are correct, the login component uses the router view to display the dashboard page if the user is logging in, or the page for email verification if the user is registering.

```
const handleSubmit = async (e) => {
    e.preventDefault()
    try {
        if (isSignUp && (password !== confirmPassword)) {
            setError('Passwords do not match!')
            return
        }
        const response = await axios.post(`http://localhost:8000/$
        {
            isSignUp ? 'signup' : 'login'}`, { email, password }
        )

        setCookie('AuthToken', response.data.token)
        setCookie('UserId', response.data.userId)

        const success = response.status === 201
        if (success && isSignUp) { navigate(`/verify/$verificationToken}`);
        } else if { (success && !isSignUp) navigate ('/dashboard') }

        window.location.reload()

    } catch (error) {
        console.log(error)
    }
}
```
CODE SNIPPET 6: AUTH COMPONENT

Once the e-mail is verified and the registration is successful, the user is redirected to the onboarding page to complete their profile by providing details such as name, university, course, year of study, date of birth, and a profile picture.

```
const Signup = () => {
    const [cookies] = useCookies(null)
    const [formData, setFormData] = useState({
        user_id: cookies.UserId,
        first_name: "",
        dob_day: "",
        dob_month: "",
        dob_year: "",
        university: "",
        year_of_study: "",
        course: "",
        show_gender: false,
        gender_identity: "man",
        gender_interest: "woman",
        url: "",
        about: "",
        matches: []

    })
```
CODE SNIPPET 7: ONBOARDING FORM IMPLEMENTATION

When a user is authenticated, a JSON Web Token (JWT) is generated on the API server and is saved on the internal storage of the client's browser. Without JWT, the user will need to log in every time they move to another page, which would not be ideal in regards with user experience. JWTs are signed with a key when they are generated and then validated with a key upon receipt so the application can verify that they have not been modified

```
if (user && correctPassword) {
    const token = jwt.sign(user, email, {
        expiresIn: 60 * 24
    })
    res.status(201).json({token, userId: user.user_id})
}
```
CODE SNIPPET 8: SIGNING A JWT TOKEN WHEN LOGGING IN

5.5.4.2 Email verification

The email verification was configured with Amazon SES. The author reused code with changes in order to configure the Amazon SDK (Software Development Kit). The original source is available at: https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/Config.html.When a user signs up with their email address, they are sent an email to that address containing a verification link. The user clicks on the link, which takes them to the web application with a unique verification token in the URL. A GET request is sent to the server to verify the token. If the token is valid, the server responds with a status code of 200 and a success message is displayed to the user. The user is then redirected to the onboarding page. If the token is invalid or the server responds with an error, an appropriate error message is displayed to the user.

```
const verifyEmail = async () => {
        try {
            const response = await
axios.get(`http://localhost:8000/verify/${verificationToken}`);
            if (response.status === 200) {
                setVerificationResult('Email verified successfully. You
will now be redirected to the onboarding page.');
                setTimeout(() => {
                    navigate('/onboarding');
                }, 3000);
            }
        } catch (error) {
            if (error.response && error.response.status === 404) {
                setVerificationResult('Invalid verification link. Please
try again.');
            } else {
                setVerificationResult('An error occurred while verifying
your email. Please try again later.');
            }
        }
    };

    verifyEmail();
}, [verificationToken, navigate]);
```
CODE SNIPPET 9: VERIFYEMAIL FUNCTION ON THE VERIFYEMAIL PAGE

The **/verify/:verificationToken** endpoint in the backend is responsible for verifying a user's email address. When a user clicks on the email verification link, the verification token is sent as a URL parameter to this endpoint. The endpoint uses this token to query the database to find a matching user. If the user is found, the endpoint updates the user's record in the database by removing the verification token, which indicates that the user's email is now verified. The endpoint then sends a response with a success status code indicating that the email has been successfully verified. If the user is not found, the endpoint sends a response with a 404 status code indicating that the verification link is invalid.

```
app.get('/verify/:verificationToken', async (req, res) => {
    const client = new MongoClient(uri)

    const {verificationToken} = req.params

    try {
        await client.connect()
        const database = client.db('app-data')
        const users = database.collection('users')
        // Find user with matching verification token and remove token from
database
        const user = await users.findOne({verification_token:
verificationToken})

        if (user) {
            await users.updateOne({verification_token: verificationToken},
{$unset: {verification_token: ''}})
            return res.status(200).send('Email verified')
        }
        // Return error if verification token is invalid
        res.status(404).send('Invalid verification link')

    } catch (err) {
        console.log(err)
    } finally {
        await client.close()
    }
})
```

Code Snippet 10: /verify/:verificationToken endpoint

The **/verify-status/:userId** endpoint is responsible for checking whether a user's email is verified or not. This endpoint takes a user ID as a URL parameter and queries the database to find the corresponding user. If the user is found and their verification token is null, the endpoint sends a response with a 200 status code indicating that the user is verified. If the user is not found or their verification token is not null, the endpoint sends a response with a 404 status code indicating that the user is not verified.

```
app.get('/verify-status/:userId', async (req, res) => {
    const client = new MongoClient(uri)
    const {userId} = req.params
    try {
        await client.connect()
        const database = client.db('app-data')
        const users = database.collection('users')
        // Check if user exists in database and if their verification token
is null (indicating they are verified)
        const user = await users.findOne({user_id: userId})
        if (user && user.verification_token === null) {
            return res.status(200).send('User is verified')
        }
        res.status(404).send('User is not verified')

    } catch (err) {
        console.log(err)
    } finally {
        await client.close()
    }
})
```
CODE SNIPPET 11: /VERIFY-STATUS/:USERID ENDPOINT

These two endpoints work together to provide the functionality to verify a user's email address and provide feedback to the frontend based on the status of the verification process.

5.5.4.3 User profiles

The user profiles feature allows users to edit their profile details. The component retrieves the user's existing profile data using a useEffect hook and stores it in the component state using a useState hook. The user's profile details are displayed in an HTML form, and the user can edit their details and submit the form. The form data is sent to the server using an HTTP request using the axios library. The component also includes a cancel button to allow the user to go back to the dashboard without saving changes. The component handles loading and error states to provide a smooth user experience.

5.5.4.4 Explore student profiles

The explore feature was achieved by implementing card elements on which users can “swipe” left or right. Cards contain information about other users who could be a potential match. Swiping left on a user profile card indicates that the user is not interested in matching, as opposed to swiping right which indicates interest. If both users swipe right, there is a match, and they can connect via chat. To implement the user profile cards in the dashboard, a node package called **react-tinder-card** was used. This package helped create swipeable elements. It is built on top of the React library and uses the React Spring library for animations. The handleSwipe and handleOutOfFrame functions, as well as the card component, were implementing by adapting code from a demo of the react-tinder-card package. The demo helped the author implement a stack of cards (user profiles) that users can swipe left or right to indicate their preference. The demo also provided functions for handling swipes, as well as props for customizing the appearance and behavior of the cards. The original source is available at: https://github.com/3DJakob/react-tinder-card-demo/blob/master/src/examples/Simple.js

```
const handleSwipe = (direction, swipedUserId) => {
    if (direction === 'right') {
        updateMatches(swipedUserId)
    }
    setLastDirection(direction)
}


const handleOutOfFrame = (name) => {
    console.log(name + ' left the screen!')
}
```

CODE SNIPPET 12: THE HANDLESWIPE FUNCTION LOGS THE DIRECTION OF THE SWIPE AND THE ID OF THE USER THAT WAS SWIPED

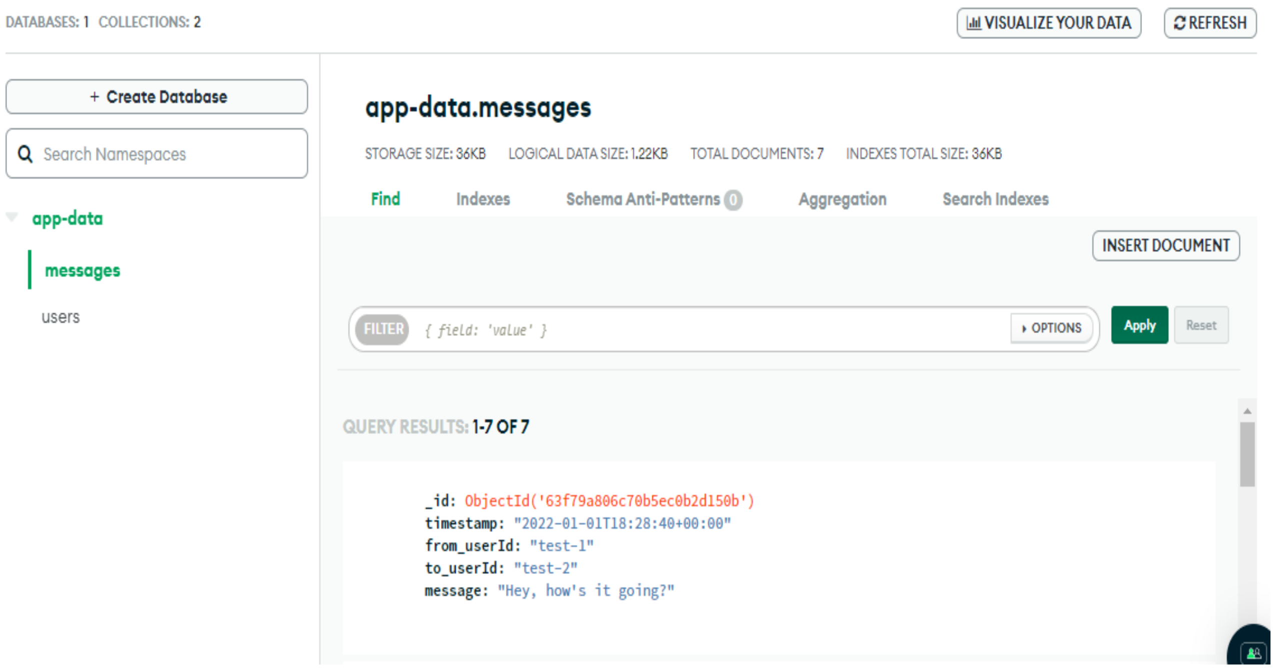
```
<TinderCard
    className="swipe"
    key={genderedUser.user_id}
    onSwipe={(dir) => handleSwipe(dir, genderedUser.user_id)}
    onCardLeftScreen={() => handleOutOfFrame(genderedUser.first_name)}>
    <div
        style={{backgroundImage: "url(" + genderedUser.url + ")"}}
        className="card">
        <h3>{genderedUser.first_name}</h3>
        <h4>{genderedUser.university}</h4>
        <h5>{genderedUser.course + ", Stage " + genderedUser.year_of_study
}</h5>
        <p>{genderedUser.about}</p>

    </div>
</TinderCard>
```

CODE SNIPPET 13: HANDLING SWIPES

In **code snippet 13**, the author demonstrates the reused **onCardLeftScreen** callback to remove swiped cards from the stack. The reused card component includes a state variable to keep track of the current stack of cards. The **handleOutOfFrame** function removes the card at the specified index from the cards array.

5.5.4.5 Real-time chat

Creating the chat functionality for the web application involved building user interface elements that allow users to enter messages and send them to the chat, including a text area element that users can fill out to send messages. This required creating a text input field and a submit button. The author created a chat log where users can view past messages. This involved storing messages in a database collection so that they can be retrieved and displayed in the chat log.


DATABASES: 1 COLLECTIONS: 2

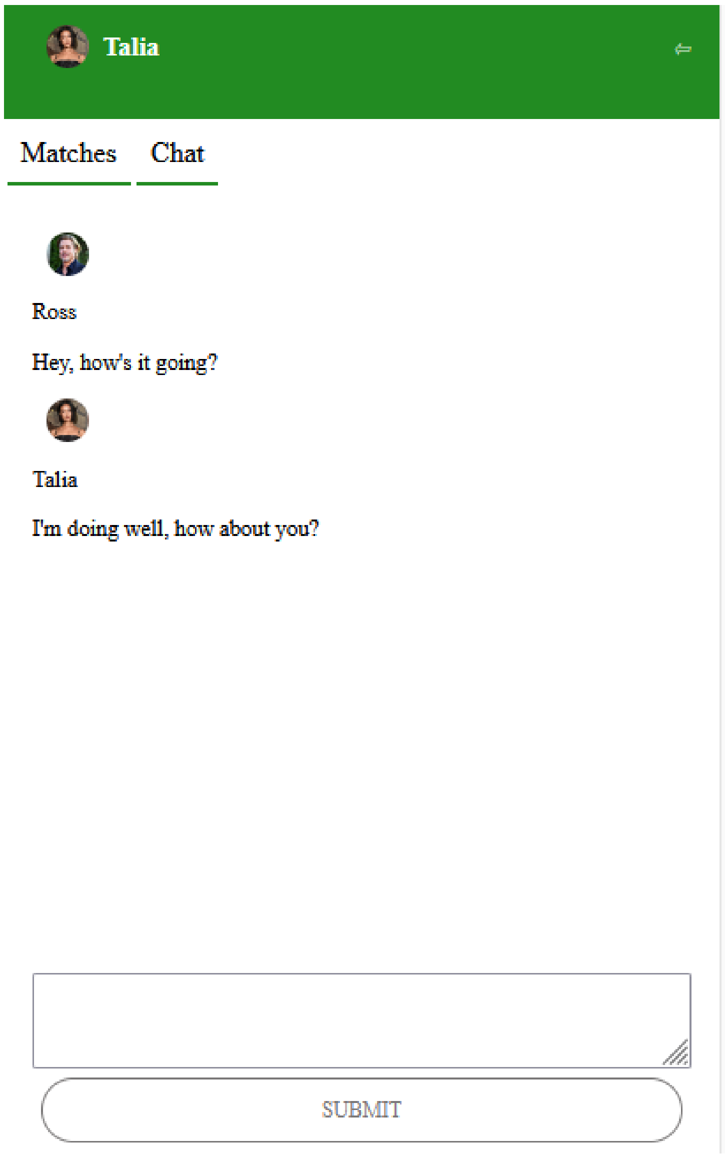
app-data.messages

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1.22KB TOTAL DOCUMENTS: 7 INDEXES TOTAL SIZE: 36KB

QUERY RESULTS: 1-7 OF 7

```
_id: ObjectId('63f79a806c70b5ec0b2d150b')
timestamp: "2022-01-01T18:28:40+00:00"
from_userId: "test-1"
to_userId: "test-2"
message: "Hey, how's it going?"
```

FIGURE 15: MESSAGES COLLECTION IN MONGODB



Ross

Hey, how's it going?

Talia

I'm doing well, how about you?

FIGURE 16: CHAT USER INTERFACE

5.5.4.6 Block a user

The ability to block users was essential in protecting users from unwanted and potentially harmful communication. As already discussed, in an online dating/communication context, users may receive unwanted messages or be subjected to abusive behavior. Implementing a feature to allow users to block other users helped to prevent such instances and promoted a safer and more positive user experience. By providing safety features such as a block function, the author demonstrated that he takes the safety and security of users seriously. This helped building trust and confidence in the platform. Furthermore, by implementing this safety safety feature, the website encouraged users to act responsibly and respectfully towards other users. It sent a clear message that inappropriate behavior will not be tolerated, which can helped deter such behavior and promote a more positive online experience for all users. With regards to compliance, many countries have laws and regulations in place that require social platforms to provide certain safety features which ensure that the software complies with legal requirements and avoids potential legal liabilities.

The author achieved this functionality by creating a function to block a user which is triggered when pressing a **Block User** button as part of the front-end. Two endpoints were also created in the Node.js back-end to send blocked users to the database and retrieve a list of blocked users, respectively.

```
app.post('/block', async (req, res) => {
    const client = new MongoClient(uri)
    const blockingUserId = req.body.blockingUserId
    const blockedUserId = req.body.blockedUserId

    try {
        await client.connect()
        const database = client.db('app-data')
        const blockedUsers = database.collection('blockedUsers')

        const insertedBlockedUser = await blockedUsers.insertOne({
            blockingUserId,
            blockedUserId,
            timestamp: new Date().toISOString(),
        })

        res.send(insertedBlockedUser)
    } finally {
        await client.close()
    }
})
```

CODE SNIPPET 14: POST METHOD AS PART OF BLOCKING FUNCTIONALITY

This method in **code snippet 14** handles the POST request to add a new user to the list of blocked users. The endpoint takes in two parameters in the request body: the **blockingUserId** and the **blockedUserId**, which identify the user who is doing the blocking and the user who is being blocked, respectively.

```
app.get('/blocked', async (req, res) => {
    const { blockingUserId } = req.query;
    const client = new MongoClient(uri);
    try {
        await client.connect();
        const database = client.db('app-data');
        const blockedUsers = database.collection('blockedUsers');

        const query = { blockingUserId: blockingUserId };
        const blockedUserDocs = await blockedUsers.find(query).toArray();
        const blockedUserIds = blockedUserDocs.map(doc =>
doc.blockedUserId);
        res.send(blockedUserIds);
    } finally {
        await client.close();
    }
})
```
CODE SNIPPET 15: GET METHOD AS PART OF BLOCKING FUNCTIONALITY

In **code snippet 15**, the function handles the GET request to retrieve a list of blocked user IDs for a given **blockingUserId**. The endpoint takes in a single query parameter: **blockingUserId**, which identifies the user whose list of blocked users should be retrieved.

5.5.4.7 Icebreaker questions

The feature consists of a pre-written list of open-ended questions that users can choose to send to their matches to initiate a conversation. To access the icebreaker questions, users can navigate to the chat interface and select the 'Get an icebreaker question' option. From there, the chat input field will be populated with a random question as an option for a conversation starter.

5.5.5 Software reuse

List of reused software packages:

- express
- mongodb
- dotenv
- bcrypt
- cors
- uuid
- jsonwebtoken
- aws-sdk
- nodemailer
- axios

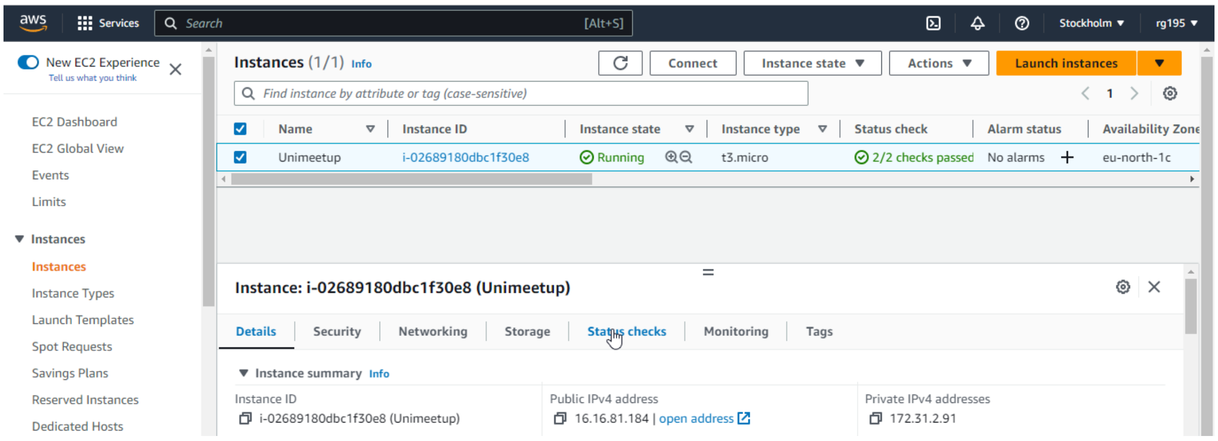
- react
- react-tinder-card
- react-cookie
- react-router-dom

The list complete list is available in **Appendix B**, including more information about the purpose of each package under **7.2.1 Software libraries**. They are also available on https://www.npmjs.com/

Reused code summary and original sources are available in **Appendix B** under **7.2.2 Reused code**.

5.6 Web hosting

To make the website accessible online, the author first created an Amazon EC2 (Elastic Compute Cloud) virtual machine instance running the Ubuntu operating system and connected to it using SSH (secure shell). Then, the author stored the web application code in Amazon S3 (Simple Storage Service) and cloned it to the instance from there. The required dependencies needed for the application to run on the instance were installed. Next, the author configured the security groups of the instance to allow incoming traffic on the ports used by the application and associated an Elastic IP address with the instance to give it a static IP address. To ensure that application runs continuously on the instance, PM2 (a process manager) was used to manage server processes. The online hosted version of the software can be accessed at: http://16.16.81.184:3000/



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
Unimeetup	i-02689180dbc1f30e8	Running	t3.micro	2/2 checks passed	No alarms	eu-north-1c

Instance: i-02689180dbc1f30e8 (Unimeetup)

Instance summary Info

Instance ID	Public IPv4 address	Private IPv4 addresses
i-02689180dbc1f30e8 (Unimeetup)	16.16.81.184 | open address	172.31.2.91

FIGURE 17: AMAZON EC2 VIRTUAL MACHINE INSTANCE HOSTING THE WEB APPLICATION

5.7 Testing

The test plan described the resources, scope, approach and schedule of proposed test activities. It identified the features to be tested, who executed particular tasks, and also identifying any risks. Test cases were designed by the author with the expected set of inputs as parameters, execution environments, and expected outputs established for different objectives throughout the testing phase.

Unit Testing

Unit testing was required to test specific software units and sets of related units. The unit testing was based on white box testing techniques, where the author verified and validated that the code works as expected. The author performed unit testing on every piece of functionality implemented in this project to ensure that the functionality works as intended.

Integration Testing

Integration test involved combining individual software components and perform test to assess the collaboration between them. Just because an individual component works as a unit, does not mean that it can operate when integrated with other components. The author utilized testing methods to validate that individual components work together when they are combined into a single one. The integration testing of this software took input from the unit element that has been unit tested, combined them, and then delivered the tested result as output.

Functional and System Testing

Functional and system testing were incorporated to inspect the high-level design requirements and plan the various test cases to make sure that the components of the web application work as expected. The functional testing covered how well the software executed the operations it is meant to achieve. These operations included registering, signing in, data manipulation, integration, user interfaces. The system testing involved testing the software in different environments to ensure the software works in typical client situations with several versions of operating systems.

Regression Testing

During the project testing phase, regression test cases were run. The regression testing is selective retesting of software components to verify that changes have not caused unexpected effects and that the software still conforms with its quantified requirements.

Figure 17 below summarised the four testing types discussed in this chapter.

Test Type	**Scope**	**Tester**
Unit	Small units of code/individual components	Author
Integration	Multiple units/components	Author
System/functional	Complete software	Author
Regression	Any of the above	Author

FIGURE 18: TESTING TYPES

Test report

The test report provides detailed test cases. A test case contains test procedures, test data, expected result and actual result.

Test case ID	**1**
Test case summary	Register a new user
Prerequisites	A PC or laptop device with an internet connection and a browser
Test procedure	1. Open the web application 2. Click on the **Register** button on the Home page 3. Fill out a form with required information
Test data	Username and password for a fictional user
Expected result	1. The user should be taken to the onboarding page upon successful registration and email verification 2. An error should be flagged if the information provided in the form is not as expected
Actual result	1. The application sent a verification email successfully 2. Verification was successful upon clicking on the email link 3. The user was taken to the onboarding page 4. Error messages were displayed when trying to register with an invalid **email address** 5. An error was displayed if the fields **password** and **confirm password** do not match
Status	Pass
Tested by	Author
Date	5th April 2023

FIGURE 19: TEST EXAMPLE

You can find the complete suite of test cases in **Appendix E**.

Chapter 5: Conclusions and Discussion

To sum up, the project goal was to create a social networking platform, enabling UK students to connect. The project objectives were achieved by designing and implementing a software system with an interactive user interface and a user-friendly layout which provides a secure website experience. Core features such as secure login, matching algorithm, and private communication were developed as the building blocks of the system. By reusing software packages/libraries where appropriate, the author reduced the likelihood of introducing new errors or bugs into the project and was able to focus on building new features, fixing bugs, and improving the overall project quality, rather than having to start from scratch for each functionality.

Potential future work on the application may include developing extra functionalities to improve it such as a feature for audio and video conferencing in addition to the already available chat messaging. Another idea could be to implement photo verification in order to ensure that users are who they say they are. With a photo verification feature, users will have to submit a selfie or another photo that will then be verified by a moderator. Furthermore, building a version of the application for mobile devices can also contribute to more success for the software.

On a personal level, the project was a great way to utilize knowledge acquired throughout the Computer Science course as well as develop new skills and gain practical experience in application development. As mentioned in **Chapter 5.5.2**, it was challenging to work with new tools across the technical stack and learn how to use them effectively in a timely manner. However, this was also a rewarding experience which allowed the author to acquire a wide array of qualities that will help with his long-term professional goals. On top of improving technical capabilities, the author has also enhanced project management, time management, critical thinking, self-awareness, and accountability qualities. The author has learnt to never approach a project as ‘too complex’ and be open to learning and working with new technologies. Another important lesson learnt was the fact that things do not always go as planned and that is the reason why it is important to be adaptable.

6. Reference List

John F. Dooley (2017). Software Development, Design and Coding. Berkeley, California: Apress.

Wikipedia (n.d.) JavaScript [Online]. Available at: https://en.wikipedia.org/wiki/JavaScript *(Accessed: 7th February 2023)*

StackOverflow (2022) Developer Survey [Online] Available at: https://survey.stackoverflow.co/2022/ (Accessed: 7th February 2023)

JetBrains (n.d.) WebStorm [Online]. Available at: https://www.jetbrains.com/webstorm/ *(Accessed: 7th February 2023)*

MongoDB (2019) MongoDB [Online]. Available at: https://www.mongodb.com/ *(Accessed: 8th February 2023)*

JSON Web Tokens (n.d.) JWT [Online]. Available at: https://jwt.io/ *(Accessed: 8th February 2023)*

Cprime (2023) Agile [Online]. Available at: https://www.cprime.com/agile/ *(Accessed: 17th February 2023)*

Visual Paradigm (n.d.) What is UML [Online]. Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/ *(Accessed: 17th February 2023)*

Kinsta (2022) What is GitHub [Online]. Available at: https://kinsta.com/knowledgebase/what-is-github/ *(Accessed: 17th February 2023)*

Nick Heath (2013) How replacing Java with JavaScript is paying of for PayPal [Online]. Available at: https://www.zdnet.com/article/how-replacing-java-with-javascript-is-paying-off-for-paypal/ *(Accessed: 8th March 2023)*

NPM (2022) React Tinder Card [Online]. Available at: https://www.npmjs.com/package/react-tinder-card *(Accessed: 13th March 2023)*

7. Appendices

7.1 Appendix A Project Definition Document

BSc (Hons) Computer Science

City, University of London

Project Definition Document

UniMeetUp: A dynamic web application for university students to meet new people

Proposed by: Rosen Georgiev
Student e-mail address: Rosen.Georgiev@city.ac.uk

Consultant: Laure Daviaud
Consultant e-mail address: Laure.Daviaud@city.ac.uk

Word count: 1009

Problem to be solved:

It can be difficult to meet new people at university especially if you have just started your first year. A lot of students, including international students, have left their friends and family behind and moved to a new place to study. If a student has a hard time making friends or has underdeveloped social skills, then they may also come up against any of the difficulties below:

- Academic problems: they may feel stressed and find their courses way harder than expected. They may realize they don't have good study or time management skills.
- Homesickness: can make students feel down and unmotivated to go out or meet anyone.
- Discomfort caused by the size of their new community: students from small towns may feel uncomfortable if their school is in a big city; students from bigger cities may feel cooped up and bored if their college is in a much smaller community.

Project Objectives:

Create a dynamic web application to enable university students to meet new people from their university or other UK universities nearby.

Students will be verified because they will be required to sign up using their @ac.uk email address. Student profiles will display photos of themselves, university name, year of study, course, and interests. Users will be able to match with others based on a mutual-like system. This would ensure a user can receive messages only from people they have expressed an interest in. Users can decide whether to use the app for chat, making friends, dating, or business.

Project Beneficiaries:

University students will benefit from having a software built specifically for them to connect. There are other similar web applications functioning on mutual-like, match, and chat basis such as Tinder and Bumble, but they are not designed solely with students in mind.

Work Plan:

High-level overview of implementation steps:

- Define application requirements.
- Design the web application, including components architecture and database design.
- Implement frontend: create user-friendly interface using HTML, CSS, JavaScript.
- Implement backend: create server-side logic using React.js and Node.js.

- The database system used will be MongoDB.
- Test the web application to identify bugs and issues in the development process so that I can fix them and ensure the application is functioning as required.
- I will use GitHub as a version control system to track and manage changes to software code. That way, I will be able to revert back to an earlier version of the code and make changes with minimum disruption if necessary.

Software will be developed following object-oriented principles of encapsulation, abstraction and code reusability. The code will be clearly commented to allow an external reviewer to understand its purpose. The application will be consistent for the end user. I plan to develop the basic functionality first, test it and then continuously improve it by adding additional features and using repeated evaluation of the system. I will follow good security practices such as not including hard coded information about specific settings in the runtime environment (e.g. file paths, IP addresses, e-mail addresses, host names).

A more detailed overview of implementation tasks:

1. Create a base file structure for both the front-end and the back-end. Create a homepage, add sign up and sign in buttons.
   Start date: 12th February 2023, end date: 19th February 2023
2. Connect the API to use MongoDB for storing users' details. Create user profile pages. Output: users will be able to register, login, update settings, and check other university student's profiles with information about their university, course, and year of study.
   Start date: 20th February 2023, end date: 5th March 2023
3. Implement JWT (JSON Web Tokens) authentication used for identification of users.
   Start date: 6th March 2023, end date: 12th March 2023
4. Add a feature that lists users and shows their profile pages to each other. Add like/dislike feature that tracks how users like each other.
   Start date: 13th March 2023, end date: 26th March 2023
5. Implement a matching algorithm for users to find each other.
   Start date: 27th March 2023, end date: 2nd April 2023
6. Add WebSockets to allow for communication between users who have matched and real-time chatting features.
   Start date: 3rd April 2023, end date: 9th April 2023

7. Test, clean up, potential additional functionality
   Start date: 10th April 2023, end date: 25th April 2023

Note: start/end dates for the specific functionalities may change throughout the project depending on progress.

Project Risks

Risks to your project:

Working with new technology: I have not implemented a real-time communication feature between two clients for a web application before. Therefore, creating this functionality can take me more time than expected and delay progress. I will plan for this in advance and spend the required time for upskilling and familiarizing myself with WebSockets.

Delay in report writing: it will be challenging to work on the project implementation and the report, so I will need to plan and prioritize accordingly. Throughout my academic career and previous work experience, I have developed good time management skills which will help me stay on track.

Risks that your project poses to others:

Data privacy: when creating an account on the app app, personal information such as date of birth, age, profile picture, university name, and course need to be provided so that the app can find the most-fitting matches for the user thereby providing an opportunity to communicate with others. From a user's perspective, giving away personal information can be a risk. To address this, users will be presented with the information about how the app handles data and will have a choice whether to accept terms and conditions when registering.

Potential criminal activities: due to the possibility of anonymity in the online world, app users could find themselves communicating with a fabricated profile. This can cause potential dangers for the victims to arise, such as coaxing them into sharing sensitive information or agreeing to meet in person to discover too late that they are someone completely different.

Online harassment: having a chat feature may lead to users becoming victims of online bullying when communicating with others.

Research Ethics Review Form: BSc, MSc and MA Projects

Computer Science Research Ethics Committee (CSREC)

http://www.city.ac.uk/department-computer-science/research-ethics

Undergraduate and postgraduate students undertaking their final project in the Department of Computer Science are required to consider the ethics of their project work and to ensure that it complies with research ethics guidelines. In some cases, a project will need approval from an ethics committee before it can proceed. Usually, but not always, this will be because the student is involving other people ("participants") in the project.

In order to ensure that appropriate consideration is given to ethical issues, all students must complete this form and attach it to their project proposal document. There are two parts:

PART A: Ethics Checklist. All students must complete this part. The checklist identifies whether the project requires ethical approval and, if so, where to apply for approval.

PART B: Ethics Proportionate Review Form. Students who have answered "no" to all questions in A1, A2 and A3 and "yes" to question 4 in A4 in the ethics checklist must complete this part. The project supervisor has delegated authority to provide approval in such cases that are considered to involve MINIMAL risk. The approval may be ***provisional*** – *identifying the planned research as* likely to involve MINIMAL RISK. In such cases you must additionally seek ***full approval*** from the supervisor as the project progresses and details are established. ***Full approval*** must be acquired in writing, before beginning the planned research.

A.1 If you answer YES to any of the questions in this block, you must apply to an appropriate external ethics committee for approval and log this approval as an External Application through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
1.1	Does your research require approval from the National Research Ethics Service (NRES)? *e.g. because you are recruiting current NHS patients or staff?* *If you are unsure try - https://www.hra.nhs.uk/approvals-amendments/what-approvals-do-i-need/*	**NO**
1.2	Will you recruit participants who fall under the auspices of the Mental Capacity Act? *Such research needs to be approved by an external ethics committee such as NRES or the Social Care Research Ethics Committee - http://www.scie.org.uk/research/ethics-committee/*	**NO**
1.3	Will you recruit any participants who are currently under the auspices of the Criminal Justice System, for example, but not limited to, people on remand, prisoners and those on probation? *Such research needs to be authorised by the ethics approval system of the National Offender Management Service.*	**NO**
A.2 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee, you must apply for approval from the Senate Research Ethics Committee (SREC) through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*

2.1	Does your research involve participants who are unable to give informed consent? *For example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf.*	**NO**
2.2	Is there a risk that your research might lead to disclosures from participants concerning their involvement in illegal activities?	**NO**
2.3	Is there a risk that obscene and or illegal material may need to be accessed for your research study (including online content and other material)?	**NO**
2.4	Does your project involve participants disclosing information about special category or sensitive subjects? *For example, but not limited to: racial or ethnic origin; political opinions; religious beliefs; trade union membership; physical or mental health; sexual life; criminal offences and proceedings*	**NO**
2.5	Does your research involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning that affects the area in which you will study? *Please check the latest guidance from the FCO - http://www.fco.gov.uk/en/*	**NO**
2.6	Does your research involve invasive or intrusive procedures? *These may include, but are not limited to, electrical stimulation, heat, cold or bruising.*	**NO**
2.7	Does your research involve animals?	**NO**
2.8	Does your research involve the administration of drugs, placebos or other substances to study participants?	**NO**
A.3 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee or the SREC, you must apply for approval from the Computer Science Research Ethics Committee (CSREC) through Research Ethics Online - https://ethics.city.ac.uk/ **Depending on the level of risk associated with your application, it may be referred to the Senate Research Ethics Committee.**		*Delete as appropriate*
3.1	Does your research involve participants who are under the age of 18?	**NO**
3.2	Does your research involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? *This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.*	**NO**
3.3	Are participants recruited because they are staff or students of City, University of London? *For example, students studying on a particular course or module.* *If yes, then approval is also required from the Head of Department or Programme Director.*	**NO**

3.4	Does your research involve intentional deception of participants?	**NO**
3.5	Does your research involve participants taking part without their informed consent?	**NO**
3.5	Is the risk posed to participants greater than that in normal working life?	**NO**
3.7	Is the risk posed to you, the researcher(s), greater than that in normal working life?	**NO**
A.4 If you answer YES to the following question and your answers to all other questions in sections A1, A2 and A3 are NO, then your project is deemed to be of MINIMAL RISK. **If this is the case, then you can apply for approval through your supervisor under PROPORTIONATE REVIEW. You do so by completing PART B of this form.** **If you have answered NO to all questions on this form, then your project does not require ethical approval. You should submit and retain this form as evidence of this.**		*Delete as appropriate*
4	Does your project involve human participants or their identifiable personal data? *For example, as interviewees, respondents to a survey or participants in testing.*	**NO**

References

Dr. Richard Hall Thayer & Dr. Merlin Dorfman (2012): Software Engineering Essentials, Volume I: The Development Process: Volume 1

John F. Dooley (2017): Software Development, Design and Coding

Nafaâ Jabeur (2013): Mobile Social Networking Applications research paper

7.2 Appendix B Reuse Summary

7.2.1 Software libraries

- express: Express.js was used for routing to define how the application's endpoints (URIs) respond to client requests
  https://www.npmjs.com/package/express
- mongodb: the official MongoDB driver for Node.js
  https://www.npmjs.com/package/mongodb
- dotenv: to store variables secretly. Dotenv is a zero-dependency module that loads environment variables from a .env file into process.env. The author stored configuration parameters in a .env file to improve security
  https://www.npmjs.com/package/dotenv
- bcrypt: to hash user passwords. It uses a password-hashing function that is based on the Blowfish cipher. The Blowfish cipher is a symmetric block cipher that provides the best encryption rate in the industry; thus, it can be used in cipher suites and encryption systems
  https://www.npmjs.com/package/bcrypt
- cors: for communication with backend. Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources
  https://www.npmjs.com/package/cors
- uuid: to generate unique identifiers. The uuid package provided functionality for generating cryptographically secure standard universally unique identifiers
  https://www.npmjs.com/package/uuid
- jsonwebtoken: required for synchronous sign-in (when an existing user logs into their account)
  https://www.npmjs.com/package/jsonwebtoken
- aws-sdk: the AWS Software Development Kit is a collection of libraries and tools provided by Amazon Web Services to help developers build applications that interact with AWS services. The SDK provides a set of APIs and programming tools to allow for a seamless integration of AWS services. The package supports a wide range of programming languages, JavaScript and can be used to manage resources on AWS. It was required for the implementation of Amazon SES (Simple Email Service) which is

a part of the authentication mechanism and is used to send a verification email to a registering user
https://www.npmjs.com/package/aws-sdk

- nodemailer: made it possible to send email messages from the Node.js backend
https://www.npmjs.com/package/nodemailer
- axios: allowed the author to make HTTP requests from the web application
https://www.npmjs.com/package/axios
- react: a library used for building the user interface
https://www.npmjs.com/package/react
- react-tinder-card: provided a simple and customizable interface for creating swipeable cards that respond to user gestures, such as swiping left or right, which was an essential feature for the project
https://www.npmjs.com/package/react-tinder-card
- react-cookie: utilized for handling cookies
https://www.npmjs.com/package/react-cookie
- react-router-dom: used to implement client-side routing
https://www.npmjs.com/package/react-router-dom

7.2.2 Reused code

7.2.2.1 package.json files (generated by npm)

The files **client/package.json** and **server/package.json** were generated by **npm** (node package manager). The **package.json** files specify project dependencies, scripts, and other important information used to manage and build the project for the front-end and back-end, respectively.

When executing **npm install,** npm looks for a **package.json** file in the current directory and installs all dependencies specified in that file. These dependencies are installed in a **node_modules** directory. Npm also generates a **package-lock.json** file, which is used to keep track of the exact version of the dependencies.

7.2.2.2 node_modules (reused libraries)

The **node_modules** folder contains a big number of sub-folders and files with the project dependencies. The author is reusing the code of the libraries installed in the **client/node_modules** and **server/node_modules** directories (these are the libraries described in **Chapter 7.2.1 Software Libraries).** These libraries are maintained by other developers and are made available for the author to use under an open-source license.

7.2.2.3 Files that include reused code

7.2.2.3.1 CLIENT/SRC/WEBPAGES/DASHBOARD.JS

Part of the code in the file that was reused with changes, as commented in the file.

The original source is available at https://github.com/3DJakob/react-tinder-card-demo/blob/master/src/examples/Simple.js

7.2.2.3.2 CLIENT/SRC/INDEX.CSS

Part of the code in the file that was reused with changes, as commented in the file:

The original source is available at https://github.com/3DJakob/react-tinder-card-demo/blob/master/src/App.css

7.2.2.3.3 CLIENT/SRC/PICTURES/STUDENTS.JPG

Photo source: https://www.qs.com/wp-content/uploads/2015/05/students-or-teenagers-with-laptop-and-tablet-computers.jpg

7.2.2.3.4 SERVER/INDEX.JS

The reused part of the file includes configuring the AWS SDK for using Amazon SES.

Original source: https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/Config.html

7.2.3.4.5 Images for creating fictional users for testing

Sources:

Photo 1

Photo 2

Photo 3

Photo 4

Photo 5

Photo 6

Photo 7

Photo 8

Photo 9

7.3 Appendix C Requirements

7.3.1 Volere requirements

Requirement ID: 1	**Requirement Type: Functional**
Description: The website shall allow its visitors to create accounts.	
Rationale: Individual accounts are required for a personalized user experience.	
Source: Analysis of existing software	
Fit Criteria: Users will receive a validation email to confirm their registration on the application	
Customer Satisfaction: 5	**Customer Dissatisfaction: 4**
Priority: Essential	**Conflicts:** None
Supporting Material: None	Volere Source: Atlantic Systems Guild
History: New requirement	

Requirement ID: 2	**Requirement Type: Functional**
Description: The system will allow registered users to sign in.	
Rationale: The website functionalities are available only for registered users. The website is created for current students and they will have to validate their student e-mail address. That way, I can ensure the website target audience meets the requirements.	
Source: Analysis of existing software.	
Fit Criteria: After providing valid credentials, the user is redirected to the dashboard.	
Customer Satisfaction: 5	**Customer Dissatisfaction: 4**
Priority: Essential	**Conflicts:** None
Supporting Material: None	Volere Source: Atlantic Systems Guild
History: New requirement	

Requirement ID: 3	**Requirement Type: Functional**
Description: The application shall have a matching algorithm for users.	
Rationale: In order to connect and chat, users need to match with others based on mutual interest.	
Source: Analysis of existing software	
Fit Criteria: If two users both swipe right on each other's profiles, they will be able to start a conversation.	
Customer Satisfaction: 5	**Customer Dissatisfaction: 5**
Priority: Essential	**Conflicts:** None
Supporting Material: None	Volere
History: New requirement	Source: Atlantic Systems Guild

Requirement ID: 4	**Requirement Type: Functional**
Description: Users who have matched shall be able to send messages between each other.	
Rationale: A private messaging system is required for users to communicate.	
Source: Analysis of existing software	
Fit Criteria: After a user sends a message, the other user will be able to see it and reply.	
Customer Satisfaction: 4	**Customer Dissatisfaction: 3**
Priority: Essential	**Conflicts:** None
Supporting Material: None	Volere
History: New requirement	Source: Atlantic Systems Guild

Requirement ID: 5	**Requirement Type: Functional**
Description: Users shall be able to use a profile picture.	
Rationale: Essential feature as part of completing a user profile.	
Source: Analysis of existing software	
Fit Criteria: Users will be able to upload a profile picture as part of the onboarding process.	
Customer Satisfaction: 4	**Customer Dissatisfaction: 3**
Priority: Essential	**Conflicts:** None
Supporting Material: None	Volere Source: Atlantic Systems Guild
History: New requirement	

Requirement ID: 6	**Requirement Type: Functional**
Description: Users shall be able to explore other user's profiles and see information about them such as name, university, course, year of study.	
Rationale: A core feature required for users to connect.	
Source: Author	
Fit Criteria: Users will be shown other users' profiles based on a matching algorithm and gender preferences.	
Customer Satisfaction: 4	**Customer Dissatisfaction: 3**
Priority: Highly Important	**Conflicts:** None
Supporting Material: None	Volere Source: Atlantic Systems Guild
History: New requirement	

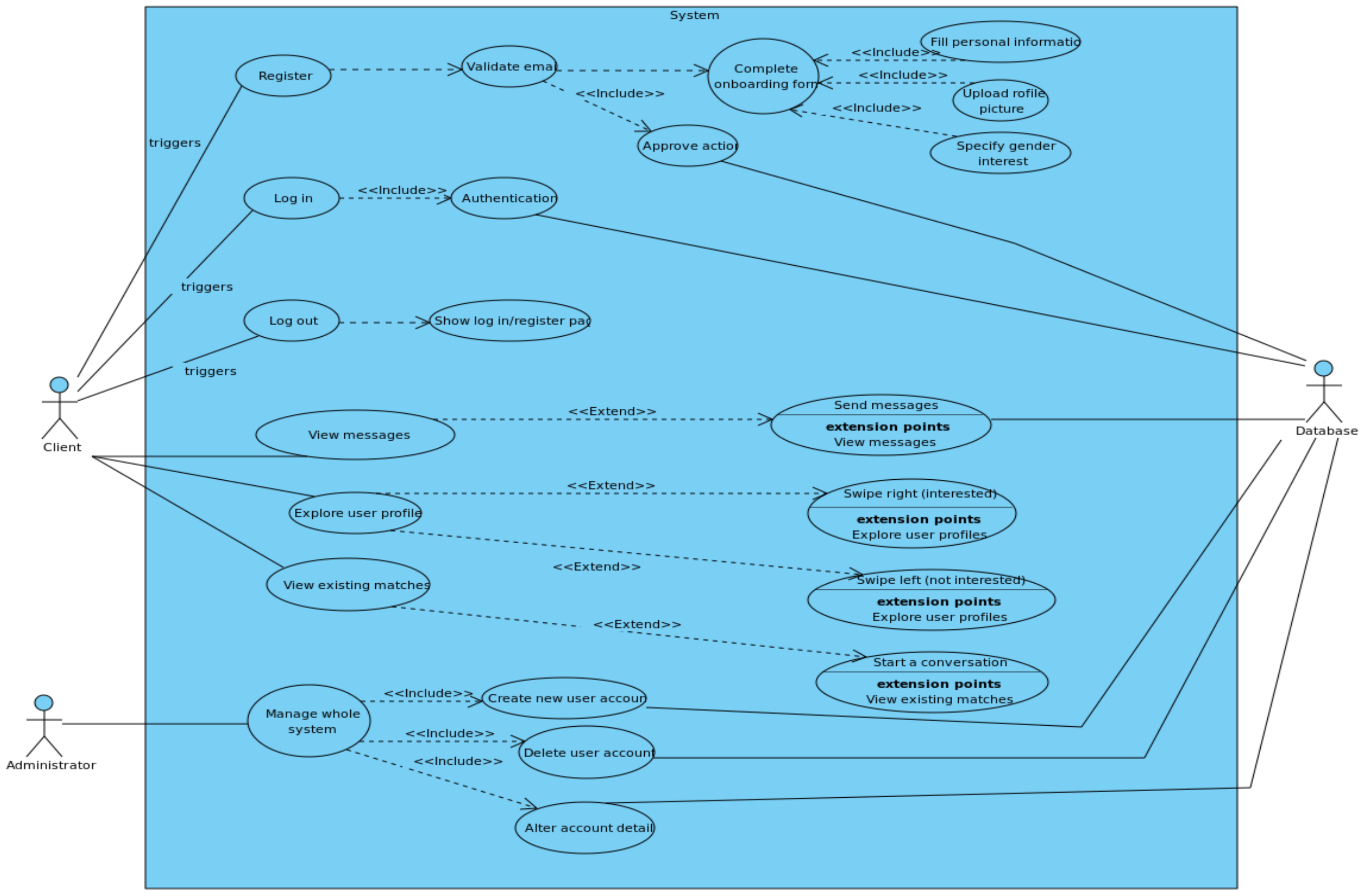
Requirement ID: 7	**Requirement Type: Functional**
Description: The system shall maintain a database of registered users' profile details.	
Rationale: To display user profiles for users to explore	
Source: Author	
Fit Criteria: All users will have a unique user ID.	
Customer Satisfaction: 5	**Customer Dissatisfaction: 5**
Priority: Essential	**Conflicts:** None
Supporting Material: None	Volere Source: Atlantic Systems Guild
History: New requirement	

Requirement ID: 8	**Requirement Type: Non-functional**
Description: The system shall meet user-friendly appearance requirements.	
Rationale: The system shall have a user-friendly interface.	
Source: Author	
Fit Criteria: The system shall display all the information clearly.	
Customer Satisfaction: 4	**Customer Dissatisfaction: 3**
Priority: Highly important	**Conflicts:** None
Supporting Material: None	Volere Source: Atlantic Systems Guild
History: New requirement	

Requirement ID: 9	**Requirement Type: Non-functional**	
Description: The system must adhere to good software security practices in order to protect user data.		
Rationale: The website stores personal information.		
Source: Author		
Fit Criteria: Security best practices will be followed. Every security conflict will be investigated to prevent future incidents.		
Customer Satisfaction: 4	**Customer Dissatisfaction: 5**	
Priority: Essential	**Conflicts:** None	
Supporting Material: None		Volere
History: New requirement		Source: Atlantic Systems Guild

Requirement ID: 10	**Requirement Type: Non-functional**	
Description: The application shall run on every browser that supports JavaScript.		
Rationale: The application was built with JavaScript frameworks.		
Source: Author		
Fit Criteria: Users can access the website without any errors.		
Customer Satisfaction: 5	**Customer Dissatisfaction: 5**	
Priority: Essential	**Conflicts:** None	
Supporting Material: None		Volere
History: New requirement		Source: Atlantic Systems Guild

7.3.2 Use case diagram



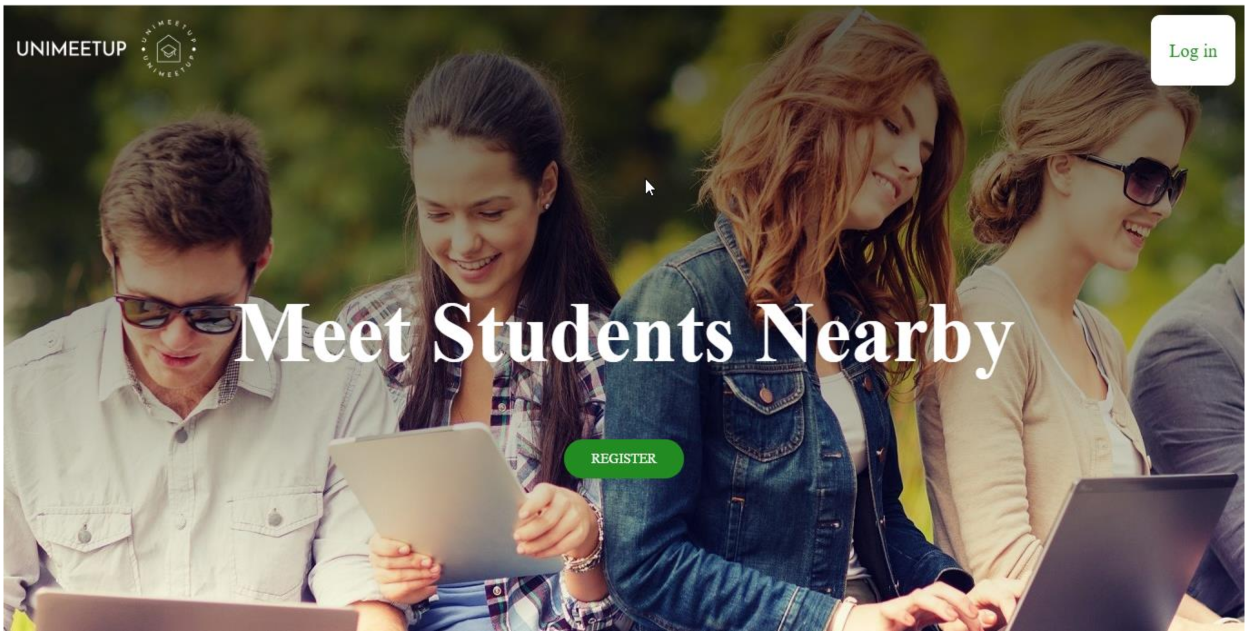
7.4 Appendix D Design

7.4.1 Logo

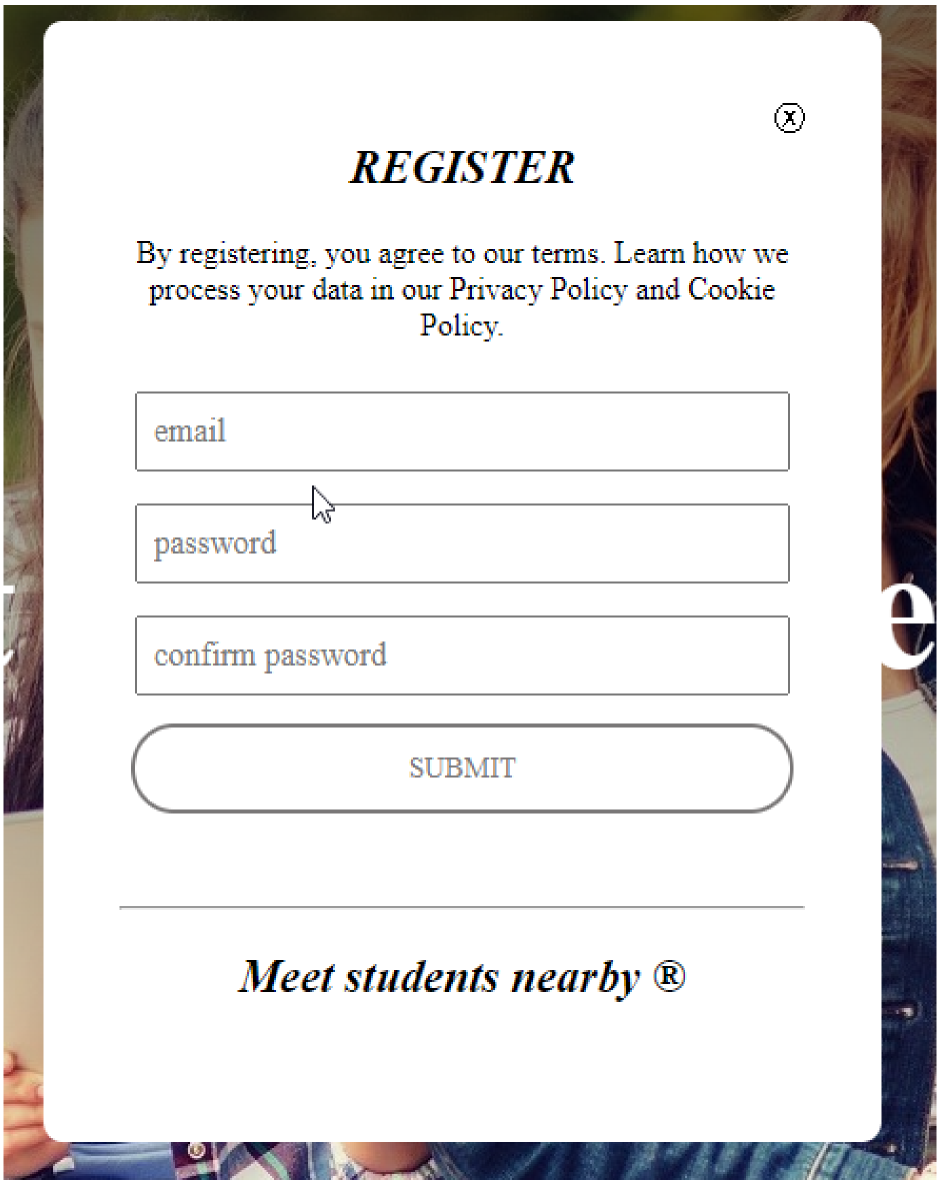
UNIMEETUP

Logo

7.4.2 Design documentation: webpages



Landing page

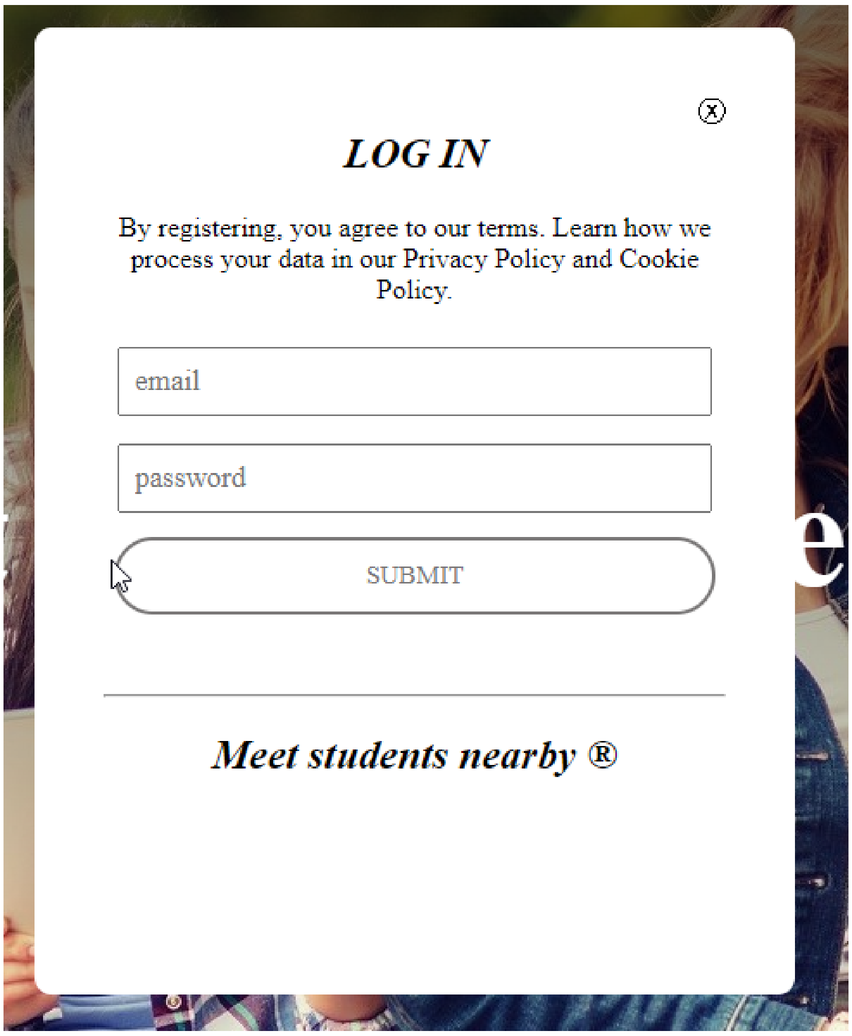


REGISTER

By registering, you agree to our terms. Learn how we process your data in our Privacy Policy and Cookie Policy.

Meet students nearby ®

Sign up form



LOG IN

By registering, you agree to our terms. Learn how we process your data in our Privacy Policy and Cookie Policy.

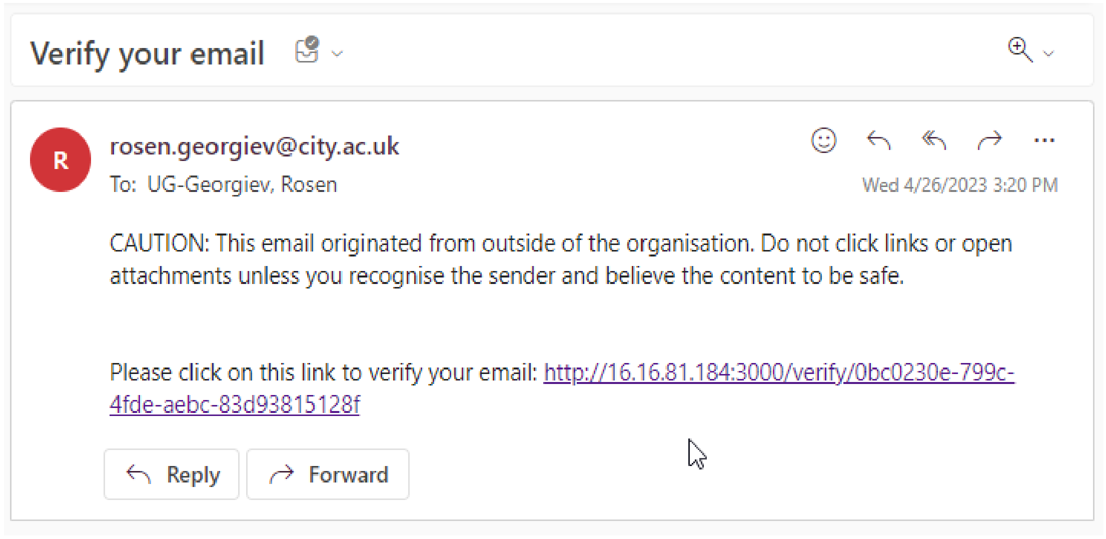
email

password

SUBMIT

Meet students nearby ®

Sign in form



Verify your email

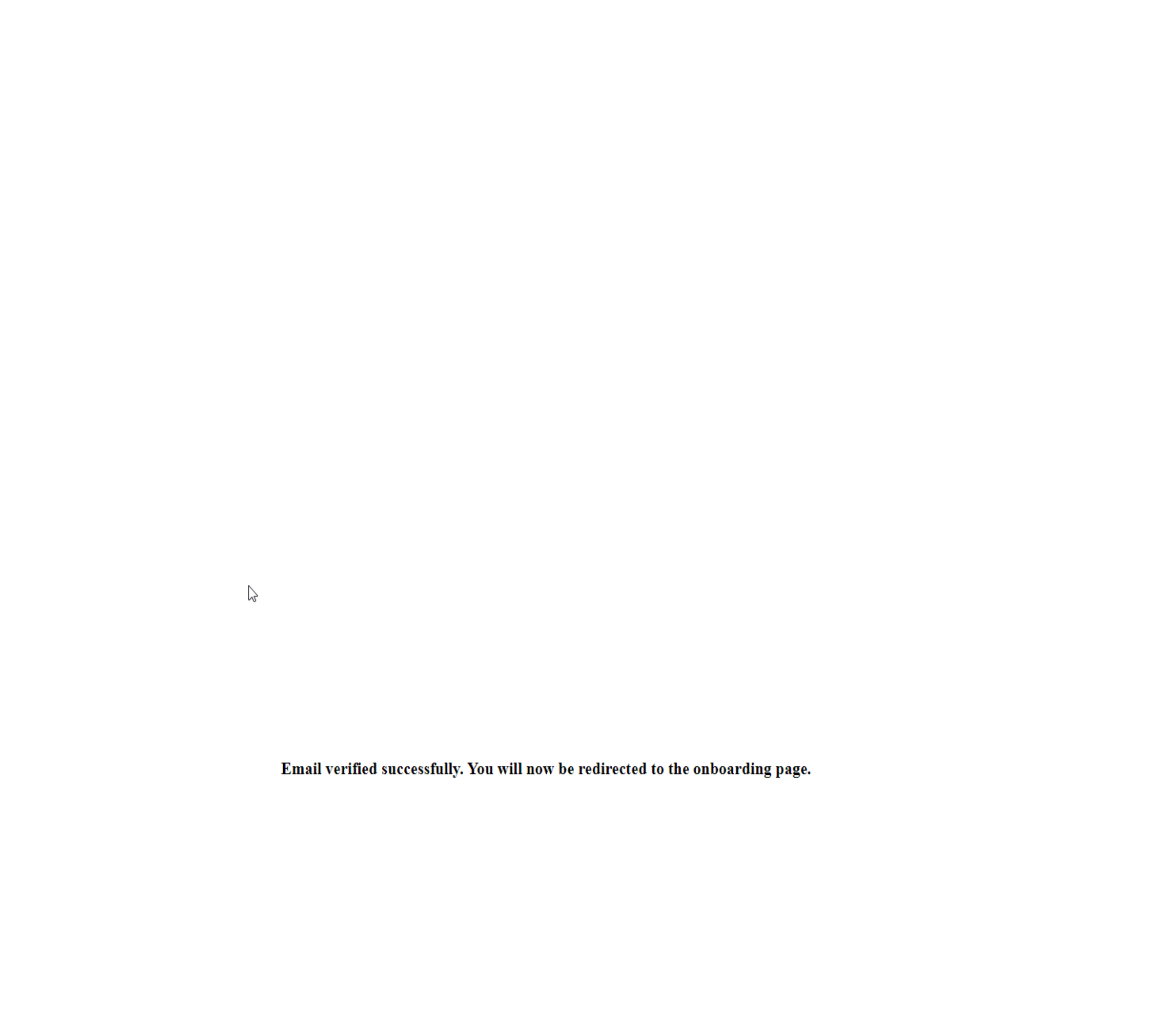
rosen.georgiev@city.ac.uk
To: UG-Georgiev, Rosen
Wed 4/26/2023 3:20 PM

CAUTION: This email originated from outside of the organisation. Do not click links or open attachments unless you recognise the sender and believe the content to be safe.

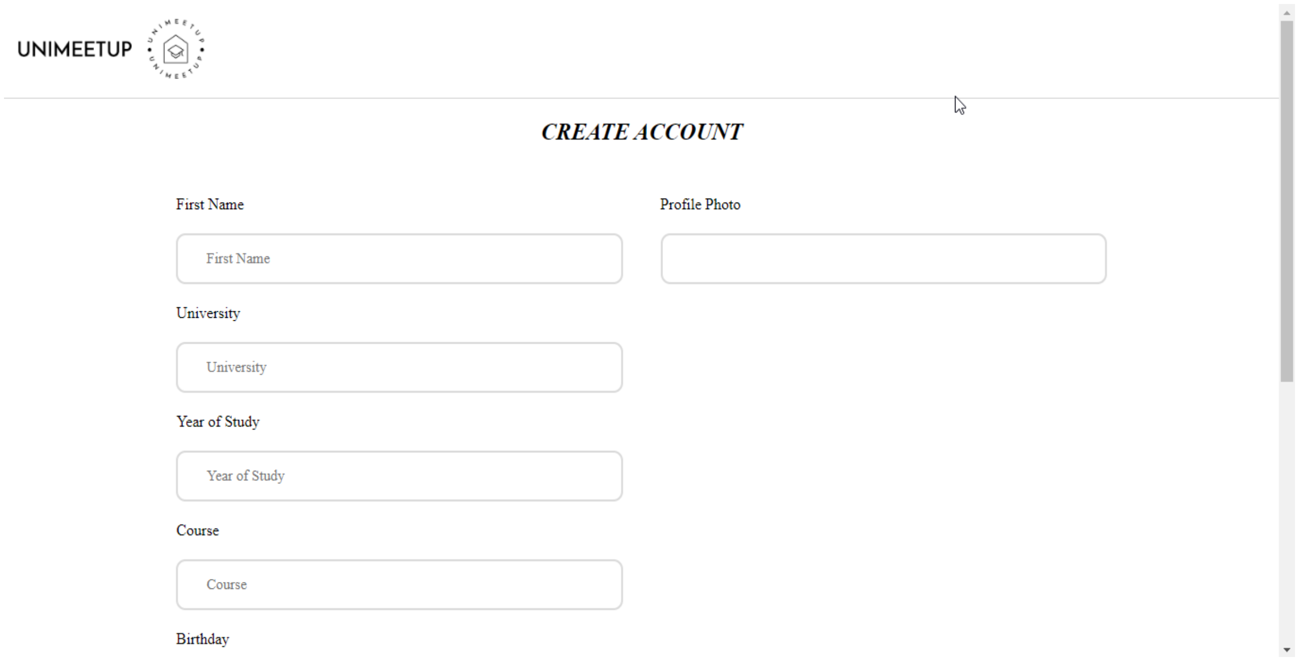
Please click on this link to verify your email: http://16.16.81.184:3000/verify/0bc0230e-799c-4fde-aebc-83d93815128f

Reply Forward

Email verification message


Email verified successfully. You will now be redirected to the onboarding page.

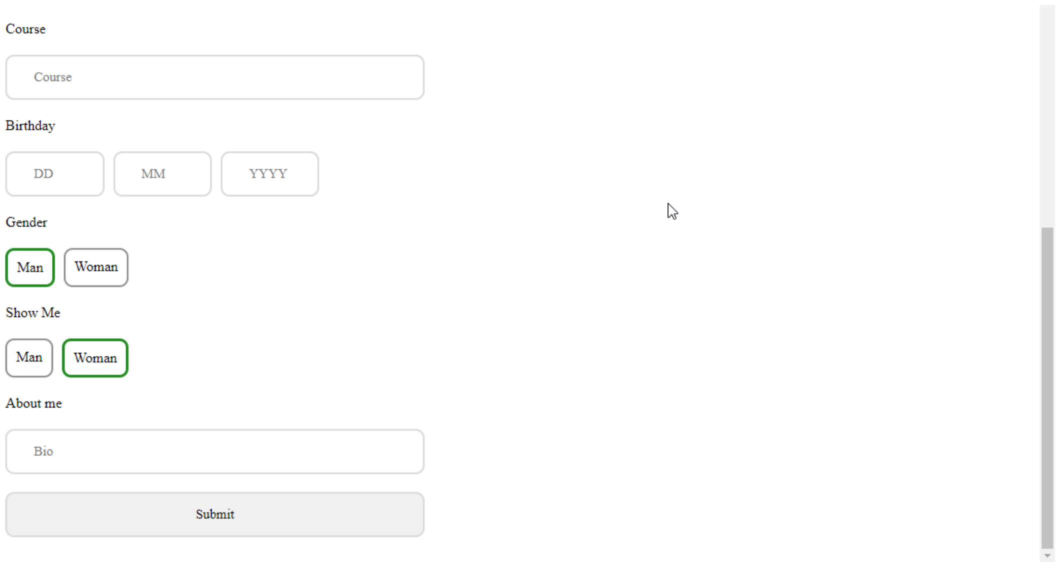
Verify email



CREATE ACCOUNT

First Name	Profile Photo
First Name	
University	
University	
Year of Study	
Year of Study	
Course	
Course	
Birthday	

Onboarding form



Course

Course

Birthday

DD MM YYYY

Gender

[✓ Man] Woman

Show Me

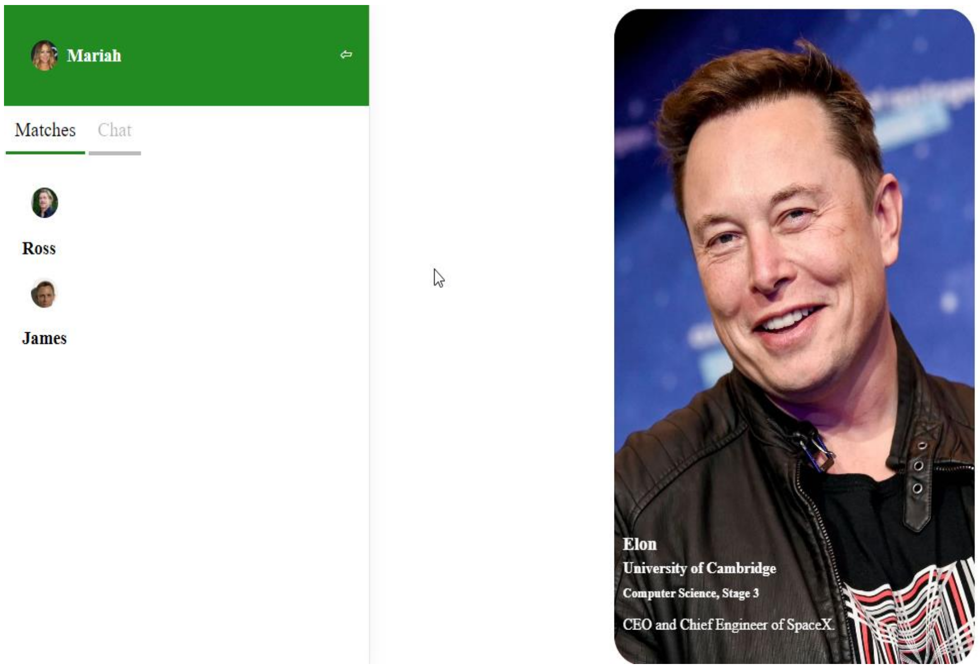
Man [✓ Woman]

About me

Bio

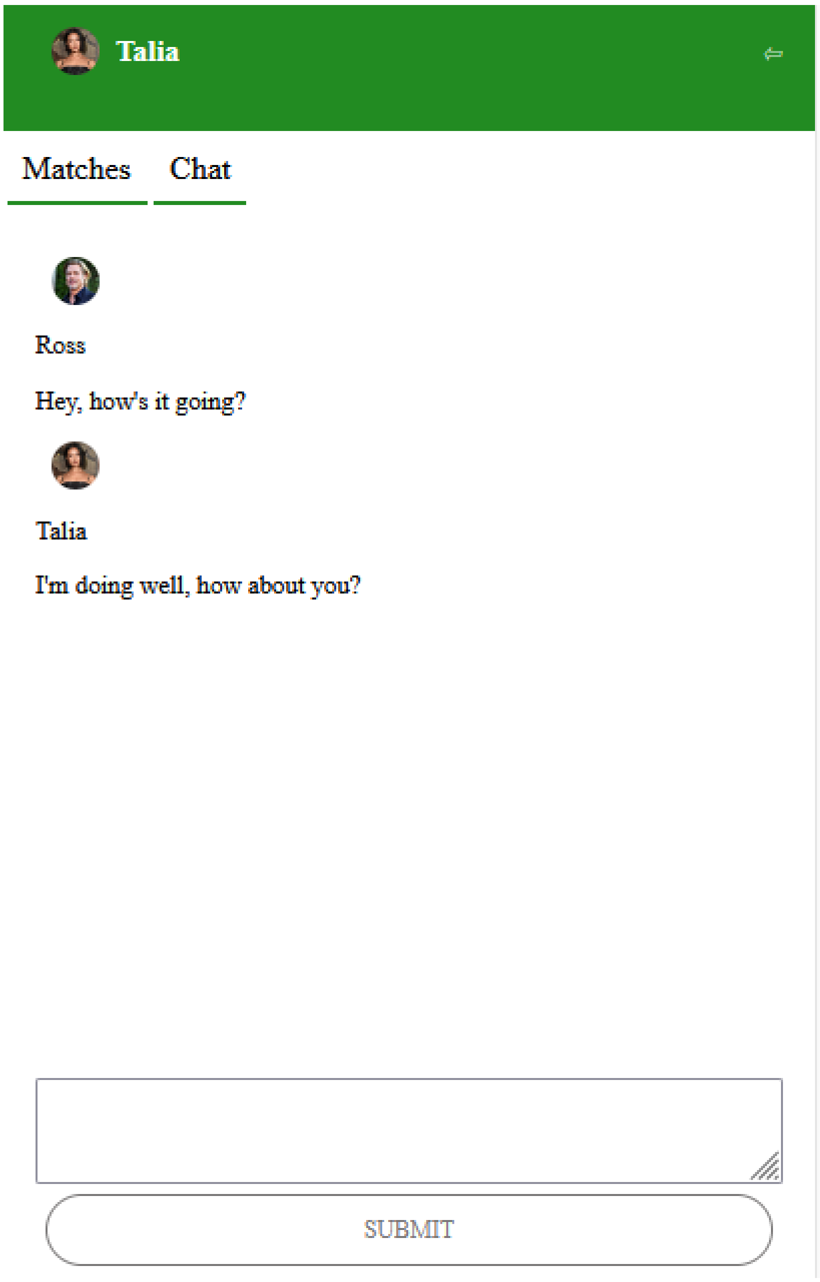
Submit

Onboarding form continued



Elon
University of Cambridge
Computer Science, Stage 3
CEO and Chief Engineer of SpaceX.

Dashboard


Talia

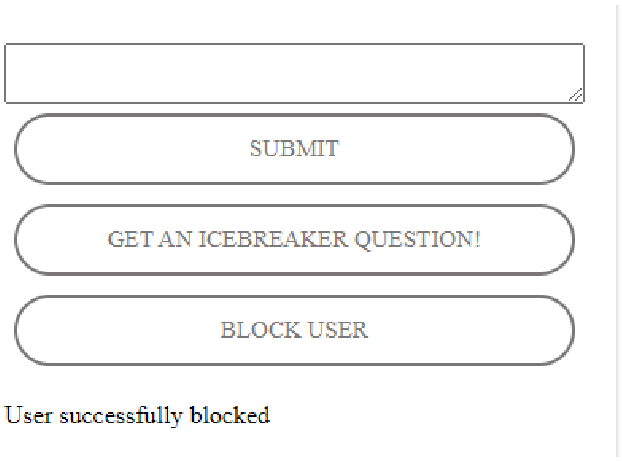
Ross

Hey, how's it going?

Talia

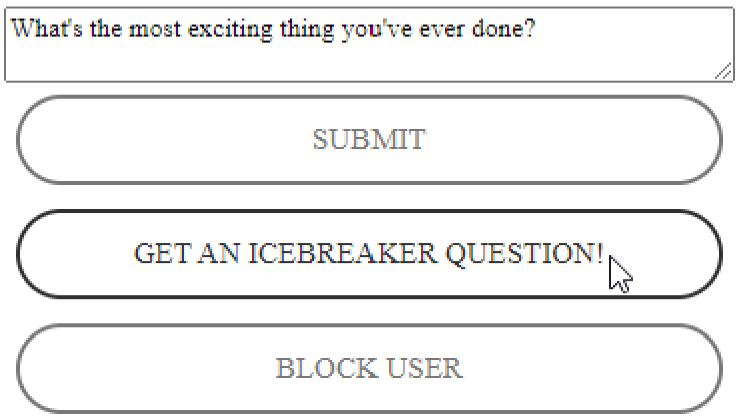
I'm doing well, how about you?

Chat

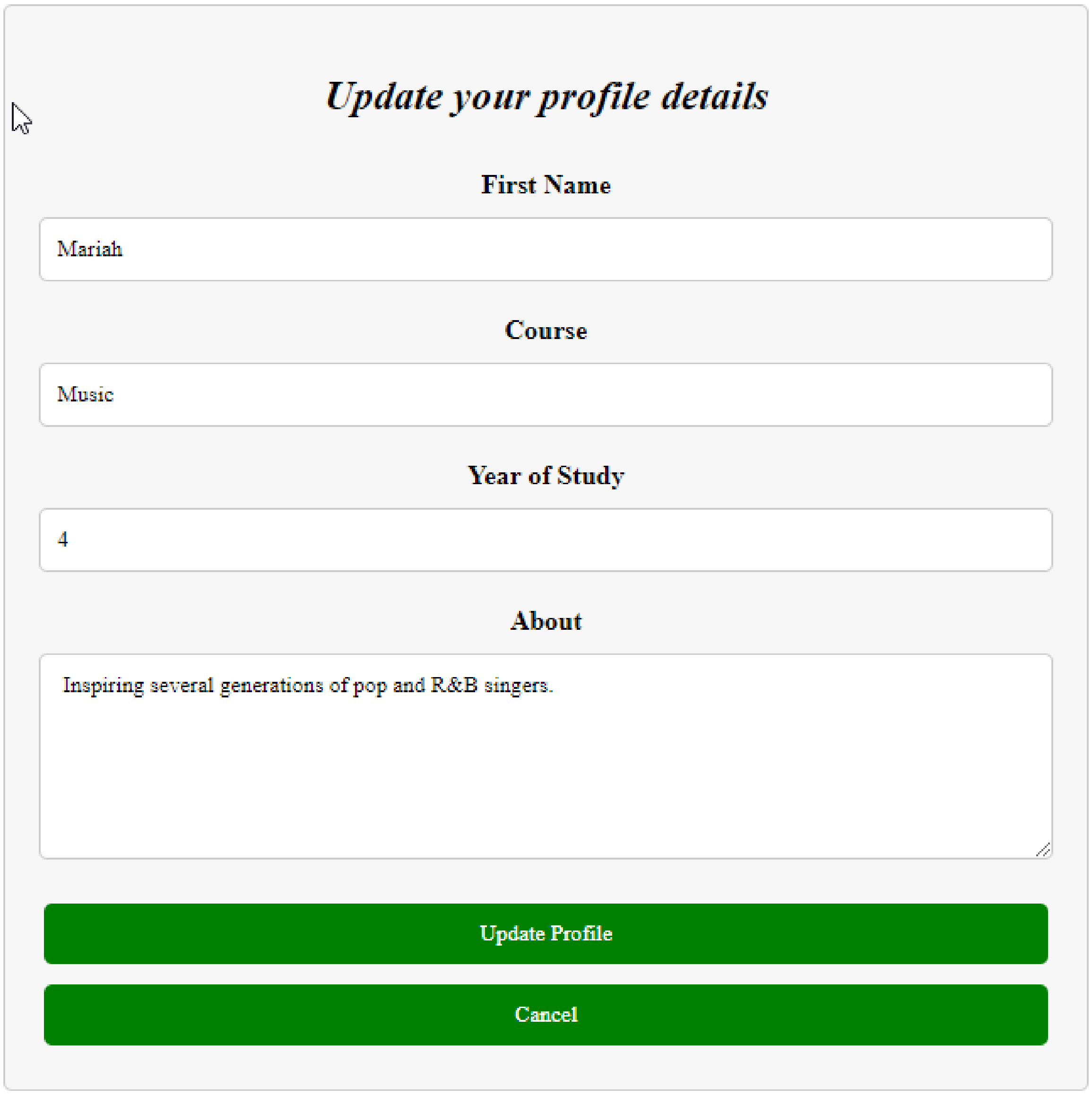


User successfully blocked

Block a user


What's the most exciting thing you've ever done?

Icebreaker questions


Update your profile details

First Name

Mariah

Course

Music

Year of Study

4

About

Inspiring several generations of pop and R&B singers.

Edit profile

7.5 Appendix E Test Report

Test case ID	**1**
Test case summary	Register a new user
Prerequisites	A PC or laptop device with an internet connection and a browser
Test procedure	1. Open the web application 2. Click on the **Register** button on the Home page 3. Fill out a form with required information
Test data	Username and password for a fictional user
Expected result	1. The user should be taken to the onboarding page upon successful registration 2. An error should be flagged if the information provided in the form is not as expected
Actual result	1. The application sent a verification email successfully 2. Verification was successful upon clicking on the email link 3. The user was taken to the onboarding page 4. Error messages were displayed when trying to register with an invalid **email address** 5. An error was displayed if the fields **password** and **confirm password** do not match
Status	Pass
Tested by	Author
Date	5th April 2023

Test case ID	**2**
Test case summary	Complete a user profile by filling in an onboarding form
Prerequisites	Test case 1 has been completed
Test procedure	Complete a form by providing information for a fictional user
Test data	Fictional data about a test user such as name, **university, course, year of study, and a profile photo**
Expected result	1. The application updates the user profile data successfully with the new information provided 2. The application displays a profile picture next to the user's name
Actual result	1. The user's profile was updated successfully 2. The uploaded profile picture appeared next to the user's name
Status	Pass
Tested by	Author
Date	5th April 2023

Test case ID	**3**
Test case summary	Log in
Prerequisites	Test cases 1 and 2 have been completed
Test procedure	1. Open the web application 2. Click on the **Sign in** button on the Home page 3. Provide username and password in the form displayed
Test data	Username and password for an already registered user
Expected result	The user should be taken to the dashboard. Log in should not be successful if a user does not exist in the database or the email and password do not match.
Actual result	1. The user was taken to the dashboard successfully. 2. An error message was displayed if the provided email does not exist in the database. Furthermore, log in was unsuccessful if the email and password do not match.
Status	Pass
Tested by	Author
Date	5th April 2023

Test case ID	**4**
Test case summary	Match with other another user based on a mutual like system
Prerequisites	Test cases 1, 2 and 3 have been completed. Tester has swiped right on the user from a different profile.
Test procedure	1. Navigate to the dashboard 2. Swipe right on a user from the explore section
Test data	User credentials
Expected result	A new user should appear in the user's matches list
Actual result	The user appears in the matches list as expected
Status	Pass
Tested by	Author
Date	5th April 2023

Test case ID	**5**
Test case summary	Communicate with other users via message feature
Prerequisites	Test case 4 has been completed
Test procedure	1. Select a user from the matches list 2. Type a message 3. Click the send button
Test data	A test message
Expected result	The application should send the message
Actual result	1. Tester sent a message 2. The message was received on the fictional user's account. 3. Tester replied successfully from the fictional user's account 4. Both messages were displayed in the chat history
Status	Pass
Tested by	Author
Date	5th April 2023

7.6 Appendix F Source Code

The product code was submitted to Moodle. It is also available in OneDrive: Unimeetup.zip

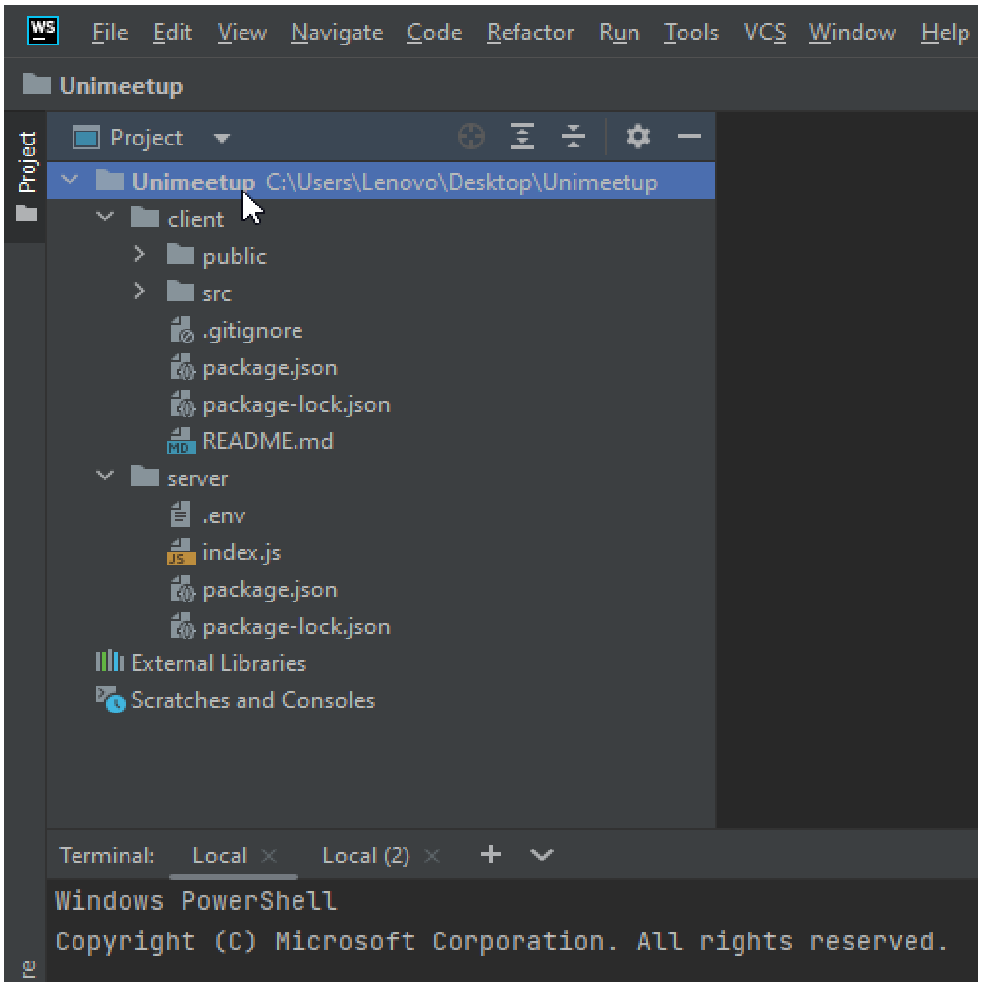
7.7 Appendix G Deployment Guide

UniMeetUp Deployment guide

URL for an online hosted version: http://16.16.81.184:3000/

Instructions for hosting the web application locally:

1. Unzip Unimeetup.zip. The web application consists of two main directories:
    - client: the React.js front-end of the project
    - server: the Node.js back-end of the project
2. In your IDE, click File -> Open and open the unzipped Unimeetup folder that contains the client and server directories.
3. This is what the file structure looks like in the IDE:



```
Unimeetup C:\Users\Lenovo\Desktop\Unimeetup
  client
    public
    src
    .gitignore
    package.json
    package-lock.json
    README.md
  server
    .env
    index.js
    package.json
    package-lock.json
External Libraries
Scratches and Consoles
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
```

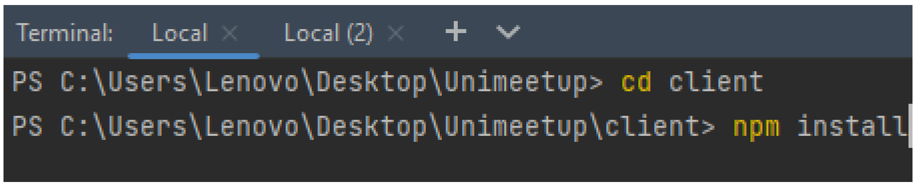
4. Make sure you have Node.js and npm installed. Npm (node package manager) is required to install project dependencies.
   https://nodejs.org/en

You can check if you have them by executing:

node -v

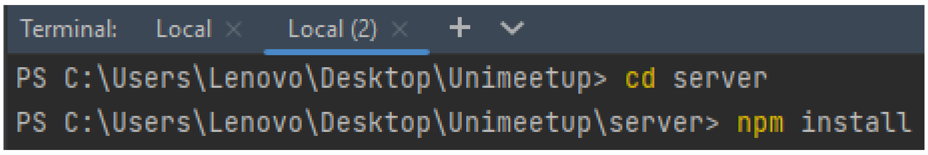
npm -v

5. Open two separate terminal windows.
6. In the first terminal window, navigate to the **client** directory using **cd client** and execute **npm install**. This will install the dependencies required to run the client.



```
PS C:\Users\Lenovo\Desktop\Unimeetup> cd client
PS C:\Users\Lenovo\Desktop\Unimeetup\client> npm install
```

7. In the second terminal window, navigate to the server directory using cd server and



```
PS C:\Users\Lenovo\Desktop\Unimeetup> cd server
PS C:\Users\Lenovo\Desktop\Unimeetup\server> npm install
```

execute **npm install.** This will install the dependencies required to run the server.

8. After installing the dependencies, execute **npm run start:frontend** in the client directory (first terminal window). This will start the frontend.
9. Then, execute **npm run start:backend** in the server directory (second terminal window). This will start the backend.
10. Once you have successfully started the client and server, you will be able to access the website at: http://localhost:3000/

The author suggests using WebStorm by JetBrains as an IDE and Google Chrome as web browser. The author has also tested the deployment using IntelliJ Idea by Jet Brains and Microsoft Visual Studio Code. If you have any questions, please contact the author.

7.8 Appendix H Video

The video shows the running software, demonstrating key features. Includes a walkthrough of a code section. It was submitted to Moodle and is also available at:

Project Demonstration Video - MediaSpace - City, University of London