CS3219 Project Final Report

Group 10

Name:	Matriculation Number:
Brendon Lau Cheng Yang	A0220999N
Elton Goh Jun Hao	A0217471H
Justin Yip Jeng Teng	A0222764E
Pang Yuan Ker	A0227560H

1. Introduction

Today, algorithmic and coding assessments have become commonplace in the interview process for software engineering roles. Interviews incorporating these algorithmic problems are called technical interviews, and hiring managers typically employ the use of these technical interviews to gauge a candidate's capacity for problem-solving and technical ability.

Similar to regular interviews, these technical interviews require a substantial amount of preparation prior to the interview in order for one to perform with an added catch - the interviewee must be both technically proficient, and be able to articulate their thought process during the interview in order to engage the interviewer. To adequately prepare for such interviews, one would have to practice in an environment that simulates the conditions as close to a real technical interview as possible, called a mock interview.

Although platforms allowing users to practice algorithmic problems exist, these platforms do not provide users with the option to simulate an interviewer that can provide oral prompts and feedback like in a real interview.

As such, our team aims to address this unmet need by building a web application as part of CS3219 Software Engineering Principles and Patterns. Our web application, PeerPrep, aims to provide candidates with a platform to hold mock interviews in an environment as close to a real technical interview as possible. This document will describe and illustrate the requirements, functions and development of the PeerPrep web application.

The project repository can also be referenced to provide more insight into implementation details.

1.1. Glossary

The following terminology and conventions will be used in subsequent parts:

Term	Definition
Mock Interview	A simulated technical interview with one interviewer and one interviewee
Web Application	A web-based application that can be run on any Chromium-based browser. Term may mean PeerPrep specifically, based on context
IDE	Integrated Development Environment
API	Application Programming Interface
Microservices Pattern	Refers to the microservices architecture. Since online sources use the microservices architecture and microservices pattern interchangeably, we will use the term microservices pattern for consistency in section 6.3
Monolithic Pattern	Refers to the monolithic architecture. Since online sources use the monolithic architecture and monolithic pattern interchangeably, we will use the term microservices pattern for consistency in section 6.3
MS	Microservice
JWT	JSON Web Token, a compact URL-safe token is used to transfer claims between two parties and it is commonly used for authentication
ORM	Object-relational mapping(s) (ORM). Provides a higher-level interface to read and write from the database
CSS	Cascading Style Sheets, the language used for styling HTML files
UI	User Interface
UUID	Universally Unique Identifier, used for identifying information that needs to be unique within a system

1.2. Intended Audience and Reading Suggestions

This report is intended for the teaching team of CS3219, to serve as a complement to the product demonstration and project repository for grading purposes. Developers and other interested parties are also welcome to peruse this document to understand our product design decisions and implementation methodology.

1.3. Project Scope

This web-based application aims to provide a user-friendly way for users to practice technical interviews in a realistic setting with others without prior arrangements. The application is modeled to emulate current-day technical interviews with two-way audio-video communication and a collaborative IDE. The application will allow users to match with others with the same selected question difficulty level.

This software is an easy entry point for users seeking to practice technical interviews. We aim to provide users with a fluid and intuitive experience, to ensure that users can maximize the value of our platform.

2. Overall Description

This section provides an overview of the PeerPrep web application, as well as the product perspective during development.

2.1. Application Objectives

The PeerPrep web application will have the following high-level requirements:

- Allow users to match with other users to conduct mock interviews
- Allow users to conduct mock interviews with questions of different difficulty levels
- Allow users to communicate with their matched user during mock interview sessions
- Allow users to review previous mock interview sessions

More specificity can be found under the requirements section.

2.2. Product Design

This section offers insight into the non-technical aspects of our product, the PeerPrep web application.

2.2.1. Product Overview

The objective of our product is to provide users with a platform to conduct mock interviews in preparation for actual technical interviews. To simulate a real technical interview, mock interviews in PeerPrep should provide an experience almost identical to industry-standard technical interviews, but with the added option of selecting difficulties enables progressive overload for a better learning experience.

This product is largely standalone, however, future extensions can potentially include external authentication services and using API from existing algorithmic problem providers for a larger question coverage.

2.2.2. Stakeholder Analysis

Our product will have 2 main stakeholders - users looking to practice mock interviews, and technical recruiters or interviewers. The former will be our main target audience.

Due to the rapid growth in the technology industry, both veterans and new entrants looking for technical roles will increase. Both experienced and inexperienced candidates will look for preparation materials and tools for technical interviews. As such, users of PeerPrep would potentially span across different demographics and age groups. To ensure that our product does not unintentionally deter users that are unfamiliar with technical interviews, we chose to go with a simple and intuitive approach when designing our user interface.

For new users, the PeerPrep application will serve as a minimalistic platform for users to start practicing mock interviews. To achieve this, users are not required to pre-arrange a partner. They can join a matchmaking queue to interview a random user. For more experienced users, there will be a history feature to allow users to view and analyze previous mock interview data. To ensure users are able to successfully conduct mock interviews, questions will also be provided by difficulty level so that users do not need to come up with their own questions.

To ensure that our service is always readily available for users, we have to prevent malicious attempts to abuse our backend API. As such, PeerPrep's services are only available upon user authentication.

However, PeerPrep does not only cater to interviewees. Technical recruiters are also key stakeholders in our product. We have established that interviewees require practice in order to perform during an interview, but that also holds true for interviewers. New technical recruiters can also use our product in order to hone their ability to identify key indicators and signals of a good candidate. To facilitate this, our team decided to opt for an audio-visual mode of communication so as to enable interviewee-interviewer pairs to have access to both verbal and non-verbal cues.

Aside from technical recruiters, stakeholders belonging to the same subset of interviewers can include career guidance staff from institutions of higher learning, where mock interviews can be offered as a tool in the career guidance staff's repertoire.

By providing a standardized platform close to a technical interview, we would be able to handle the needs of these stakeholders.

2.2.3. Usage Process

The main usage process will be as follows:

1. A user looking to practice for a technical interview learns about the application and creates an account.
2. After signing up, the user can log in to the application to access all services.
3. The user can navigate to the match-making page and select a difficulty level to begin searching for a mock interview partner.
4. After getting paired with another user, both users will enter a room together, where they will begin a video call with one another. Both users can access a collaborative IDE.
5. Users will assign roles on their own, one will be an interviewer while the other will be an interviewee.
6. The interviewer will pose a technical interview question to the interviewee in their preferred mode of communication, either verbal or using the collaborative IDE. Interviewers are recommended to use questions provided by the PeerPrep development team due to the fixed difficulty rating.
7. The interviewee can attempt the question using both verbal communication and the collaborative IDE. Syntax highlighting is supported for python.
8. After completing the question, the interviewee and interviewer can choose to swap roles and repeat steps 6 and 7.
9. When both users are satisfied with the mock interview session, they are able to leave the room.
10. After leaving the room, users can navigate to the history page to review their previous mock interview data and statistics.

3. Individual Contributions

The table below shows the contributions by each of the members

Name	Technical Contributions	Non-Technical Contributions
Brendon Lau Cheng Yang	● Frontend (Room and Matching) ● Communication Microservice ● History Microservice	Final Report Presentation
Elton Goh Jun Hao	● Frontend (room and matching related) ● Matching service ● Collaboration service ● Dockerize backend	Final report Presentation
Justin Yip Jeng Teng	● Frontend (Settings, Question, History, Navigation Bar) ● Question Service	Final Report Presentation
Pang Yuan Ker	● Frontend (Login, Settings, Question, History) ● User Service	Final Report Presentation

4. Requirements

4.1. User Stories

We thought of our user stories based on the point of view (persona) of a student in a technology-related field looking to practice questions with varying levels of difficulty for technical interviews with a partner in real-time, in order to get proficient in such interviews to achieve their dream jobs.

As a ___	I want to___	So that I can ___	Priority
user	be able to choose the difficulty levels and try a variety of questions	learn progressively	High
user	be able to match with a partner	simulate an actual technical interview	High

user	be able to collaborate (code) in real time	give feedback and observe	High
user	be able to select a topic	work on a specific type of questions	Low
user	be able to view the questions that I have attempted before	review what I have done	Medium
user	be able to view my interviewer feedback after the session	learn from feedback at anytime	Medium
user	be able to view the code that I have written during a session	review what I have done	Medium
user	be able to practice answering cs fundamentals through a question bank	revise on cs fundamentals	Low
user	select the type of language	work on problems with a language of my choice	Very Low
user	be able to chat with the interviewer (audio, message)	interact during the session	Medium

4.2. Functional Requirements

The list of functional requirements is grouped according to the microservices that cater to the specific requirements.

4.2.1. User Service			
ID	**Functional Requirements**	**Priority**	**Implemented**
U-1	The system should allow users to create an account with a username and password	High	Yes
U-2	The system should ensure that every account created has a unique username	High	Yes
U-3	The system should allow users to log into their accounts by entering their username and password	High	Yes

U-4	The system should allow users to log out of their accounts	High	Yes
U-5	The system should allow users to delete their accounts	High	Yes
U-6	The system should allow users to change their password	Medium	Yes

PeerPrep uses JWT for authorization of user sessions to ensure that users have to be logged in in order to use the web application's functionalities.

4.2.2. Matching Service			
ID	**Functional Requirements**	**Priority**	**Implemented**
M-1	The system should allow users to select the difficulty level of the questions they wish to attempt	High	Yes
M-2	The system should be able to match two users with the same difficulty levels and put them in the same room	High	Yes
M-3	If there is a valid match, the system should match the users within 10s	High	Yes
M-4	The system should inform the users that no match is available if a match cannot be found within 10 seconds	High	Yes
M-5	The system should provide a means for the user to leave the room once matched	Medium	Yes

PeerPrep allows users to select a difficulty preference and be matched with individuals of the same difficulty preference. This would allow for progressive learning and also cater to individuals of different abilities.

4.2.3. Question Service			
ID	**Functional Requirements**	**Priority**	**Implemented**
Q-1	The system should provide a randomized question based on the selected difficulty to a room	High	Yes
Q-2	The system should allow the interviewer to skip questions	Medium	Yes
Q-3	The system should allow users to swap roles as interviewee and interviewer	Medium	Yes (Done implicitly)
Q-4	The system should allow developers to maintain questions to the question bank easily	Medium	Yes
Q-5	The system should allow users to select the topic after selecting the difficulty	Low	No

PeerPrep would suggest a question based on the difficulty selected by the user. These questions will be randomly chosen from a select pool of questions which are likely to be asked during a technical interview. The application also allows users to skip questions in case the users have seen the question before or if the pair would like to attempt more questions. We did not implement the stretch goal of allowing users to be matched based on a selected topic and this feature can be explored in the future.

4.2.4. Collaboration Service			
ID	**Functional Requirements**	**Priority**	**Implemented**
CL-1	The system should allow users to view and edit a collaborative code editor in real-time	High	Yes
CL-2	The system will provide syntax highlighting for code in the collaborative code editor (One language)	Low	Yes

PeerPrep uses an IDE similar to that of Coderpad or Hackerrank which are commonly used in technical interviews. The IDE currently provides syntax highlighting for Python and the

highlighting can be extended to other languages in future by allowing users to select a programming language.

4.2.5. Communication Service			
ID	**Functional Requirements**	**Priority**	**Implemented**
CM-1	The system should provide an audio chat for users to interact in the room	High	Yes
CM-2	The system should provide videocam for users to see each other	Medium	Yes
CM-3	The system should provide a chat for users to interact in the room	Low	No

Although our team had originally planned to settle for a chat component for the pair to communicate with one another, we came to the conclusion that it would not be reflective of an actual technical interview where the interviewer and interviewee would be verbally collaborating on solving the given problem. Thus, we decided to implement an audio/video communication platform to more accurately replicate the actual technical interview environment. The functions of the chat is fully covered by both the audio/video component as well the collaborative IDE.

4.2.6. History Service (Stretch Goal)			
ID	**Functional Requirements**	**Priority**	**Implemented**
H-1	The system should track and highlight questions previously completed by the user	Low	Yes
H-2	The system should allow users to view code written in the sessions	Low	Yes
H-3	The system should allow users to sort past interview details by difficulty, time and name	Low	No
H-4	The system should allow users to view past chat records	Low	No

PeerPrep allows users to view their past mock interview session records in a clear and concise table format. Users can also view the code that was written on the collaborative IDE when accessing a specific interview session. Though we had this history feature listed as a stretch goal, we felt that this would allow them to revise certain questions and also provide a means for them to keep track of their progress, which would greatly improve the user's experience.

4.3. Non-functional Requirements

No	**Non-Functional Requirements**	**Attribute**	**Priority**
NFR1	Users' passwords should be hashed and salted before storing in the DB	Security	High
NFR2	User must be authenticated using JWT	Security	High
NFR3	The communication and collaborative environment must be real-time (within 300ms delay) and seamless	Usability and Performance	High
NFR4	The system should be robust to handle unplanned disconnects such as refreshing the page	Usability	High
NFR5	The application will need to work on Chromium based browsers such as Chrome, Edge, Firefox, Safari etc.	Usability	High
NFR6	The system should allow development on the following operating systems: Linux, Mac and Windows	Usability	High
NFR7	The system will have data persistence to keep track of user data	Usability and Performance	Medium

4.4. Quality Attributes

Legend	
Score	**1 (Lowest) to 7 (Highest)**
<	**Attribute on the left has a higher priority than attribute above**
^	**Attribute above has a higher priority than attribute on the left**

Attribute	Score	Performance	Security	Usability	Availability	Scalability	Reliability	Extensibility
Performance	**6**		**<**	**^**	**<**	**<**	**<**	**<**
Security	**5**			**^**	**<**	**<**	**<**	**<**
Usability	**7**				**<**	**<**	**<**	**<**
Availability	**2**					**<**	**^**	**^**
Scalability	**1**						**^**	**^**
Reliability	**4**							**<**
Extensibility	**3**							

4.4.1. Top 3 Priority Ranking

1. Usability
2. Performance
3. Security

We chose Usability as the highest priority as we felt that the user experience would be of utmost importance for an application that is built around simulating the technical interview process. Performance was a close second as we would need the system to be performant in the aspect of communication and the collaborative code editor which should have low latency. Security was also relatively high on the list as we would not want malicious users to be able to join rooms that they are not supposed to or add random data into our system (e.g. adding random data into the question database resulting in this data being fed to the user as a question).

5. System Architecture

5.1. Microservice Architecture

The microservice architecture [1] is highly suitable for the website application.

Firstly, the functionalities can be broken up into different modules and each module represents a single responsibility. Also, the modules can be represented by each service and do not share data with one another. For example, the question service and communication service can run independently as the question service only needs to be concerned about providing questions for the users and the communication service only needs to be concerned about the video chat.

Secondly, the software can be broken into loosely coupled modules. This increases scalability, testability and deployability as each module is independent of others, it can be managed independently and hence, reducing complexity.

Last but not least, it improves work efficiency and eliminates long-term commitment to a technology stack. Microservice Architecture allows developers to develop services independently which minimizes bottlenecks in development and allows the different suitable technology stacks to be used for different services.

5.2. Software Architecture Diagram

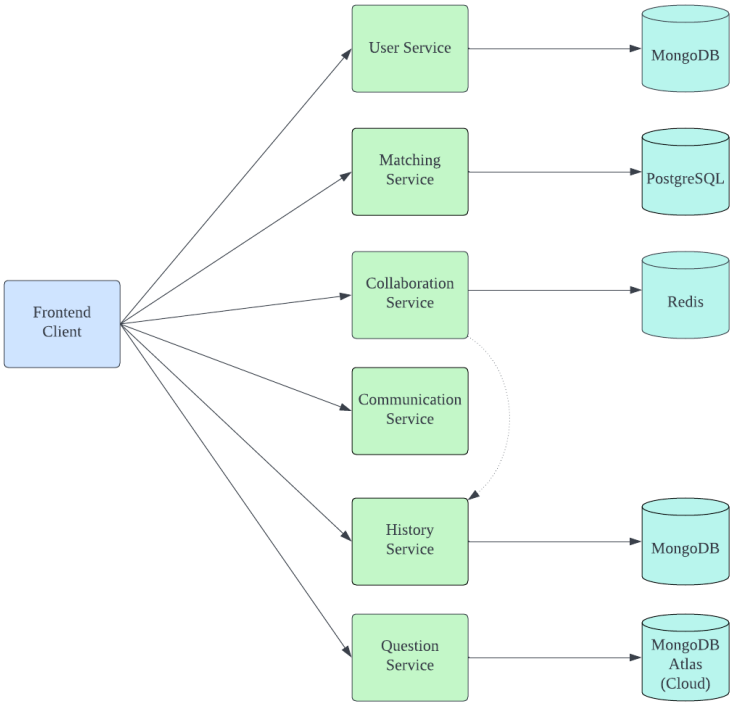

Figure 5.2. Microservice Architecture Diagram

The frontend client will communicate with the microservices to perform operations via well-defined APIs.
The microservices in the software are as followed:

- User Service
- Matching Service
- Collaboration Service
- Communication Service
- History Service
- Question Service

6. Design and Implementation

6.1. Software Features

To illustrate how our microservices will be used in a more concrete manner, the PeerPrep web application will provide the following features to users:

- Users are able to log in to the application
- Users are able to change their password
- Users are able to delete their account
- Users are able to select a difficulty rating to match with another user
- Users are able to use a collaborative IDE for mock interviews with matched user
- Users are able to communicate via audio and video during mock interviews
- Users are able to retrieve questions based on difficulty rating within the mock interview room
- Users are able to change questions within the mock interview room
- Users are able to view data on past mock interviews

6.2. Technology Stack

The main frameworks, libraries and databases used are listed below:

- ReactJS, ExpressJS, NodeJS
- MongoDB, PostgreSQL, Redis
- Mongoose, Prisma

6.3. Design Patterns and Considerations

6.3.1. Microservices Pattern

As mentioned in the system architecture section, we adopted a microservices pattern to modularize the different components of our web application.

The alternative to this pattern would be its counterpart, a monolithic pattern. The advantage of developing an application using the monolithic pattern is that deployment would be more straightforward and relatively easier as compared to a microservice pattern. Debugging and testing during development would be more streamlined, since the backend would operate as a single unit and it is easier to follow each API call to identify the source of error.

However, a monolithic pattern is difficult to scale and would involve much more complexity when attempting to develop iteratively in a team. In contrast, the modular nature of microservice architecture would make splitting of work more intuitive and allow for asynchronous development of features from different components of the application. Team members are able to have just a high-level understanding of the application and specialize in the development of specific sections of the application.

Furthermore, a microservice pattern would be much easier to scale as new features can be added as extensions to current services, or as a new service altogether. This would be in line with the open-close principle as the components would be open to extensibility and closed to modification.

By going with the microservices pattern, we will be able to explicitly ensure that we are able to execute the separation of concerns principle, as each microservice would operate independently from one another. Meanwhile, this would not be as explicit in a monolithic pattern due to the nature of a monolithic application being a single unit.

6.3.2. Pub-Sub Pattern

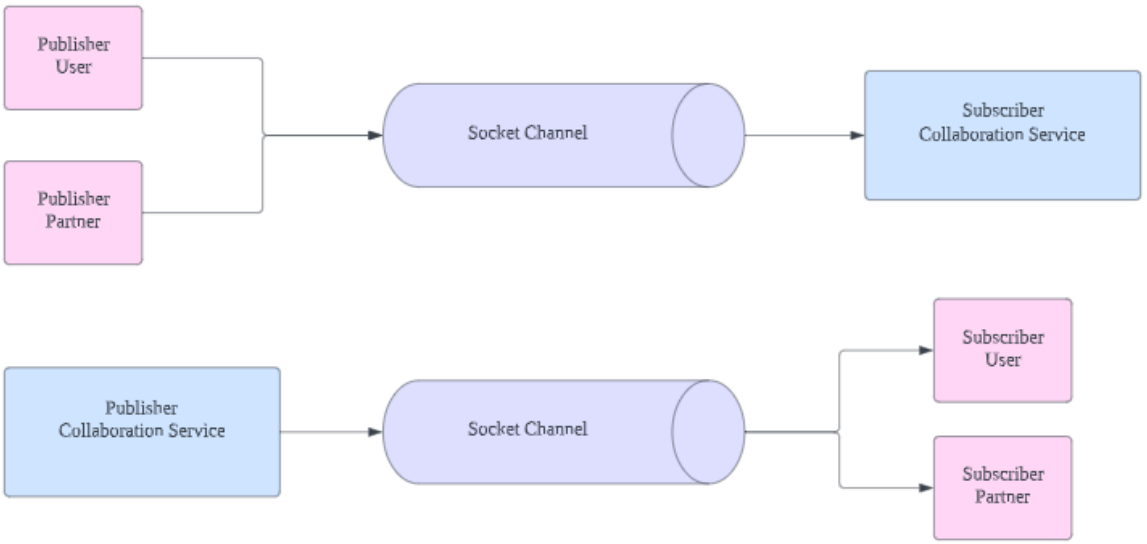

Figure 6.3.2. Collaboration Pub-sub diagram

For clients to communicate with one another in real-time, we use WebSockets. It is utilized in our communication service, collaboration microservice, and matching microservice. For instance, under the collaboration service, clients can subscribe to updates made by their interview partner and publish changes they have made. Pub-sub pattern [2] is used because it allows for low-latency communication. It also allows our services to be loosely coupled as the publisher and subscriber can operate independently without knowledge of one another.

An alternative message pattern that we had considered was the observer pattern, but we realized that the publishers do not actually need to have knowledge of the subject/subscriber. Thus, the pub-sub pattern is more appropriate for our situation.

6.3.3. Model-View-Controller Pattern and N-tiered Architecture

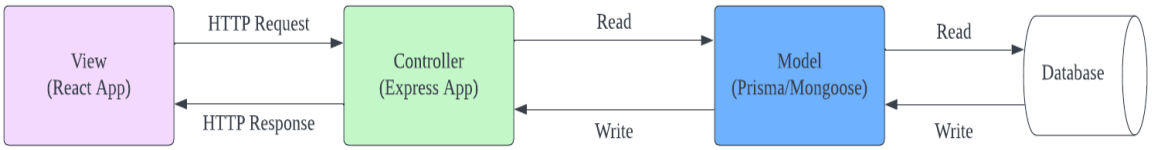

Figure 6.3.3. MVC diagram

Our services follow the MVC pattern [3]. MVC helps us to establish a separation of concerns as the components are independent of one another. Additionally, it makes it simple to add a new controller, view, or model without affecting existing components, making our program more extensible rather than modification intensive, effectively adhering to the Open Closed Principle. Our view would be our React application. Our controller would be the application

logic of our microservices (express application). For our model, we use ORMs such as Prisma and Mongoose, which offer a higher-level interface for accessing the database.

Originally, our team considered the Facade pattern as a close alternative [3]. Using the Facade pattern would allow our application to encapsulate our microservices to ensure that there is an added layer of security in our application. However, since a Facade would run contrary to the objective of decoupling our microservices, we decided to use the MVC pattern instead.

Additionally, within the model component, we employed the use of **3-tiered architecture** implicitly for each microservice. This further accentuates the separation of concerns principle.

6.3.4. Shared Data Pattern

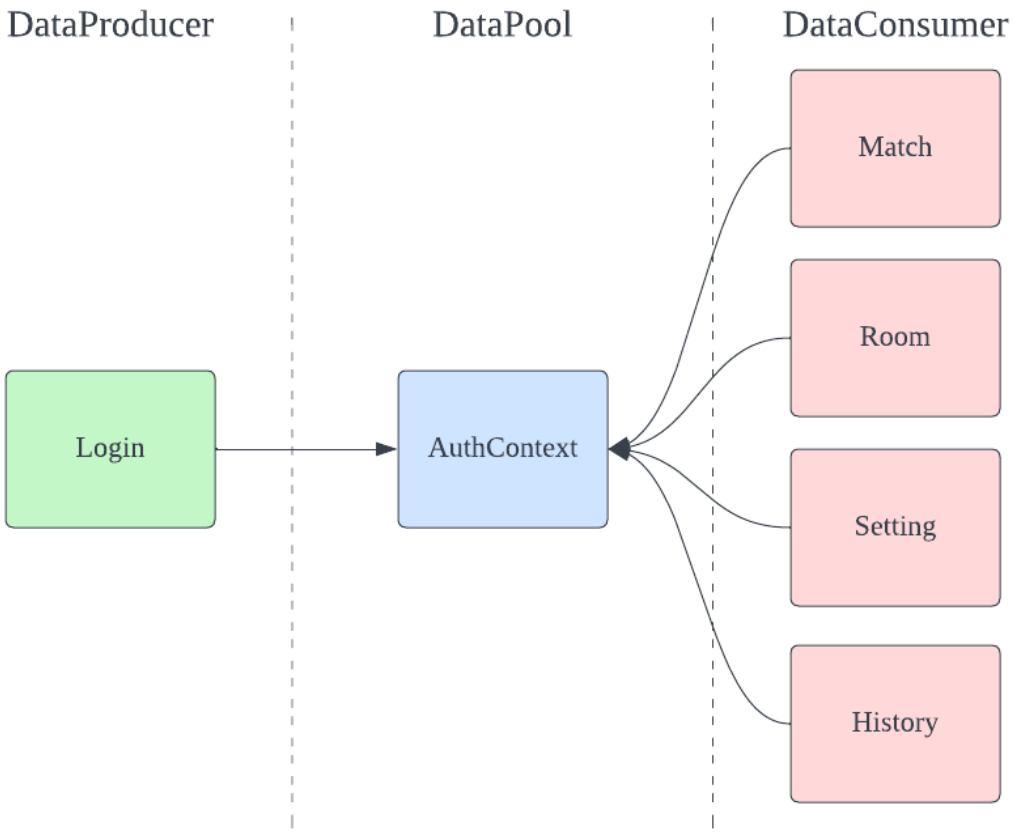

Figure 6.3.4 Diagram showing implementation of AuthContext

The Shared Data Pattern is a design pattern that complements the Microservice Architecture because it reduces coupling and improves cohesion by reducing data duplication and inconsistency in our databases [4]. In our web application, this pattern is used to decouple the producers from the consumers of data and comprises 3 types of classes. The Data Producer Class generates the data to be exchanged, the Data Pool Class is where the data

to be exchanged are stored and the Data Consumer Class receives the data to be exchanged.

In the client application, we used the useContext hook as the Data Pool for the Shared Design pattern. There are 2 Data Pool Classes which are AuthContext and SocketContext. Since both classes have a similar implementation, the Authcontext will be used for further elaboration. This pattern is suitable for this use case to decouple the Login Class from other classes that require the user's information generated by the Login Class and the data is needed synchronously. Such as accessing the cookie in AuthContext. Furthermore, the design pattern allows for extending more Data Consumer Classes without changing the implementations of other classes.

Another alternative we considered implementing was the Mediator pattern. The Mediator pattern allows other classes and components to interact with the Commander Class to update and retrieve relevant data. However, we do not need two-way interactions between the Colleague Classes and the Commander Class. Hence, using the Shared Data pattern is more suitable to reduce dependency as the data only flowed from the Data Producer Class to Data Pool Class and finally to the Consumer Classes. Also, since the main focus is on moving data and the Shared Data Pattern is able to prevent the potential bottleneck at the mediator interface as we increase the number of classes and components that shares data.

6.3.5. Data Transfer Object Pattern

When developing the History feature on the frontend, there was a need to retrieve multiple data fields through queries. To reduce the number of queries sent to the backend, we made use of the Data Transfer Object pattern [5]. In order to encapsulate these data fields into a single query on the endpoint instead, our team created a 'Session Response' interface.

This 'Session Response' interface is produced by passing a session object (queried directly from the database) through a transformer method. The interface will operate as the Data Transfer Object which contains only the fields that the receiver needs to carry out its work.

The alternative to the Data Transfer Object pattern would be to leverage on the model's schema. However, this would expose unnecessary details and result in a reduction in performance to get resources as there would be a greater number of remote calls for data from different services.

Thus, we decided to implement this pattern to cut down the network overhead associated with the additional calls.

6.4. Microservices

6.4.1. User MS

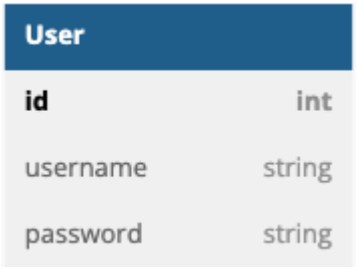

Figure 6.4.1a. User DB Schema Diagram

This microservice handles the creation of users for the web application. When a user creates an account for PeerPrep, a unique id is automatically generated by increment and assigned to that account. The password will be hashed and salted before being stored.

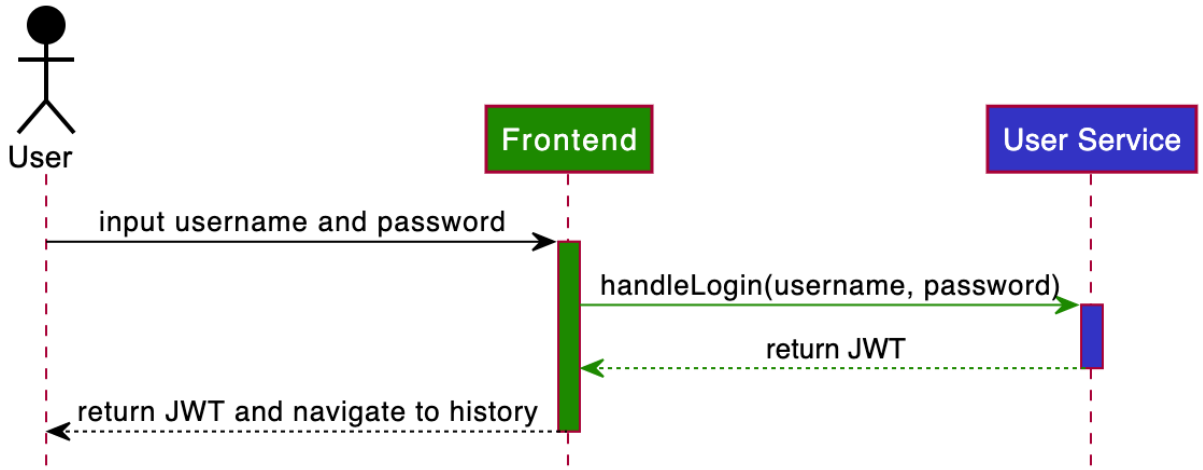

Figure 6.4.1b. Sequence Diagram for user login

User authentication is also handled in this microservice. When users log in on the frontend, the user microservice will issue a JWT containing id and username. When other services require user authentication, the frontend will send the JWT to the backend for verification. If the JWT is invalid or expired, the user will be denied access.

6.4.2. Matching MS

Figure 6.4.2. Room DB Schema Diagram

As stated in 6.3.2, Matching MS uses the Pub-sub pattern. Through the use of Pub-sub pattern, this microservice facilitates the pairing of two users into a room. This microserviced also manages the room by giving clients access to information about the room as well as the ability to close/leave it.

By maintaining a map that maps the user's identity to the socket id, we can keep track of who the user is. The matching page can only be accessed by one instance of a user at a time, and we will disconnect the user if they attempt to connect using additional browsers or tabs.

There are three difficulties that the users can select from. If there are two users in the same difficulty queue at the time of joining, there will be a room created for both users. The users who created the room can access the room information, as well as close or exit the room.

6.4.3. Questions MS

Figure 6.4.3. Questions DB Schema Diagram

This microservice helps to store and provide sample technical interview questions. Also, provides functionality such as the generation of a random interview question based on the user-selected difficulty.

This microservice also allows the system administrators to manage the data in the questions database so that the list of recommended questions can be easily maintained.

6.4.4. Collaboration MS

As stated in 6.3.2, collaboration MS uses the Pub-sub pattern. Through the use pub-sub pattern, this service synchronizes the state of the code of the users in the room. On a code change, the user will send the updated code to the other user for them to update their state. On each update, we also use a Redis server to store the code state so that when users refresh and relog in, they are able to retrieve the current code state.

6.4.5. Communication MS

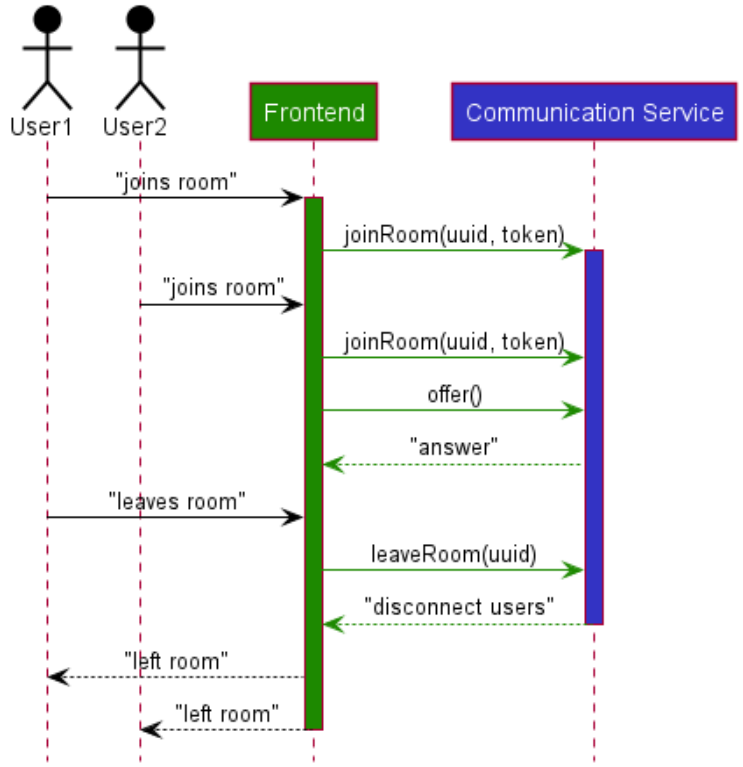

Figure 6.4.5. Sequence Diagram to establish video calls

When establishing a signaling channel for a video call, the Communication MS will first authenticate the user by verifying the JWT. Once verified, the user then can join a room specified by a UUID. The UUID will only be given to users who are matched based on difficulty level and the UUID is used to communicate between 2 users via WebSocket. When both users join the room, the user that joined earlier will initial a Real-Time Communications (RTC) connection with another user through the simple-peer library. The sequence diagram below shows how the Frontend interacts with the Communication MS.

6.4.6. History MS

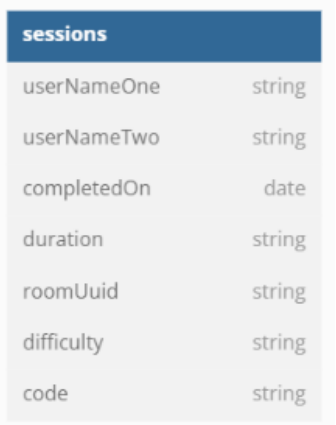

Figure 6.4.6. History DB Schema Diagram

The history MS stores and maintains all past interview data. It provides APIs to store interview session information. Also, sorts and retrieves all relevant data pertaining to the user after the request is successfully authenticated.

6.5. Program Executions

This section will illustrate the main program flow and API executions when a user is using the software application.

6.5.1. User signs up and logs in

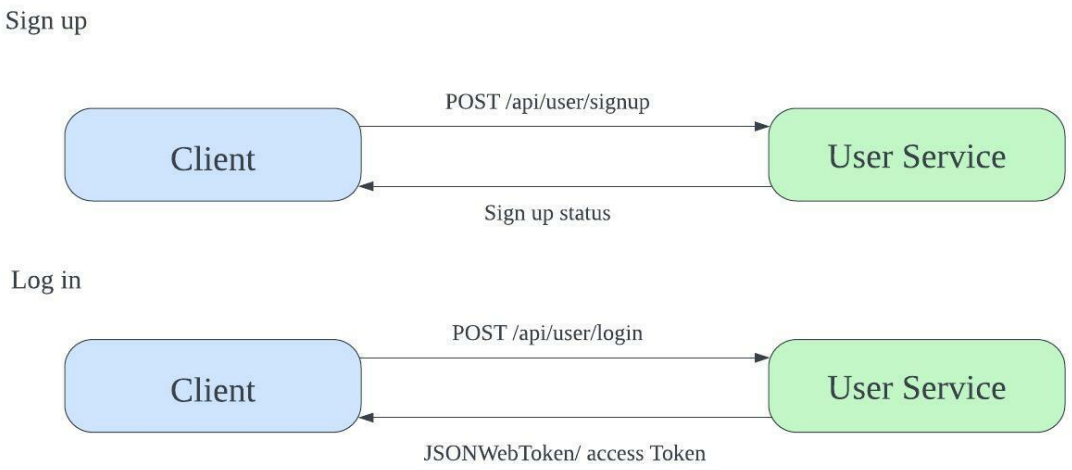

Figure 6.5.1. Sign up and Log in diagram

When the user signs up or logs in, the frontend client will communicate with the User Service. After signing in, the User Service will return a JWT token and then store it in a cookie. Whenever the client makes a request to an API that requires authentication, the cookie will be attached to the request and the JWT token is used for verifying a valid request by the microservices.

6.5.2. Match users

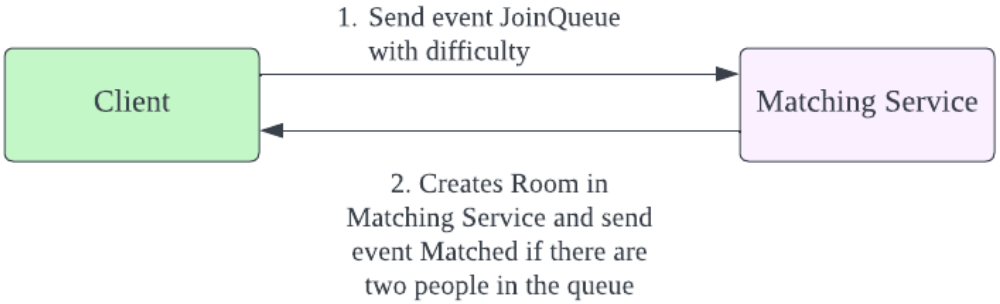

Figure 6.5.2. Matching Diagram

After logging in, the user can select the difficulty they want to be matched with. If there are two people in the queue of the same difficulty, they will be matched and a room will be created.

6.5.3. Establish Room with Video Call and Code Editor

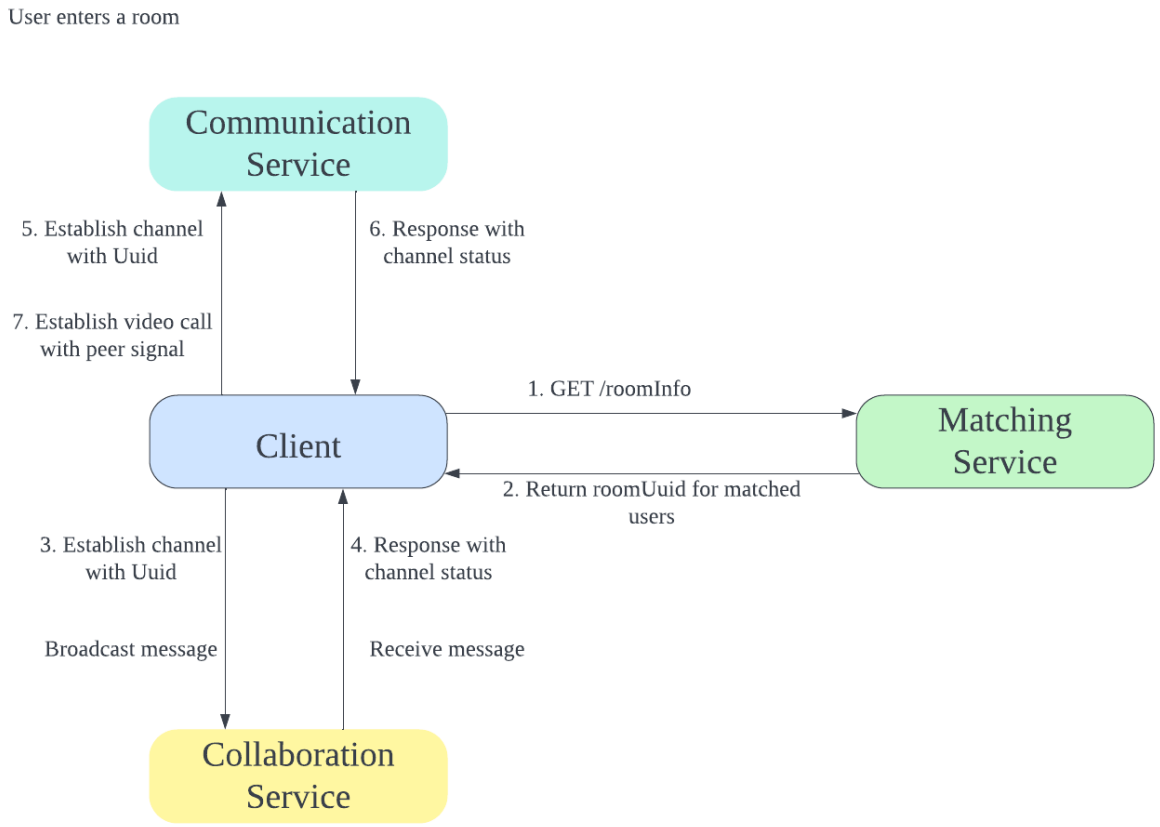

Figure 6.5.3a. User Enter Room Diagram

After being matched, the user will be directed to the Room Page. The client will send a request with the access token to the Matching Service to get a UUID that both users that matched can use. The UUID along with the access token will be sent to establish a connection with WebSocket in Communication and Collaboration services. Then through the Communication Service, the users will send and accept the signal for WebRTC connection.

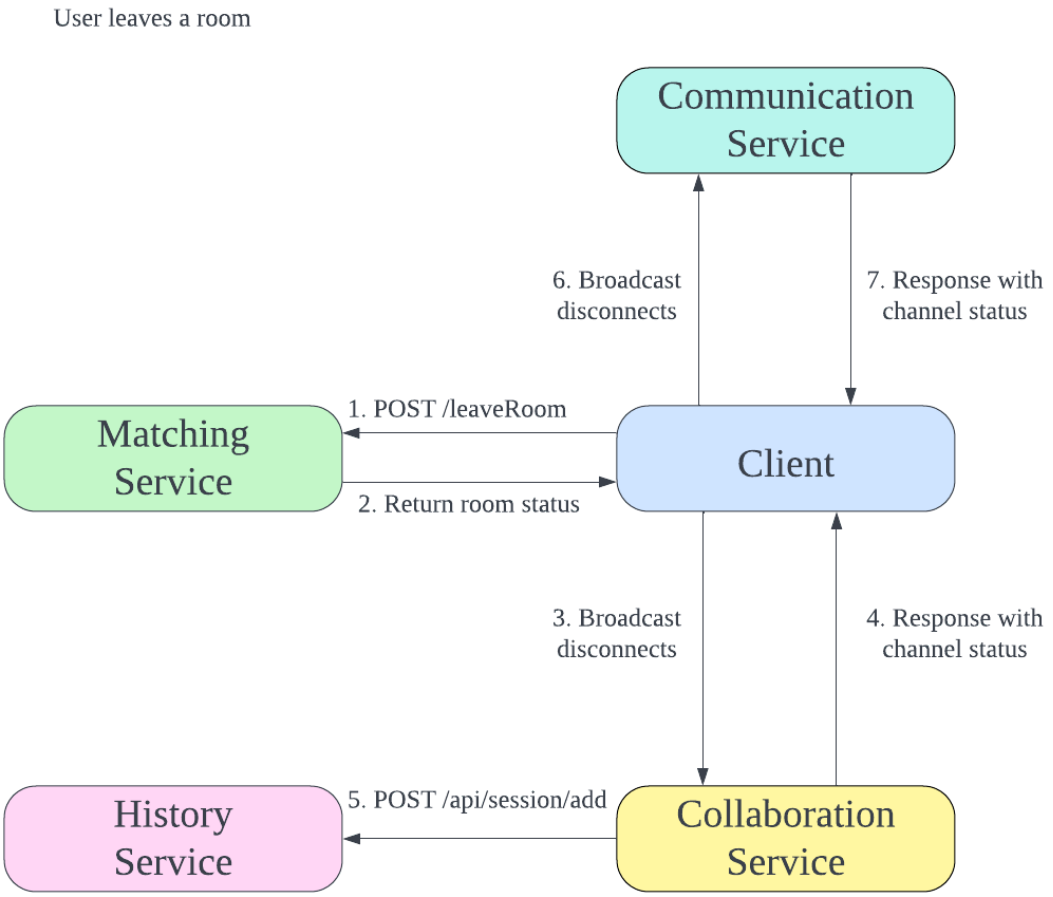

Figure 6.5.3b. User Leave Room Diagram

When a user leaves the room, the client will send a leave request to Matching Service to remove the room UUID from the database. Then the client will broadcast “disconnect” to both Communication and Collaboration Service to disconnect all the users in the respective channels. Last but not least, the collaboration service will retrieve the interview session information from Redis and send the data to the History Service.

6.5.5. Generate Random Interview Questions

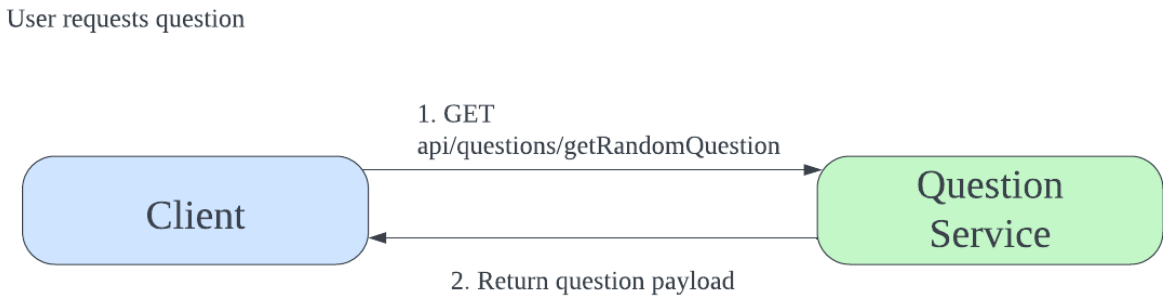

Figure 6.5.5a. User Requests Question Diagram

While the pair of users are in the room, either user can click on the ‘Question’ button or the ‘Next’ button within the question modal to prompt the client to send a post request (containing the room’s difficulty in the header and the JWT stored in context) to the Question Service. The Question Service will then generate a random question with the given difficulty and return it to the client which will format the data and allow the user to copy the question to their clipboard to paste it on the IDE.

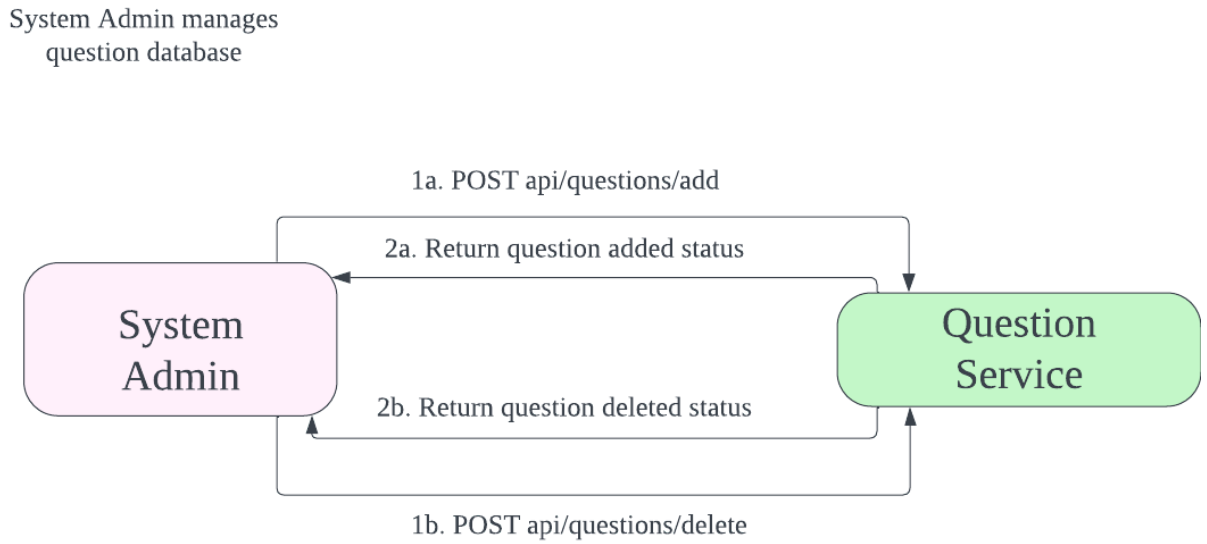

Figure 6.5.5b. System Admin Manages Question Database Diagram

The system administrator will be able to add and delete questions to the questions database hosted on MongoDB Atlas by sending requests to the Question Service with the question details in the request body. Note that only system admins with the secret MongoURI key can access and alter the questions database.

6.5.6. Display Past Interviews

Figure 6.5.6. History Diagram

In the History Page, the client will send a request to retrieve all past interview data relevant to the user with the cookie to the History Service. Based on the cookie, the Service will return relevant data pertaining to the user. Finally, the client will format and display the data to the users.

6.6. Database Design

The databases from User, Matching, History and Question services are separated. The benefits are reduced complexity of the schema design, and increased testability and deployability as the databases are independent of one another. These services require data persistence to keep sensitive and important information safe such as users' details and their interview data. Additionally, the matching service stores the pairing information in non-volatile storage to ensure that the users will be connected to the same room in case of unintended disconnection or temporary server failure.

The Collaboration Service uses Redis as its database for caching information such as code during the mock interview. The code editor should be highly interactive and responsive with text appearing in real time for the interview process to be effective. Hence, Redis is suitable as Redis data resides in memory, which enables low latency and high throughput data access. When the interview is disrupted such as a loss of connection, the data can be retrieved quickly when the user joins back the room.

6.7. Libraries

The following are some of the noteworthy libraries that we have used.

- **React:** React is a JavaScript library that we have used to build our front-end interactive user interface.
- **Codemirror:** We used codemirror as our code editor component. Codemirror comes with a lot of features built-in and supports multiple programming languages.

- **Socket.IO:** Socket.IO enables low-latency event-based communication between clients and servers.
- **Simple-peer:** We used simple-peer for audio/video communication between users by allowing the transfer of files over WebRTC.
- **Jsonwebtoken:** We used the JSON web token library to generate and decrypt JWT for authentication purposes.
- **Material-UI (MUI):** We used MUI for most of our frontend design for ease of styling with its highly modular and extensible components.
- **Tailwind:** The Tailwind framework is used to provide standardization of CSS code in our frontend components.
- **Bcrypt:** Used for hashing and salting of passwords for improved security

6.8. Other Design Considerations

6.8.1. Question Pool Management

Our current question pool implementation uses MongoDB Atlas which is a fully-managed cloud database to store the questions. The system administrators would be able to add and remove questions from this database if they have the MongoDB URI secret key.

An alternative design that we were considering for the implementation of the question database was to use a third-party API to retrieve questions directly from an interview question site such as Leetcode API. An advantage of this would be the existing volume present in these third-party servers, as well as the standardization of data.

Ultimately, we decided to go with MongoDB Atlas. A few of the considerations that led to this decision. First, we wanted to be able to directly moderate the level of difficulty for questions based on user feedback. Next, we wanted to ensure that questions are always available to users at any point during usage. Using a third-party API may potentially be error-prone due to the dependency on the third-party API. This may affect the availability of the question service.

Another alternative we decided against was to let users input their own questions. However, to maintain quality control and standardization of difficulty levels, we decided that the question pool management should be exclusively handled by system administrators.

6.8.2. Authentication

As our group decided to use the Shared Data pattern on the frontend to transmit data between the various microservices, there is a certain degree of exposure to malicious intent if users attempt to abuse our backend endpoints from the network. To prevent this, we implemented a requirement for requests to our backend to contain a JWT token for the authentication of each request. This ensures that each request to the backend is validated since the JWT has to be issued from the backend encrypted with a secret key.

6.8.3. Frontend Design

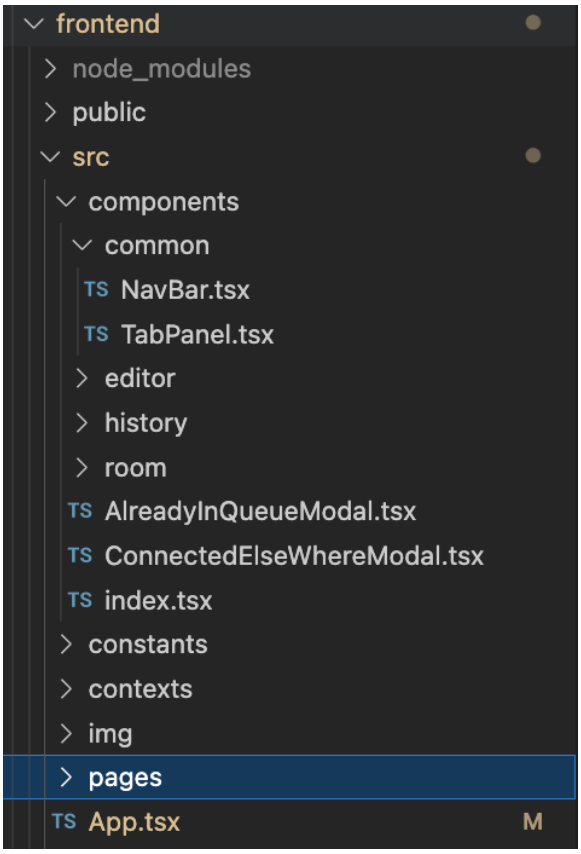

Figure 6.8.3. Frontend design structure

In accordance with the DRY (Don't Repeat Yourself) principle, we compartmentalized our frontend into different components, with components that were widely used throughout the application (such as the navigation bar and tab panel) grouped under a "common" directory. By ensuring that we are not repeating code, we facilitate the ease of code reuse while preventing possible errors resulting from changes to existing components.

7. Development Process

Our group used the Agile software development process. We usually have a two-week sprint and weekly stand-ups where we discuss our progress and address any potential roadblocks.

Sprints	Wk 3 - 5	Wk 6 - 7	Wk 8 - 9	Wk 10 - 11	Wk 12 - 13
Brendon	Set up development environments Implement front-end for Match Page	Set up development environments Integrate frontend and backend for matching users	Set up development environments Implement front-end for Room page and backend services Implement Communication microservice Integrate video chat to frontend	Set up development environments Implement History microservice	Final touches Completing the project report
Elton	Setup linters Create matching microservice	Implement matching	Implement collaboration	Integrate history service into the matching and room workflow	
Justin	Create user service Implement Navigation Bar Implement hashing and salting of passwords	Implement frontend features for deleting accounts and changing passwords Integrate frontend and backend for change password feature	Implement question microservice backend Populate question database with questions	Implement history service frontend (Pair programming)	
Yuan Ker	Incorporate use of JWT for user service	Implement backend functionality for delete and	Implement Question Microservice frontend	Implement history service frontend (Pair programming)	

	Integrate login and signup page on frontend	changing password Implement settings page on frontend			

Figure 7. Sprints

8. Reflection

8.1. Potential improvements/enhancements

8.1.1. Support for more languages

Only Python and Python syntax highlighting are supported by our project. The number of users who can use our program is therefore significantly constrained. Allowing users to choose popular languages (like Typescript and C++) to practice with is something our team believe would be beneficial.

8.1.2. Support for a text-based chat

We only allow video calls for communication with interview partners. We are aware that there are instances when video calls cannot be used to converse. For example, when there is a lot of background noise, we can address any issues that users may have with video calls through text chat. However, it should be noted that users are currently still able to communicate with each other through text via the collaborative IDE.

8.1.3. Support for persistent state management

We use React Context hook to manage states globally. The drawback of using a Context hook is that every time the component re-renders it has to fetch and recompute data which can be expensive. Moving forward, we could use a specialized state management library such as Redux to reduce the expensive operations and be highly scalable with the size of the project.

8.1.4. Support for topic-based matching

Currently, PeerPrep only supports matching users based on question difficulty level. A possible extension to this could be the addition of a topic filter mechanism for our matching metric. This would allow users who are weaker in a certain area to get more targeted practice in specific topics.

8.1.5 Support for user profiles

A good quality-of-life feature would be to include user profiles. This allows users to establish contact with one another or learn from each other by looking at their profile statuses. Currently, this was not a feature introduced in PeerPrep because it is not a high-priority feature for users to be able to conduct mock interviews. However, it can be a good candidate for a future extension because users might be able to get higher quality mock interviews if they know more about each other's strengths and weaknesses by looking at the profile page.

8.2. Learning points from the project process

Due to the use of the microservices pattern, our team developed each microservice independent from one another in terms of design. Another method of development would have been to design each microservice with considerations of other microservices in mind so that we would be able to improve cohesion between the microservices.

8.2.1. Microservice architecture

Through this project, we have gained an understanding of the functions of microservices and the reasons they are such widely used architecture in the industry. This architectural design minimized roadblocks during development and made it simple for various team members to work on various project components simultaneously.

8.2.2. Design Patterns

Through this project we get to apply the design principles and patterns learnt from the lectures. Firstly, design patterns gave us a common goal and vocabulary to work and communicate. For example, everyone has a common understanding of microservice architecture hence we are able to discuss the benefits and its implementations effectively. Secondly, design patterns reduce the common problems developers faced during development. For example, using the MVC pattern to build web applications for easier planning and maintenance. Last but not least, design patterns allow us to improve the reusability of code as we can use a common solution to recurring common problems.

9. Effort

This section will highlight some of the exceptional “must have” and the “nice to have” features to demonstrate our efforts in going above and beyond what is required.

9.1. Fanciful UI

As described in our stakeholder analysis, a key success factor to ensure that our product is highly usable for users spanning different levels of technical competency is to ensure we have a minimalistic design. This is achieved by having a linear usage path rather than a cluttered UI with superfluous features that may distract users from the main function of our platform. Despite having a minimalistic design, we ensured that a certain degree of aesthetics is maintained. To achieve this, what our team focused on was consistency and ensuring that there is congruence and synergy between the different components. Some examples are the use of different tabs on the settings page, the single page toggle on the login and sign-up page, as well as the use of modals to display questions when users are in mock interview rooms.

9.2. Syntax Highlighting and Keyboard Shortcuts

To closely replicate most of the ‘trendy’ code editors that are used in most technical interviews, we implemented syntax highlighting to allow both the interviewee and the interviewer to be better able to understand the code due to its enhanced readability. For greater ease of programming, we also implemented standard keyboard shortcuts that most IDEs share.

9.3. Resilience to Non-User Initiated Disconnection and Page Refresh

Data is non-persistent on the client-server hence any disconnection or page refresh will cause data loss. We have implemented many fail-safe features to ensure that our frontend is reliable and always available in any event. When a user disconnects from the room or closes the web application, there are three prongs to this problem:

Firstly, they will not be able to rejoin the same room. To tackle this, we implemented a database to store the user’s socket and room information in Matching MS. Hence, even when the user disconnects or the server is down temporarily, they would be able to rejoin back to the same room until one of them chooses to leave the room.

Secondly, the data on the code editor will be lost and using a database to store and retrieve the data will greatly dampen the user experience as it will be too slow. The solution is to use

Redis, a fast in-memory data store to save and retrieve the progress on the code editor in the Collaboration MS.

Lastly, the shared data such as the user's information and component states will be lost. To overcome this issue, in our data pool and data producer classes such as the SocketContext, we implemented the fetching of data with the useEffect and useMemo hooks. Hence, whenever the components re-renders or the page refreshes, the data will be available for other components to consume.

9.4. Audio Visual Communication

To mimic real-life interview experience, we implemented video chat features so that the users can communicate seamlessly with one another. One important aspect to highlight is that video chat allows users to learn to read verbal cues and body language. As a result, the users would become effective in communicating and presenting themselves in the interview. To handle the communication between users, we implemented the Communication MS and used WebRTC in the software.

9.5. History Service

When users track their progress, they will become more purposeful and motivated about their work. We implemented the History MS to handle storing and tabulating relevant data pertaining to past interview sessions. Through the History Page, we enable the users to review their past interview sessions and make improvements to their interview skills. The information we displayed is essential to the users' growth such as the duration of the interview where interviewees are required to perform under time constraints in actual interviews.

9.6. Authentication for All MS

In our design considerations, we discussed the importance of authenticating every request made to the backend microservices to prevent malicious abuse of our API. Since the microservice architecture suggests that we should reduce and avoid coupling and dependencies between different MS, we decided that an optimal way to allow authentication throughout all MS would be to leverage on the Shared Data Pattern to get the access token. This way, we would be able to ensure that a user is authenticated before being able to access any of our microservices.

9.7. Dockerizing All MS

Due to the number of MS required for the full PeerPrep Web Application to be deployed, our team decided that the most sustainable way to launch all backend services on docker, using docker-compose. As each MS has different launch requirements, dockerizing allows for a

more streamlined development process as well as a more optimal infrastructure for deployment on cloud platforms in future.

10. Product Screenshots

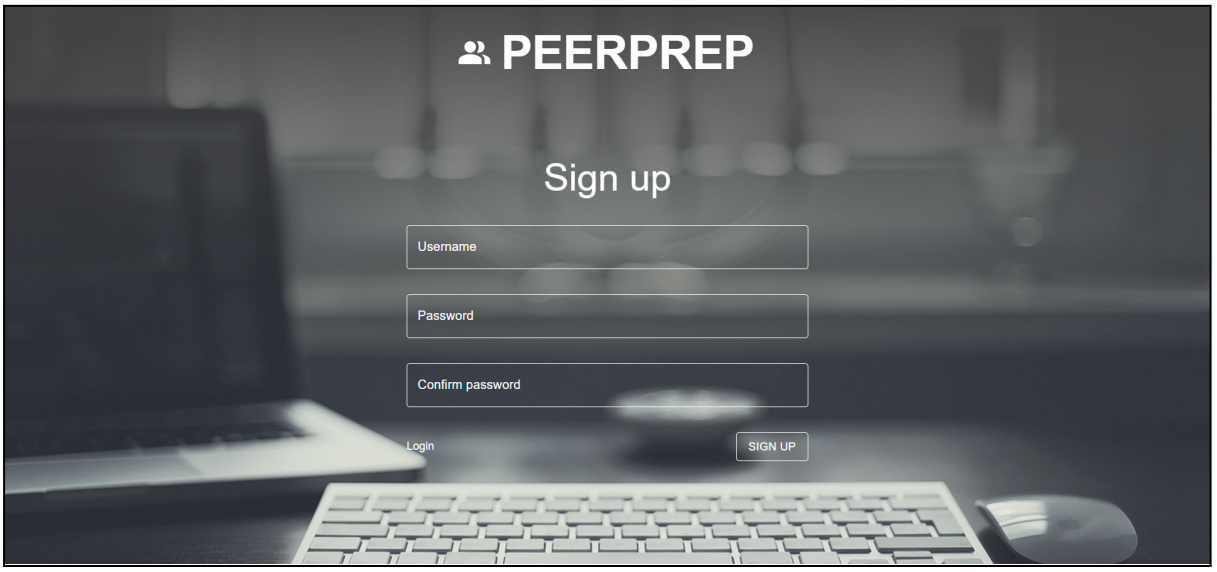

Figure 10.1a. Image showing Sign up page

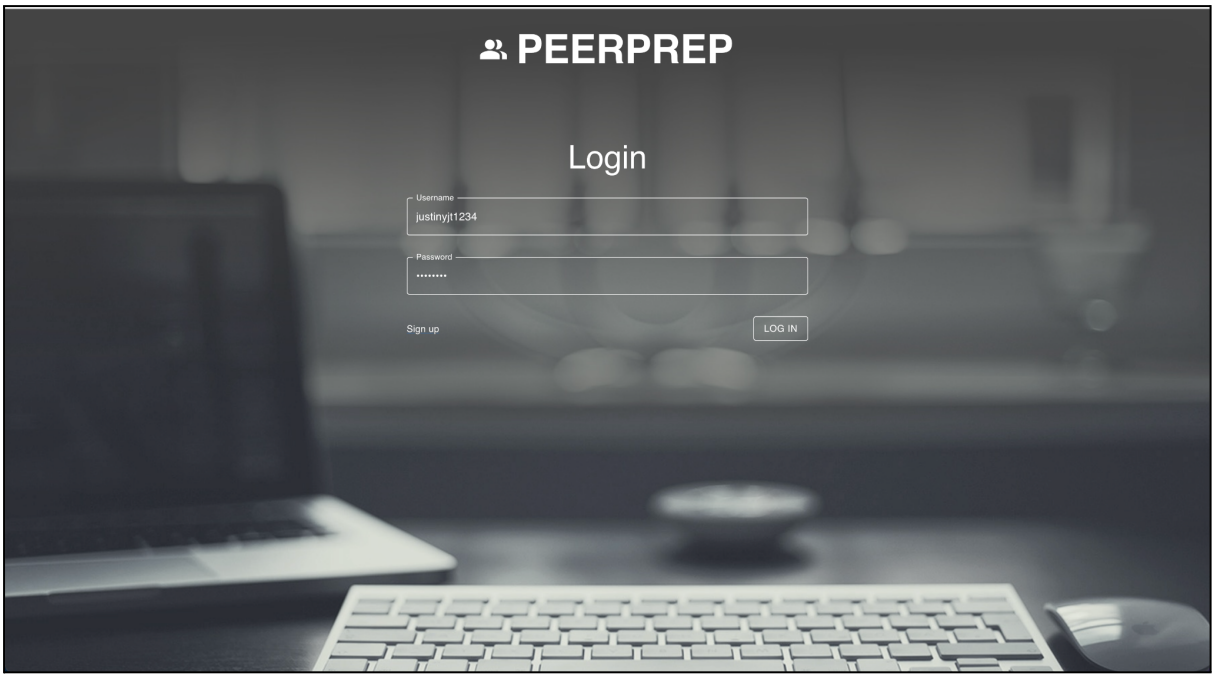

Figure 10.1.b Image showing Login page

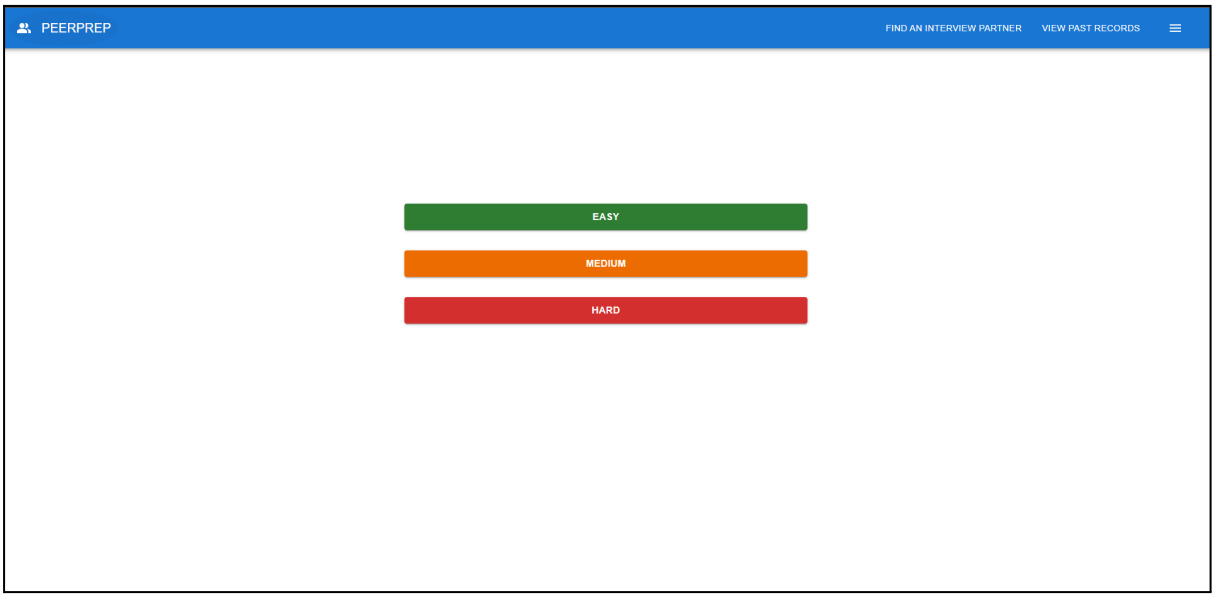

Figure 10.2. Image showing the matching page

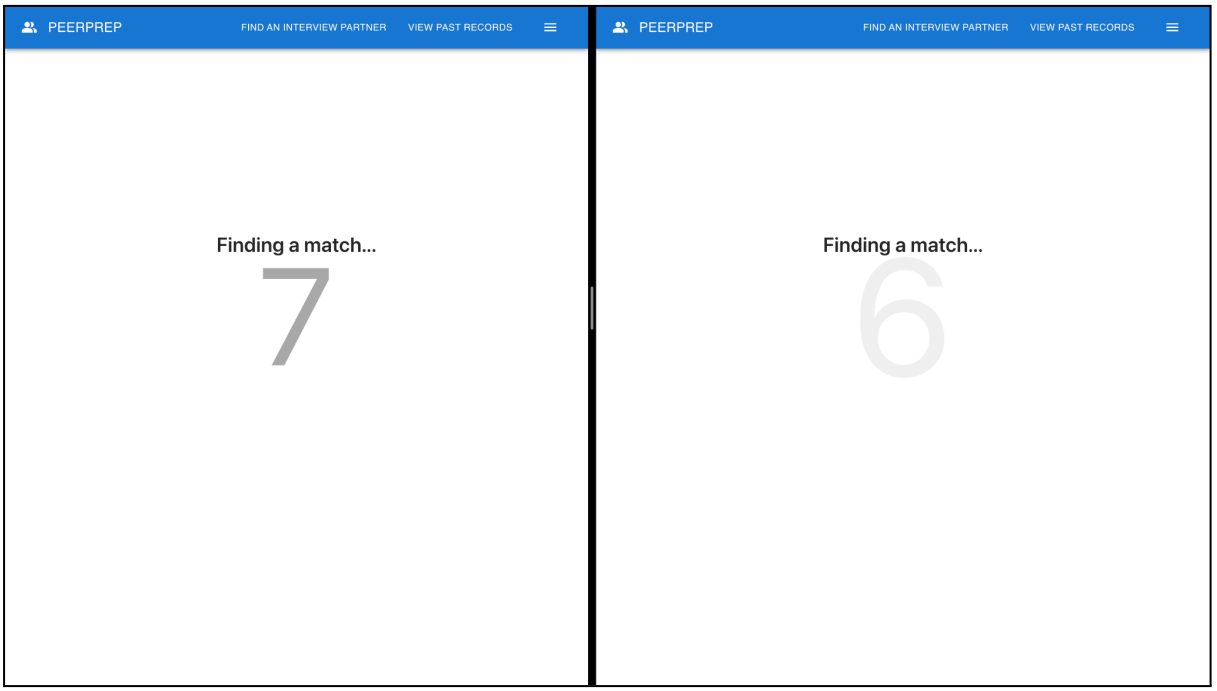

Figure 10.3. Image showing two users matching

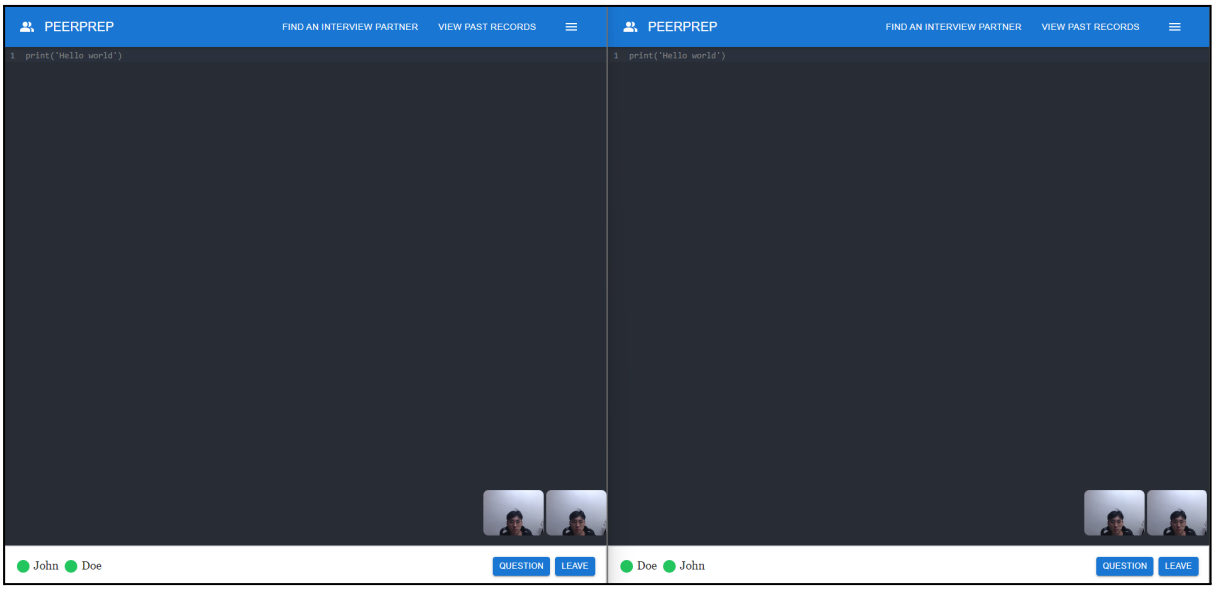

Figure 10.4. Image showing two matched users in the same room

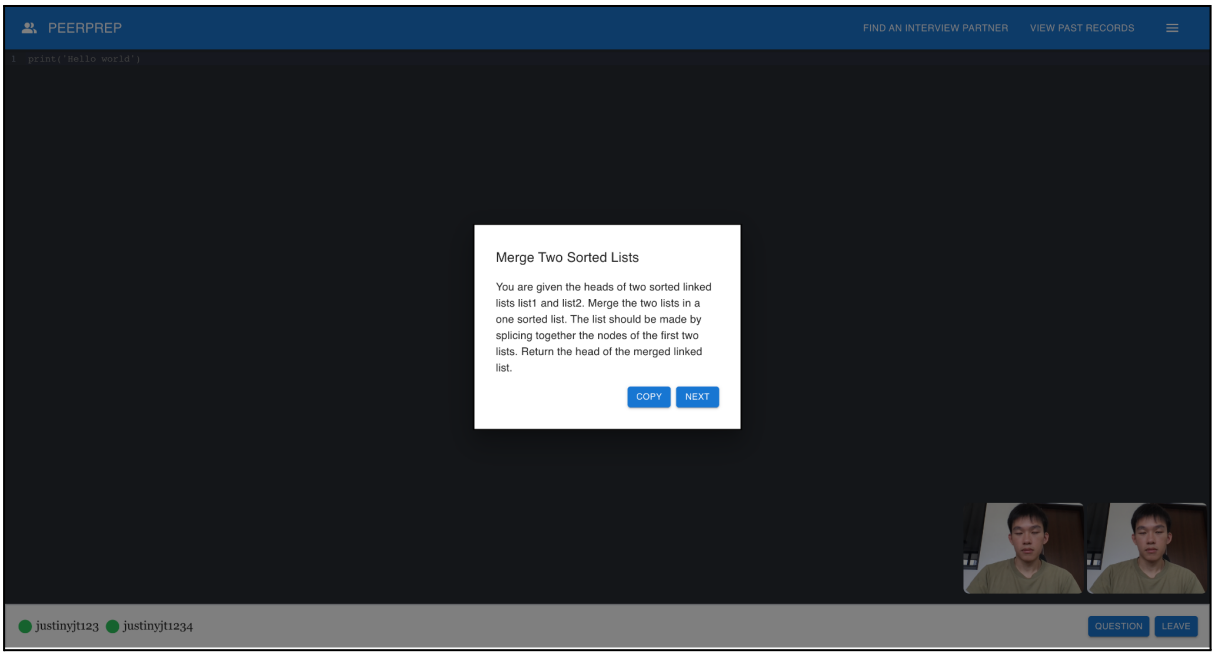

Figure 10.5. Image showing randomly generated question after user clicked on question button

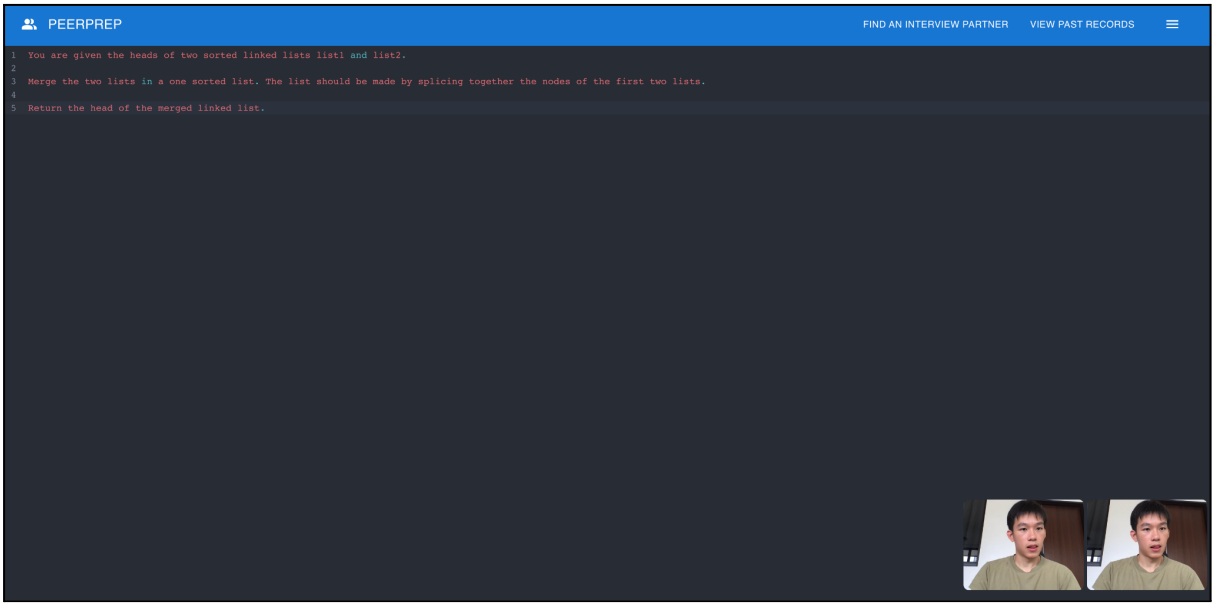

Figure 10.6. Image showing question pasted on the IDE from the randomly generated question.

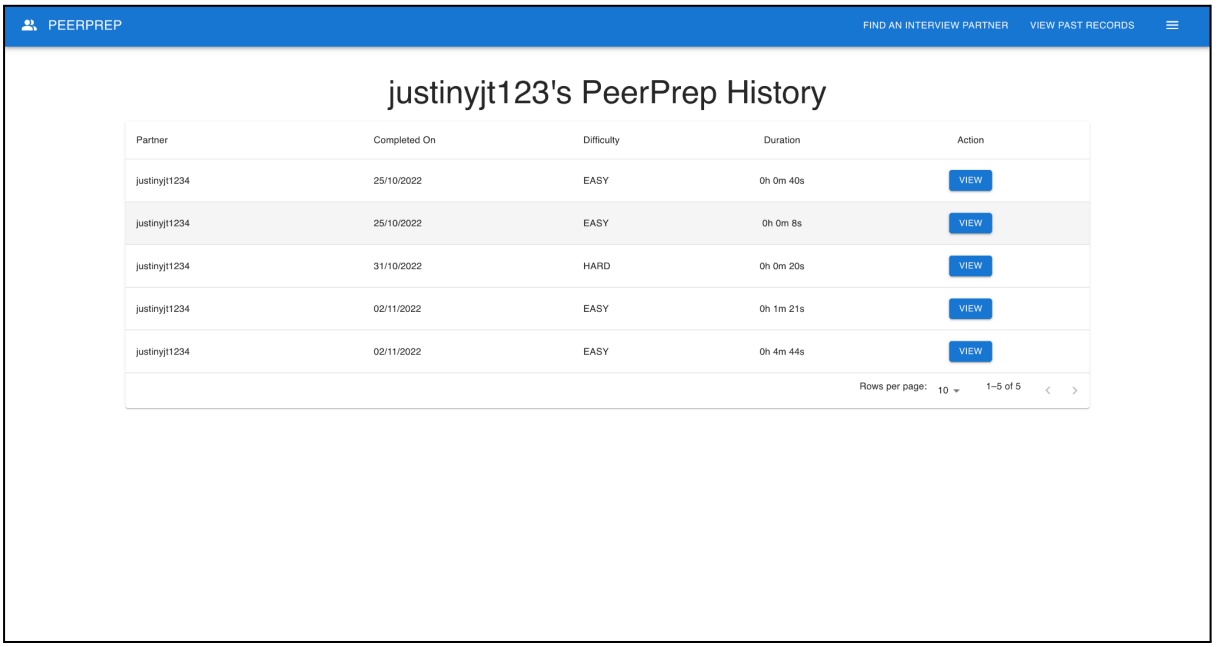

Figure 10.7. Image showing user history

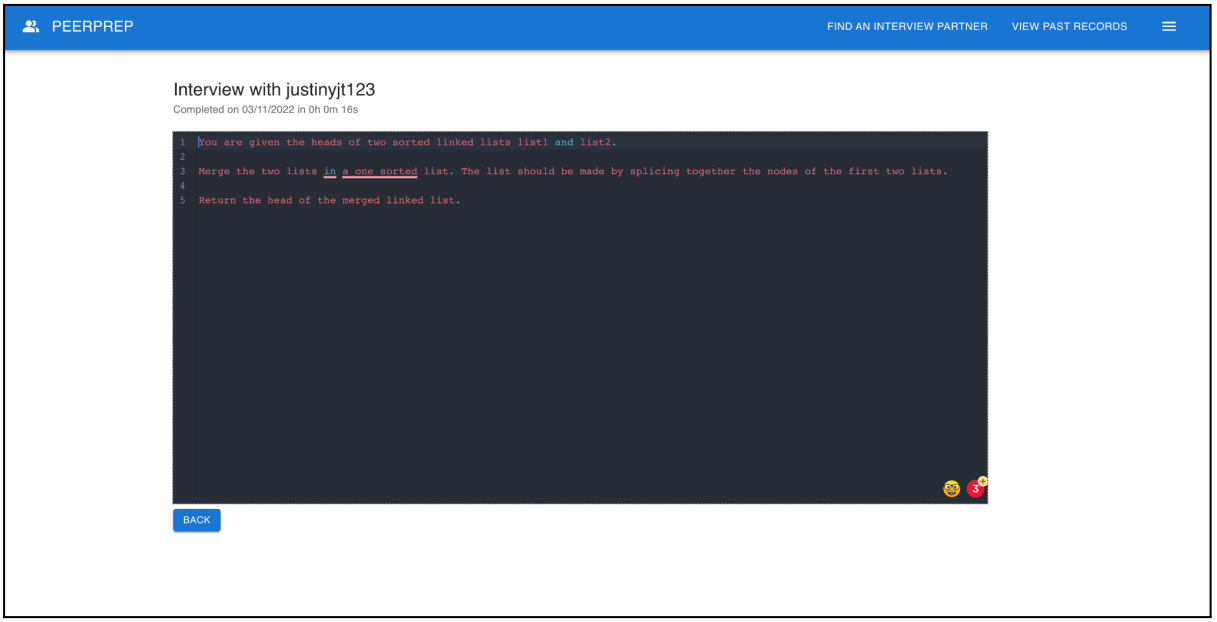

Figure 10.8. Image showing code from a specific session between two users

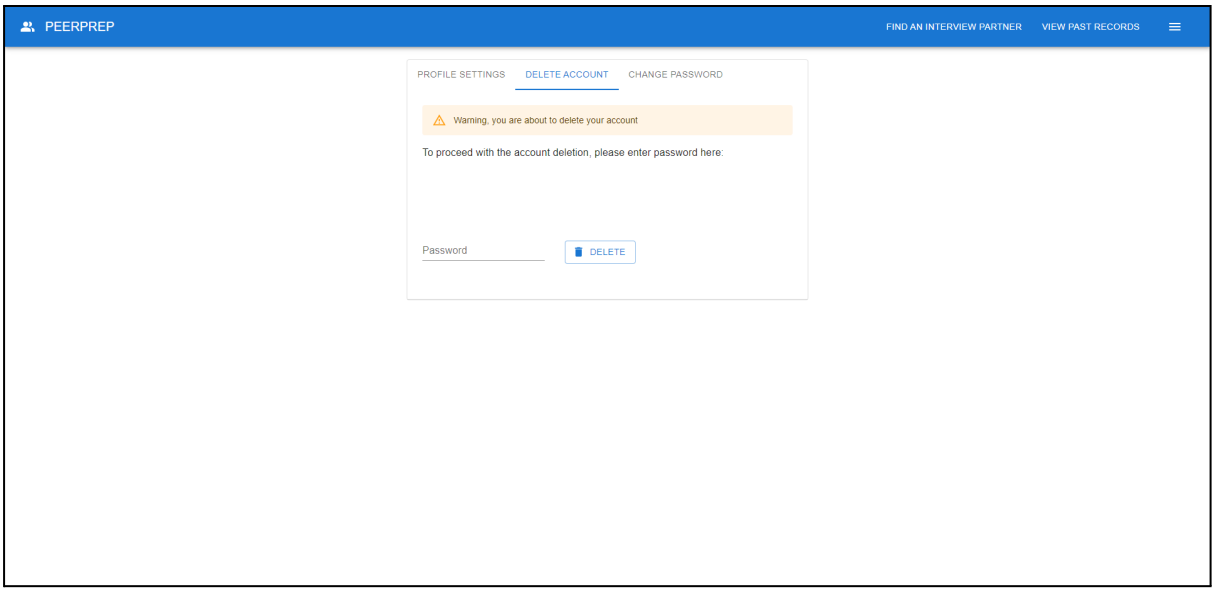

Figure 10.9. Image showing the delete account page for user account deletion

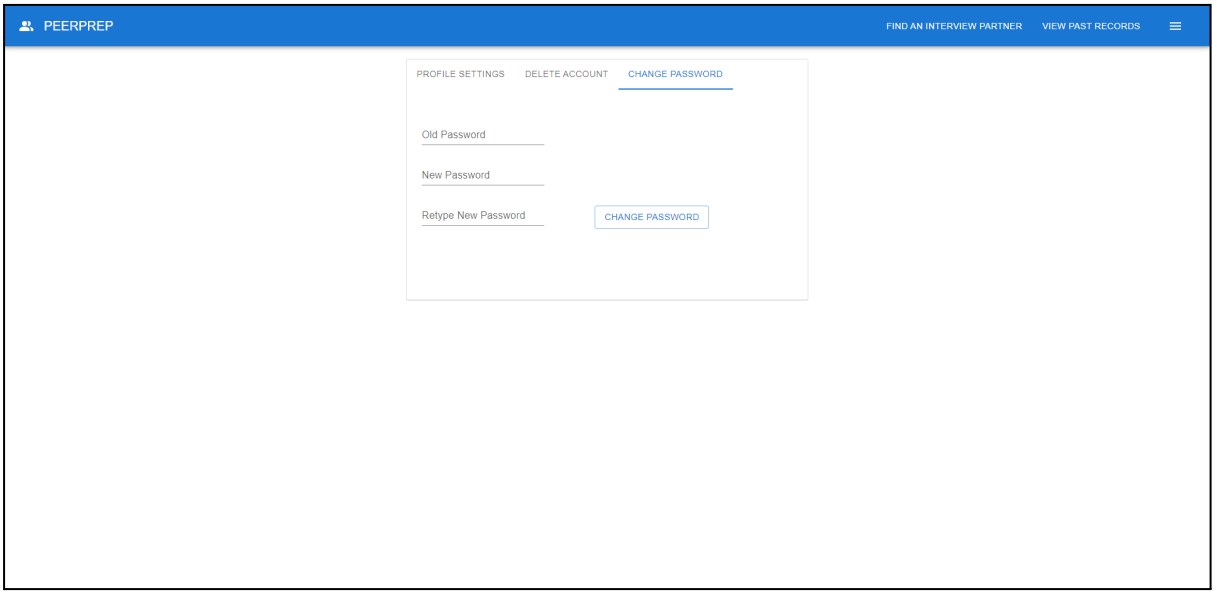

Figure 10.10. Image showing change password account page for user to the change password

11. References

[1] C. Richardson, “Microservices Pattern: Microservice architecture pattern,” *microservices.io*. [Online]. Available: https://microservices.io/patterns/microservices.html. [Accessed: 07-Nov-2022].

[2] D. Suthar, “Implementing publisher-subscriber pattern using JavaScript, nodejs and WebSockets,” *Medium.com*, 09-Dec-2018. [Online]. Available: https://medium.com/unprogrammer/implementing-publisher-subscriber-pattern-using-javascript-nodejs-and-websockets-82036da7e174. [Accessed: 07-Nov-2022].

[3] D. C. Rajapakse, “Software Design Patterns,” *CS2103/T - Software Design Patterns*. [Online]. Available: https://nus-cs2103-ay2223s1.github.io/website/se-book-adapted/chapters/designPatterns.html. [Accessed: 07-Nov-2022].

[4] S. Kappagantula, “Everything You Need To Know About Microservices Design Patterns,” *Edureka.co*, 26-Mar-2022. [Online]. Available: https://www.edureka.co/blog/microservices-design-patterns. [Accessed: 07-Nov-2022].

[5] “The DTO pattern (data transfer object),” *Baeldung*, 19-Sep-2022. [Online]. Available: https://www.baeldung.com/java-dto-pattern. [Accessed: 07-Nov-2022].