CS3219 Group 45 Project Report

1. Background and Purpose

1.1 Background

In the current job seeking process, most companies are conducting not only the traditional behavioural interviews, but also technical interviews for their prospective interns or full-timers. These technical interviews often include a coding question (similar to that of one on LeetCode or Hackerrank), and the element of communicating the thought process of the question to the interviewer. While students can practise their problem solving skills individually on LeetCode or Hackerrank, they often need a partner to practise the communication aspect of a technical interview.

1.2 Purpose

PeerPrep was created to allow students and job seekers to be matched with like minded individuals and help each other practice taking technical interviews. Upon being matched, one user will be the interviewee attempting the question, while the other user will play the role of an interviewer. Ideally, PeerPrep would aid their users in their practice of taking technical interviews, and help them land their dream job,

2. Individual Contributions

Name	Student ID	Contributions
Aizat Bin Azhar	A0206022B	• Matching service • Communication service • Question service
Banerjee Aditya	A0200602E	• Developer Dependencies • Frontend for all Services
Cao Jiahao, Jonathan	A0199324W	• User Service • Dockerizing • CI/CD
Zhou Yi Kelvin	A0199333W	• Front End of Matching Service • Developer Documentation

3. Application Requirements

3.1 Functional Requirements of Services

S/N	Component	Functional Requirement	Priority
FR1.1	User Service	The system should allow users to create an account with username and password.	High
FR1.2	User Service	The system should ensure that every account created has a unique username.	High
FR1.3	User Service	The system should allow users to log into their accounts by entering their username and password.	High
FR1.4	User Service	The system should allow users to log out of their account.	High
FR1.5	User Service	The system should allow users to delete their account.	Medium
FR1.6	User Service	The system should allow users to change their password.	Medium
FR2.1	Matching Service	The system should allow users to select the difficulty level of the questions they wish to attempt.	High
FR2.2	Matching Service	The system should be able to match two waiting users with identical difficulty levels and put them in the same room.	High
FR2.3	Matching Service	The system should provide up to 30s for a valid match to be found.	High
FR2.4	Matching Service	The system should inform the users that no match is available if a match cannot be found within the 30 seconds provided.	High
FR2.5	Matching Service	The system should provide a means for the user to leave a room once matched.	Medium
FR3.1	Collaboration Service	The system should allow matched users to send messages to each other in the collaboration room.	High
FR3.2	Collaboration Service	The system should provide a coding pad for matched users to code together in real-time.	High
FR4.1	Question Service	The system should randomly provide a coding challenge with the selected difficulty from a pool of challenges.	High

FR4.2	Question Service	The system should provide matched users within the same collaboration room with the exact same question.	High

3.2 Non-Functional Requirements of Services

S/N	Component	Non-Functional Requirement	Priority
NFR1.1	User Service	Users' passwords should be hashed and salted before storing in the DB.	High
NFR1.2	User Service	Session tokens are blacklisted upon logout.	Medium
NFR3.1	Communication Service	Sent chat messages and new code input should appear instantly for both matched users.	High
NFR5.1	All Services	The system should have a fast response time. (No longer than 3 seconds for any user action)	High
NFR5.2	All Services	The system components should be deployable to a local staging environment via containers	Low
NFR5.3	All Services	The system components should be deployable to a production environment via a CI/CD pipeline	Low

Justifications for NFR Priorities:

Taking into consideration the current state of the app, we have chosen to prioritise the user experience of the app with **NFR3.1** and **NFR5.1** since it is related to the core functionality of the app and is what users will be interacting with for the majority of the time using this app.

Non-functional requirements that pertain to security such as **NFR1.1** and **NFR1.2** are a medium priority in general but will become more important as more personalisation features are added to the app and user accounts hold more personalised information such as saved questions or previous question attempts which can be used to continue where they left off.

The reason why **NFR1.1** is considered a higher priority than **NFR1.2** is that storing passwords in plaintext makes users highly vulnerable to database breaches, allowing attackers to freely access their accounts.

Meanwhile, session tokens are designed to have short lifespans, only granting authentication/authorisation capabilities for a short period of time, unlike valid passwords which can allow an attacker to generate session tokens anytime they want. Hence, it is more important to secure the password than session tokens.

For non-functional requirements related to DevOps like **NFR5.2** and **NFR5.3**, they are considered of a lower priority at this stage of development since the development team is still small and the scale of the app is relatively small as well. Since the end goal of the development process is to deploy the app to production in a steady and scalable state, ideally through automation, these requirements will eventually take a higher priority. Additionally as the team grows in size to facilitate the increasing scale of the app, proper CI becomes more important as more people are contributing to a single codebase to ensure code quality. Furthermore, a staging environment will become important to test the system as if it is in a production environment.

4. Developer Documentation

4.1 Milestones

4.1.1 Milestone 1

- Create Matching Service/Communication Service
- Improve User Service

4.1.2 Milestone 2

- Create Question service
- Improve frontend UI
- Containerize application components using Docker

4.1.3 Milestone 3

- Improve question bank to support better formatting
- Add CI/CD

4.2 Technology Stack

- MongoDB
- Express JS
- React JS
- Node JS

4.3 Architecture/Component Design

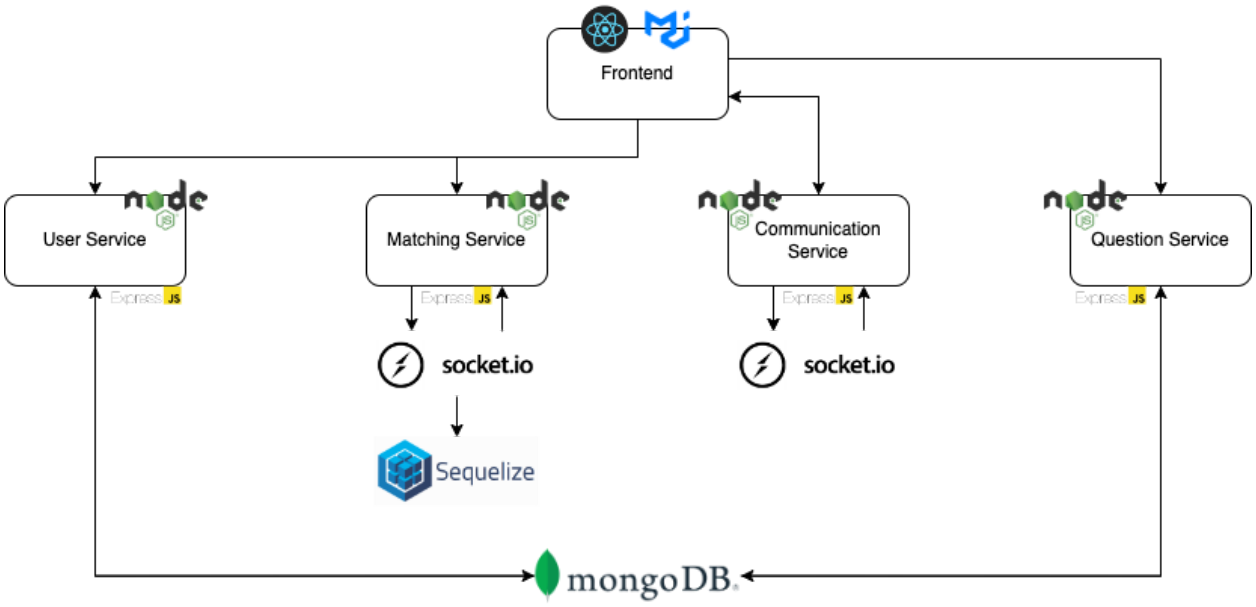

Figure 1: Architecture Diagram

Our application adopts Microservices architecture. We identified the different microservices that we would create by first identifying our bounded contexts, or divisions of our solution space. The bounded contexts were based on the requirements for our solution:

- User should be able to sign up for and log in to the platform
- Users should be able to match other users
- Once matched, users should be presented with a coding question
- Users should be collaborate and communicate in order to answer the question

Based on these requirements, we identified the following bounded contexts:

- Users
- Matching
- Questions
- Communication
- Collaboration

Subsequently, we derived our microservices based on these bounded contexts. The microservices are as follows:

- User-service
- Matching-service
- Question-service
- Communication-service (includes collaboration)

Aside from the microservices, we built a frontend that receives input from users and directs their requests to the appropriate service(s).

4.3.1 Frontend

The front end is developed using ReactJS framework, using Material UI theme and icons. It is the main bridge between the user actions and the various microservices below.

4.3.2 User Service

The user service is an ExpressJS server application that enables users to sign up with a unique username, which can be used to uniquely identify them when chatting with other users during code collaboration, and password, which is hashed and stored in the database.

It also allows users to authenticate themselves when logging in through the frontend, generating a session token, which is required for accessing authenticated routes on the frontend, such as the matching page and collaboration page, as well as authenticated requests to user service such as changing password or account deletion.

It handles requests from the frontend through the use of REST APIs. Examples are detailed in Section 4.4.

User information (username and password) are stored in a MongoDB Cloud Database. NoSQL database is chosen due to the simplicity of the data that is required to be stored.

4.3.3 Matching Service

Matching service serves to match 2 users who selected the same difficulty and is implemented with ExpressJS and the matching of 2 users functionality is implemented with the help of Socket.io.

We model a user that is in queue as a Sequelize object called QueueingUser which has 2 fields - the client's socket id and the difficulty they selected.

When a user attempts to find a match, they emit an event find-match and there are 4 possible events that follow:

1. There is another unmatched user in the FIFO queue for that difficulty - in this case they are paired and the unmatched user that was in the queue is removed from the queue
2. The queue for that difficulty is empty - in this case they are added to the corresponding queue
3. The user cancels the request for a match - in this case they are removed from the queue

4. After 30 seconds of not finding a match - in this case they are removed from the queue and an event is emitted to the client informing that no match could be found

A sequence diagram on this can be found in section 4.4.3.

4.3.4 Communication Service

Communication service functions to allow 2 users to communicate with each other in the form of the collaborative coding pad and instant messaging. Similar to the matching service, it is implemented with ExpressJS and Socket.io.

For the collaborative coding pad, when a user types into the coding pad, an event is emitted from the client with the text as payload. An event is then broadcasted from the server to the other client with the latest text and is rendered by the frontend.

A similar sequence of events happens when instant messaging, except that the event is emitted when a client sends the message.

4.3.5 Question Service

Question service provides a simple way for getting coding questions and it is implemented with ExpressJS. Our question bank is stored in MongoDB and every question is modelled with an id, title, difficulty and the question string itself.

The GET API takes in a difficulty (easy, medium or hard) and a parameter called “random” which is used to get one random question. The random parameter takes in anything and hashes it to an integer which is then used as an index to access the question array.

A sequence diagram on this can be found in section 4.4.3.

4.4 Behavioral Design (Sequence Diagrams of Microservices)

4.4.1 User Creation

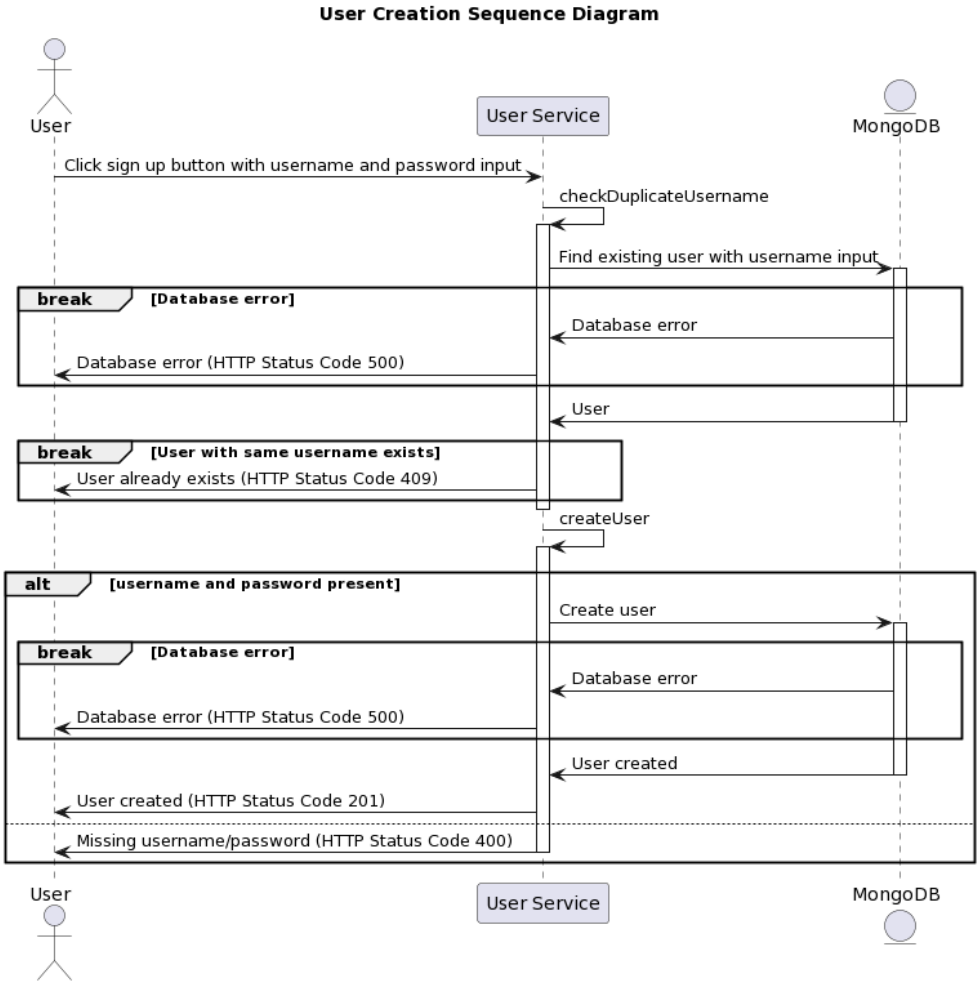

Figure 2: Sequence Diagram for User Creation

When a user clicks the sign up button with username and password fields filled out, a HTTP POST request is made to an API endpoint in user service. This endpoint will first invoke the **checkDuplicateUsername** middleware, which checks if there is an existing user in the MongoDB database with the same username as the username input, returning a 409 Conflict response if so.

It will then invoke the main controller function of the API endpoint called **createUser**. It first checks if the request contains both username and password. This check is mainly for testing during development since the frontend will perform this check on the username and password fields as soon as the user clicks the sign up button, if either is absent the request will not be made at all. That said, in the context of this endpoint, a 400 Bad Request response is returned if either username or password is absent. Otherwise, the password input is hashed and salted using the

bcryptjs package. Then a new user with username input and hashed password input is created in the MongoDB database, returning a 201 Created response.

4.4.2 User Login

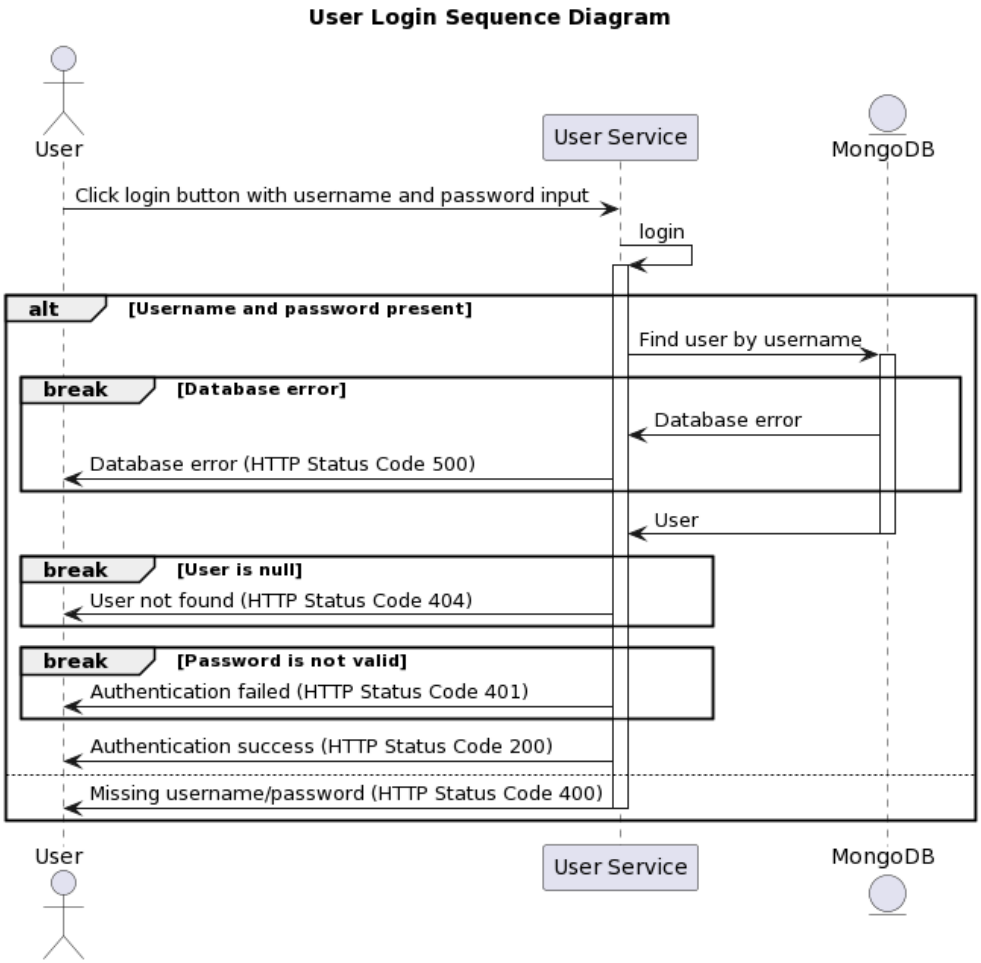

Figure 3: Sequence Diagram for User Login

When a user clicks the login button with username and password fields filled out, a HTTP POST request is made to an API endpoint in user service. This endpoint will invoke the main controller function **login**, which will return a 400 Bad Request response if either username or password is missing.

Otherwise, it will search the MongoDB database for a user with the username input. If there is no user with that username found, a 404 Not Found response is returned.

Otherwise, it checks if the password input hashes to the same value as the one stored in the database, if not the authentication fails and a 401 Unauthorised response is returned.

If authentication succeeds, a 200 OK response is returned, a JWT is generated using the **jsonwebtoken** package by signing the user id with a secret and returned to the frontend, which stores it in a cookie as a session token.

4.4.3 Matching/Question/Communication

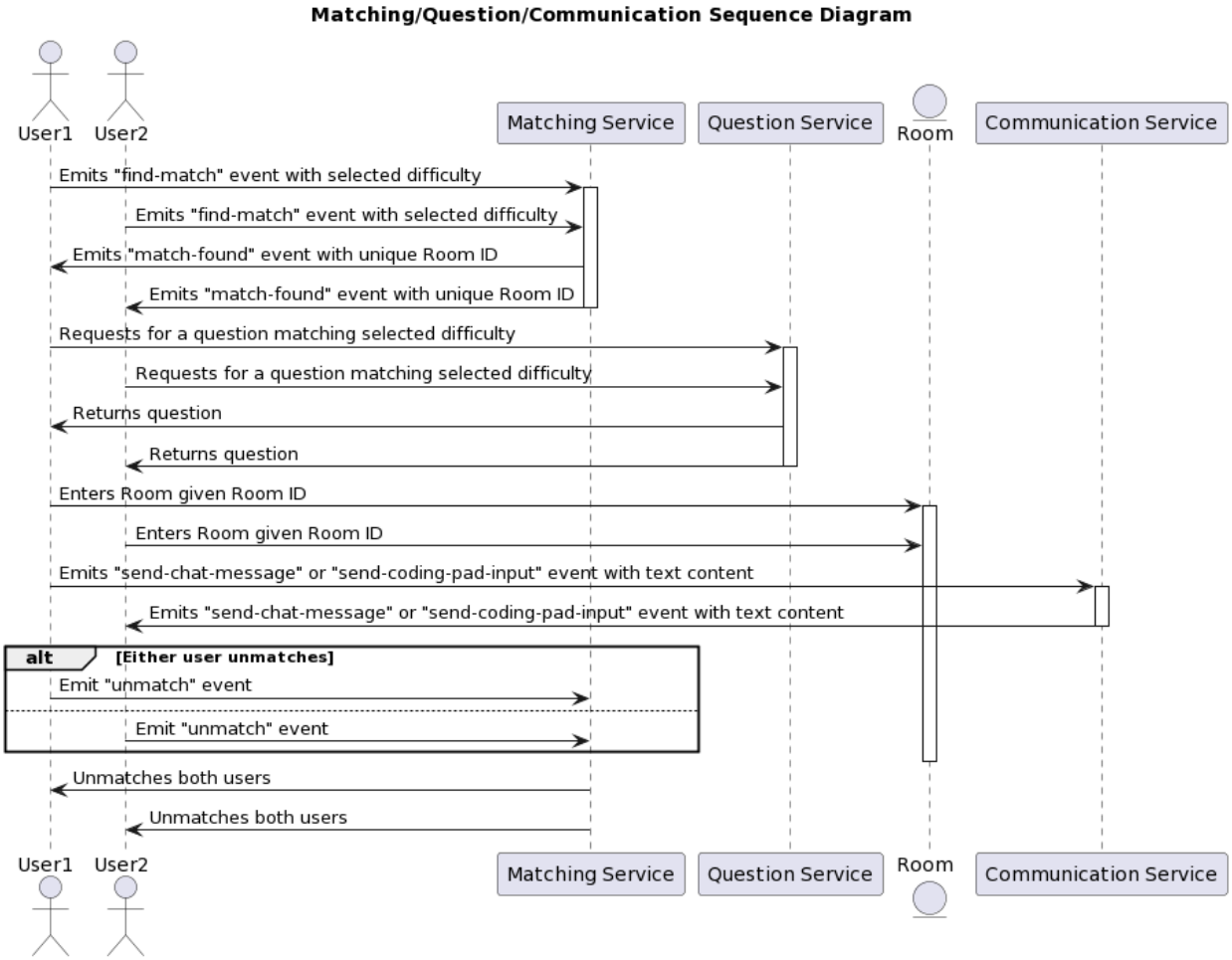

Figure 4: Sequence Diagram for matching users, fetching question and chat messaging

When two users select the same difficulty and click the Match button, they will both emit a "find-match" event, and when matched, the Matching Service will emit a "match-found" event back to them.

Both users will then request a question matching the selected difficulty to the Question Service, and the Question Service will respond with a question. Both users will then be given a Room ID, which will then lead to them entering the "Room", where they can send chat messages or type code into the coding pad. When that happens, a "send-chat-message" or "send-coding-pad-input" event will be sent to the Communication Service, and then the Communication Service will emit that same event to the other user.

When the interview is complete, and any user clicks the Back button, an unmatch event will be emitted to the Matching Service, and the Matching Service will unmatch both users promptly.

4.5 Design Patterns and Decisions

4.5.1 Facade

We have used the Facade pattern in matching-service and user-service whereby we hide the complexity by providing a simplified interface for the functionality that is required. This can be found in the `repository.ts` files.

4.5.2 State

Our frontend defines 2 contexts: Authentication/Authorisation and Matching. These contexts enable other components to behave in a manner specific to the internal state of the application. Some components of our frontend change their behaviour based on the state snapshots provided by the contexts.

An example would be the RouterContainer component. The component serves as an interface for performing state-specific routing behaviour. Based on its internal authentication state, it renders one of two subcomponents: AuthenticatedRoutes or UnauthenticatedRoutes. These serve as the interfaces for each authentication state, and perform routing accordingly.

4.5.3 Event Message

The microservices and users can emit and receive events, which will then determine their subsequent action. For example, as mentioned in Section 4.4.3, when a user selects a difficulty and clicks on the Match button, the user will emit a “find-match” event. When the Matching Service receives a “find-match” event, then the Matching Service would proceed to look for another user that is also finding a match.

4.6 Development Process and Deployment Details (CI/CD)

Development goals are split into the various Functional Requirements which are determined during every sprint planning session. After which, they are tracked using Github issues.

The various Sprints lined up with the Milestones set up by the Teaching Team, and are split as such:
Sprint 1: Week 4-6 (Milestone 1)
Sprint 2: Week 7-9 (Milestone 2)
Sprint 3: Week 10-12 (Milestone 3)

Continuous Integration (CI) workflows are implemented using Github Actions, which automates the running of tests and builds for all components whenever a pull request is made to the main branch. This ensures that all commits to the main branch meet a code quality standard.

Continuous Deployment (CD) workflows are automatically triggered upon a merge to the main branch. The triggers for building of each component are implemented on Google Cloud Build with a dedicated trigger for each component to build independently with its own environment variables. Google Cloud Build will build Docker images into the Container Registry using the Dockerfiles in each component's directory. Google Cloud Run will then run container instances of these Docker images. It also automatically scales the number of container instances based on the number of incoming requests.

5. Improvements and Enhancements

In our application, we have achieved all the necessary functionality to facilitate collaborative coding sessions, such as:

- Coding prompts/questions
- Difficulty-based matching service
- Real-time collaborative editing
- Instant messaging

As such, most of the improvements and enhancements that we would like to incorporate pertain mainly towards improving the coding experience of individuals on the platform.

From a coding perspective we would like to implement the following improvements:

- Add syntax highlighting to the coding pad
- Auto-fill and suggestions
- Add compilation and execution options
- Persistent collaborative sessions
- Support for test case suites

From a collaborative perspective, we would like to implement the following improvements:

- Options for voice and video call
- Group collaborative coding
- Contribution extensions
    - Overall contribution statistics
    - Code-to-author tagging
    - Git support/GitHub integration

From an individual users perspective, we would also like to implement the following features:

- Options to save questions/attempts at answering questions
- Ability to skip the given question if wish to attempt a different question instead

- User activity statistics
    - Questions answered
    - Difficulty of questions answered
    - Contribution to answers

From a technical perspective, as our application is containerised using Docker, we would also like to incorporate Kubernetes in order to manage and scale our deployed application according to functional needs.

From a security and efficiency perspective, the blacklisting of session tokens can be improved by storing blacklisted tokens in a Redis cache instead of in a MongoDB database. All authenticated endpoints such as changing password and account deletion will be required to check if the current session token is blacklisted. Since session tokens will be frequently used for these endpoints, accessing the database each time may not be very efficient. Using Redis cache will allow for much faster access to the blacklisted tokens. Additionally, the non-persistence of Redis will not be an issue as JWT tokens are short-lived.

6. Reflections and Learning Points

Throughout the duration of working on the PeerPrep project, we learned a lot about the process of deploying a full-stack project. Most of us had prior experience with frontend or backend but not both. However, in order to develop our respective components, we had to first understand the other components that they interacted with. For example, when developing the frontend, we were required to understand each service that the frontend interacted with. In fact, sometimes we had to alter some backend services to match the requirements for our frontend and vice-versa.

We have always wondered how the real-time synchronisation aspect of applications like Google Docs worked and to us it seemed like it would be very complex to implement. It was thus quite interesting for us to learn that we could do something similar with the help of socket.io when implementing the real-time, bi-directional communication aspects of the project (collaborative coding pad and instant messaging).

Overall, as a group we feel that this has been one the smoothest collaborative development experiences we have had. We believe that this was enabled through conformity to good version control practices. Examples of this include but are not limited to requiring approvals before merging PRs, testing PRs before approving them, enforcing code quality and coding standard, and coordinating work in a manner that prevents merge conflicts. Seeing the outcome of adhering to these practices allowed us to see why it is recommended that they are followed during the software development process, and why we should follow them in any future software development projects that we undertake.