



**CS3219 Final Report**  
**PeerPrep**  
**Team 15**

Ee Kar Hee, Nicholas (A0214353N)

Ong Xing Wei (A0217682Y)

Tan Yi Guan (A0217680A)

Teo Sin Yee (A0222912N)

<b>1. Introduction</b>	<b>4</b>
1.1. Background	4
1.2. Document Conventions	4
1.3. Intended Audience and Reading Suggestions	5
1.4. Project Scope	5
1.5. References	5
<b>2. Overall Description</b>	<b>6</b>
2.1. Product Perspective	6
2.1.1 Product Overview, Context and Scope	6
2.2.2 Stakeholder Analysis	6
2.2.3 Usage Process	7
<b>3. Individual Contributions</b>	<b>11</b>
<b>4. Requirements</b>	<b>11</b>
4.1 Functional Requirements	11
4.1.1 User Management Service	11
4.1.2 Matching Service	12
4.1.3 Question Service	13
4.1.4 Coding Service	13
4.1.5 History Service	14
4.1.6 Communication Service	14
4.2 Non-functional Requirements	15
4.2.1 Security	15
4.2.2 Performance	15
4.2.3 Robustness	16
4.2.4 Portability	16
4.3 External Interface Requirements	17
4.3.1 Hardware Interfaces	17
4.3.2 Communication Interfaces	17
4.3.3 Software Interfaces	17
4.4 Feature List	18
<b>5. Developer Documentation</b>	<b>19</b>
5.1 Tech Stack	19
5.1.1 Frontend	19
5.1.2 Backend	19
5.1.3 Third-Party Libraries	20
5.2 Architecture	21
5.3 Frontend	23
5.4 Microservices	24
5.4.1. User Management Service	24
5.4.2 Question Service	25
5.4.3 Matching Service	26

5.4.4 Coding Service	27
5.4.5 Room Service	28
5.4.6 History Service	29
5.4.7 Communication Service	29
5.4.8 Microservices Intercommunication	31
5.5 Design Patterns	31
5.5.1 MVC Pattern	31
5.5.2 Pub-Sub Pattern	32
5.5.3 Facade Pattern	33
5.5.4 Repository/DAO Pattern	34
5.5.5 Data Transfer Object (DTO)	34
5.6 API Calls	34
5.6.1 Auth Events	34
5.6.2 Dashboard Events	36
5.6.3 Matching Events	37
5.6.4 Room Events	38
5.6.5 Coding Events	40
5.7 Other Design Considerations	42
5.7.1 Communication Service	42
5.7.2 Email and password-based authentication	42
<b>6. DevOps</b>	<b>43</b>
6.1. Sprints	43
6.2 CI/CD	44
6.3 Continuous Deployment	44
<b>7. Reflections</b>	<b>45</b>
7.1 Future Enhancements	45
7.1.1 Integrating Third-Party Authentication Services	45
7.1.2 Collaborative Whiteboard	45
7.1.3 Specific Role Matching	45
7.1.4 Rating System	45
7.2 Learning Points	46
7.2.1 Pros and cons of Microservice Architecture	46
7.2.2 Importance of a well-defined requirement document	46
<b>8. Product Screenshots</b>	<b>47</b>

# 1. Introduction

## 1.1. Background

Increasingly, students face challenging technical interviews when applying for jobs which many have difficulty dealing with. Issues range from a lack of communication skills to articulate their thought process out loud to an outright inability to understand and solve the given problem. Moreover, grinding practice questions can be tedious and monotonous.

To tackle this issue, we are creating an interview preparation platform and peer matching system called PeerPrep, where students can find peers to practise whiteboard-style interview questions together.

This document serves to specify the requirements of this project as well as the design and development decisions made by our team while building PeerPrep.

The working application can be found at <https://peerprep.live> and the repository for PeerPrep can be found here: <https://github.com/CS3219-AY2223S1/cs3219-project-ay2223s1-g15>.

## 1.2. Document Conventions

This document uses the following terminologies and conventions:

Terminology/Convention	Meaning
API	<b>Application Programming Interface:</b> a way for two or more computer programs to communicate with each other
REST	<b>Representational State Transfer:</b> REST is a set of architectural constraints, not a protocol or a standard
RESTful API	APIs that conform to the architectural constraints of REST which are: <ul style="list-style-type: none"><li>• A client-server architecture made up of clients, servers, and resources, with requests managed through HTTP.</li><li>• Stateless client-server communication, meaning no client information is stored between get requests and each request is separate and unconnected.</li><li>• Cacheable data that streamlines client-server interactions.</li><li>• A uniform interface between components so that information is transferred in a standard form. This requires that:<ul style="list-style-type: none"><li>• resources requested are identifiable and separate from the representations sent to the client.</li><li>• resources can be manipulated by the client via the representation they receive because the representation contains enough information to do</li></ul></li></ul>

	so. <ul style="list-style-type: none"> <li>• self-descriptive messages returned to the client have enough information to describe how the client should process it.</li> <li>• hypertext/hypermedia is available, meaning that after accessing a resource the client should be able to use hyperlinks to find all other currently available actions they can take.</li> <li>• A layered system that organises each type of server (those responsible for security, load-balancing, etc.) involves the retrieval of requested information into hierarchies, invisible to the client.</li> </ul>
Technical Interview	A session where the interviewee has to solve a coding problem while communicating their thought process to the interviewer, typically conducted as a part of the hiring process for software engineers
Mock Interview	A practice session that simulates a real technical interview. Both users will take turns being the interviewer and interviewee
OS	Operating system

### 1.3. Intended Audience and Reading Suggestions

This document is intended for all developers, designers, project managers who are interested in contributing to PeerPrep, as well as the teaching team of CS3219.

It is intended for readers to follow this document in sequential order, with the earlier sections providing an overview and the later sections providing insights on the design decisions (following a top-down approach).

### 1.4. Project Scope

PeerPrep is a web application that helps students better prepare themselves for technical interviews. It provides a user-friendly platform for like-minded students to practise technical interviews together, while preserving the realistic aspects of an actual interview. Users will have the opportunity to play the roles of an interviewee and interviewer.

PeerPrep is built for anyone who wants to prepare for their technical interview and improve their chances of securing an internship/job.

### 1.5. References

- [What is a REST API](#)
- [Product Requirements Specification: Product Perspective](#)
- [External Interface Requirements in SRS](#)

## 2. Overall Description

### 2.1. Product Perspective

This section will provide a high level, non-technical overview of PeerPrep. It defines the application in terms of stakeholders needs and demands and determines what features the application should have.

#### 2.1.1 Product Overview, Context and Scope

(The context was covered in greater detail under [Introduction](#))

The product's objective is to provide a mock interview platform for users to practise technical interviews, a common hurdle developers have to get through to secure a job/internship in the tech industry. It mimics an actual interview as closely as possible, providing features such as randomised question generation based on chosen difficulty for “interviewers” to ask the “interviewees”, a collaborative coding space for both parties to write their code, as well as video and voice calling function for real-time communication during the mock interview.

The product can be considered a standalone application with no additional external dependencies, except for a working internet connection as well as a compatible browser. Future developments of this product could come with minimal integrations with third-party authentication services such as GitHub or Google authentication.

#### 2.2.2 Stakeholder Analysis

The first group of stakeholders that directly benefits from this project are the users. Although technical interviews are conducted across all levels, regardless of years of experience, this product mainly focuses on the Data Structures and Algorithms part of the technical interview process. As such, we are mostly focusing on junior developers who wish to practise algorithmic questions and improve their communication skills, as well as senior developers who wish to refresh their algorithmic coding skills as the target users.

To ensure that this product is easily accessible, we have chosen to host it as a web application so that users would not have to download any dependencies to start practising. We have also taken into the consideration that users will have differing ability in solving the practice questions and would want to be paired with someone with similar abilities to have a productive session. Thus, we have set up a system where users would be able to select the difficulty of the questions they would want to practise, allowing the system to match them with another person who wants to practise at the same difficulty level.

Lastly, this product also provides users with a history of all the questions they have done on this platform, whether they got the practice question right and comments given to the interviewee by their mock interviewer.

## 2.2.3 Usage Process

### Overall process:

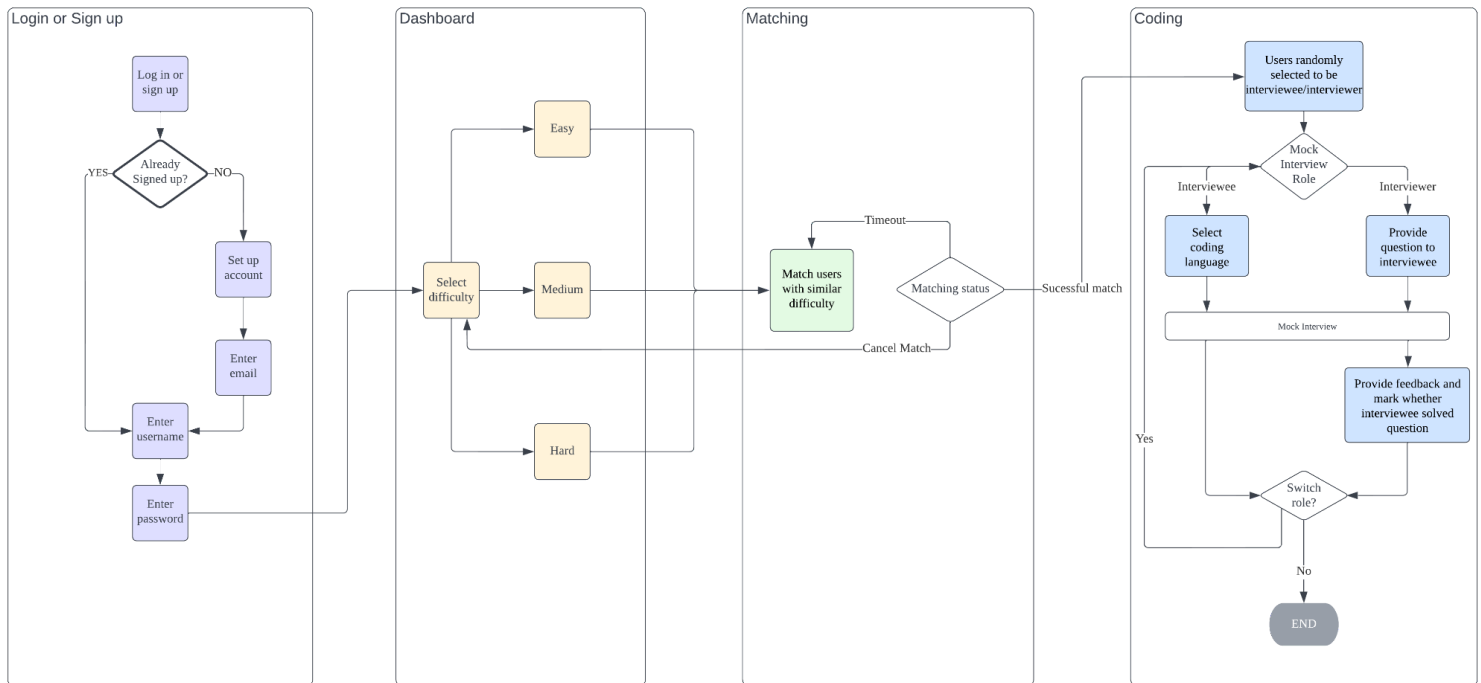
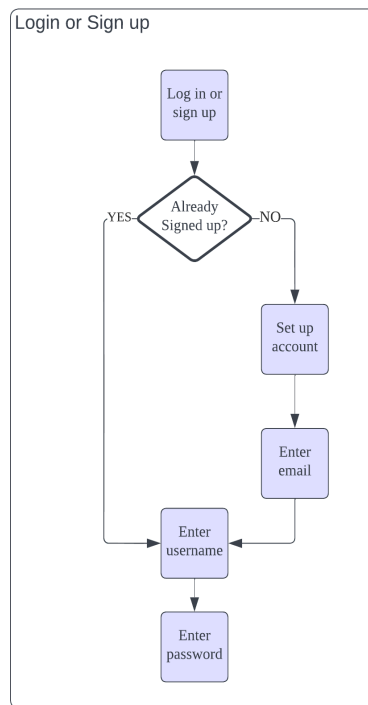


Figure 2.2.3a: Overall User Flow

## Login/Signup Page:

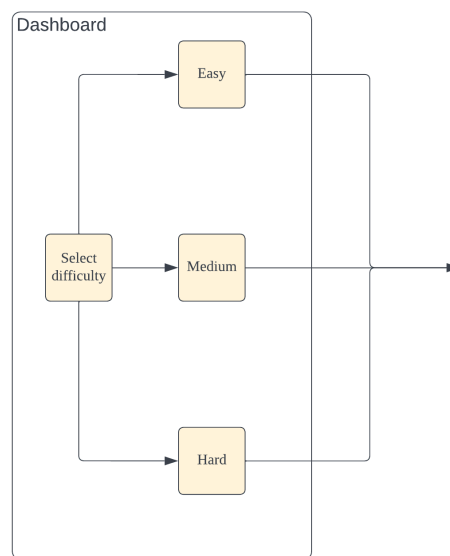


*Figure 2.2.3b: User flow in landing page*

1. A user who visits the PeerPrep app will be greeted with the Login page
2. An existing user would enter their username and password to be authenticated to the app while a new user would have to go through a simple sign up process before being able to use the app.



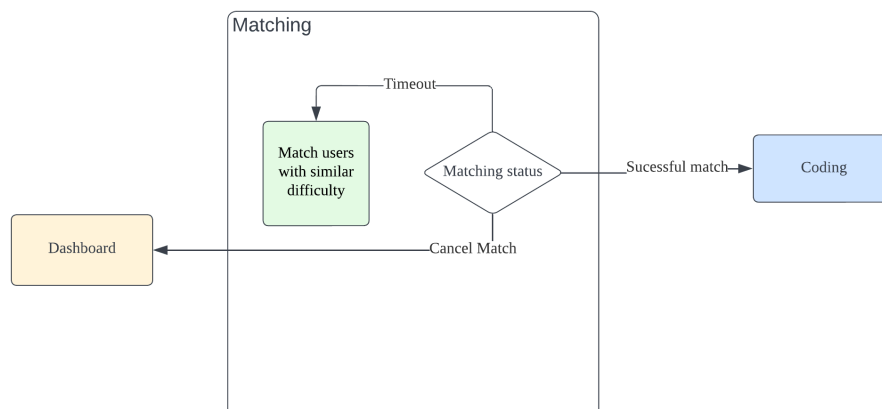
## Dashboard Page:



*Figure 2.2.3c User Flow in Dashboard Page*

3. After signing into the app, the user will be presented with the dashboard, where they will be able to see a history of all the questions they have attempted.
4. Should they wish to start a practice session, they would be prompted to select a difficulty of the question they wish to practise.

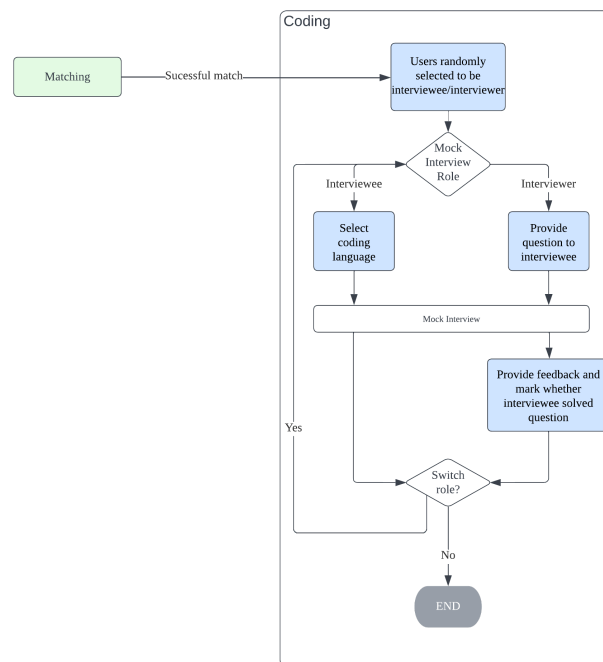
## Matching Page:



*Figure 2.2.3d User Flow in Matching Page*

5. The user will be brought to the matching page where they can wait up to be matched with another user who wants to practise at the same difficulty level. Once matched, they would be brought into the coding room.
6. If the user chooses to cancel the match, the user will be brought back to the dashboard.
7. If the timer times out during the matching process, the user could choose to try matching again or return to the dashboard.

## Coding Page:



*Figure 2.2.3e User Flow in Coding Page*

8. The matched users will be randomly assigned the roles of interviewee and interviewer.
9. The interviewee will be able to select the language they wish to code in and attempt the question prompted. The interviewee will be able to execute the code and the result will be visible to both parties so that the code can be checked for correctness.
10. Once a question has been attempted by the interviewee, they can choose to swap roles and carry out step 9 again. This swapping process could carry on indefinitely until either one of them chooses to leave the practice room.
11. Users can then view the comments left by their interviewer on the dashboard after the practice session is done to review their performance.

### 3. Individual Contributions

Name	Technical	Non-Technical
Ee Kar Hee Nicholas	Implement Frontend Integration of services with Frontend	Final Report Design Frontend
Ong Xing Wei	Implement User Service Implement Room Service Implement Gateway Service	Final Report
Tan Yi Guan	Implement Matching Service Implement Question Service Implement History Service Dockerized Application	Final Report
Teo Sin Yee	Implement Communication Service Integration of services with frontend	Final Report

### 4. Requirements

#### 4.1 Functional Requirements

##### 4.1.1 User Management Service

S/N	Functional Requirement	Priority	Justification
U-1	The system should allow users to create an account with email and password	High	An account should be a base requirement for the application to function as an identity is needed for matching
U-2	The system should ensure that every account created has a unique email	High	Unique emails ensure that each account and its details are only tied to one email which will not cause any confusion
U-3	The system should allow users to log into their accounts by entering their email and password	High	Users should be able to log in to their accounts to use the application and access their own practice history
U-4	The system should allow users to log out of their account	High	Ensures that no user information is retained and leaked on the device being used if the user

			decides to log out of their account
<b>U-5</b>	The system should allow users to delete their account	High	Allows users to fully be able to disassociate themselves from the application if they so desire
<b>U-6</b>	The system should check if the users are Authenticated before calling any APIs	High	Ensures that the APIs are only being called by a registered user and not maliciously from an external source
<b>U-7</b>	The system should allow users to change their password	Medium	Allows users to change their passwords as they deem necessary.
<b>U-8</b>	The system should allow users to reset their forgotten password	Medium	Allows for the human error of forgetting one's password
<b>U-9</b>	The system should allow users to set a display name	Medium	Allows customizability for users
<b>U-10</b>	The system should allow users to set/upload a display picture	Low	Allows the user's account to be customisable and personalisable

#### 4.1.2 Matching Service

<b>S/N</b>	<b>Functional Requirement</b>	<b>Priority</b>	<b>Justification</b>
<b>M-1</b>	The system should allow users to select the difficulty level of the questions they wish to attempt	High	Users with the same difficulty selection typically will have the roughly same proficiencies and goals and will be most effective and helping each other
<b>M-2</b>	The system should be able to match two waiting users with similar difficulty levels and put them in the same room	High	This allows for the two users to start the mock interview with each other
<b>M-3</b>	If there is a valid match, the system should match the users within 30s	High	Allows for a better user experience by ensuring that users do not have to wait too long to start their practice session
<b>M-4</b>	The system should inform the users that no match is available if a match cannot be found within 30 seconds	High	Notifies the user that there is currently no match to be found and reduce server load by ensuring user will not be perpetually in queue
<b>M-5</b>	The system should be able to swap the roles of the participants	High	Allows both participants of the room to experience being interviewed

<b>M-6</b>	The system should provide a means for the user to leave a room once matched	Medium	Allows the user to leave the current match and room based on their own discretion
<b>M-7</b>	The system should allow users to cancel the matching before a match has been found	Medium	Allows for a margin of error such as when a wrong difficulty is selected
<b>M-8</b>	The system should allow participants to continue the session with the same user	Medium	Users can continue to conduct mock interviews with the same match as they see fit such as when they think the other party is proficient in interviewing

#### 4.1.3 Question Service

<b>S/N</b>	<b>Functional Requirement</b>	<b>Priority</b>	<b>Justification</b>
<b>Q-1</b>	The system should be able to retrieve from the question bank and display them in the shared workroom	High	Improves user experience as users do not need to find questions themselves
<b>Q-2</b>	The system should provide sample inputs and outputs for each question	Medium	Allows for a better understanding of what the question is asking for

#### 4.1.4 Coding Service

<b>S/N</b>	<b>Functional Requirement</b>	<b>Priority</b>	<b>Justification</b>
<b>C-1</b>	The system should allow participants in the room to concurrently edit the code	High	Allows for real-time collaboration between users
<b>C-2</b>	The system should allow users to use their preferred coding language	High	Mimics real technical interviews as interviewees typically can choose their most comfortable coding language to partake in the interview
<b>C-3</b>	The system should be able to compile and run the code	High	Allows for immediate testing of the code rather than only inspecting it visually
<b>C-4</b>	The system should be able to show the output of the code	High	Allows the users to verify the output and hence correctness of the written code
<b>C-5</b>	The system should provide some syntax highlighting	Medium	Improves user experience
<b>C-6</b>	The system should allow	Low	Allows the interviewer to provide

	interviewers to leave notes for the interviewee		feedback to the interviewee which can be reviewed at a later date
<b>C-7</b>	The system should allow for autocompletion of code	Low	Improves user experience

#### 4.1.5 Room Service

<b>S/N</b>	<b>Functional Requirement</b>	<b>Priority</b>	<b>Justification</b>
<b>R-1</b>	The system should inform the other user if one user ends the interview	High	So that the user is not left alone in the room
<b>R-2</b>	The system should allow users to swap roles	High	Allow users to have a chance to try both roles

#### 4.1.6 History Service

<b>S/N</b>	<b>Functional Requirement</b>	<b>Priority</b>	<b>Justification</b>
<b>H-1</b>	The system should store and display past practice sessions of users	Medium	Allows users to review their code and feedback from interviewers at a later date

#### 4.1.7 Communication Service

<b>S/N</b>	<b>Functional Requirement</b>	<b>Priority</b>	<b>Justification</b>
<b>V-1</b>	The system should allow for voice and video chat between the two matched parties.	Medium	Mimics a real technical interview with voice/video chat being the main form of communication
<b>V-2</b>	The system should allow users to mute themselves in the voice chat	Medium	Allows users to choose what is being broadcasted to the other user
<b>V-3</b>	The system should allow users to adjust volume input and output levels	Medium	Improves user experience as they can fine tune their voice chat experience to their liking

## 4.2 Non-functional Requirements

### 4.2.1 Security

S/N	Non-functional Requirements	Priority	Justification
SEC-1	Users' passwords should be hashed and salted before storing in the DB.	High	Ensures that even if the users' database is leaked, it would not put the users' account at risk
SEC-2	Users' Password strength should be sufficiently strong (1 Uppercase, 1 Lowercase, 1 Special Character, 1 Number, at least 8 character)	High	A complex password helps to combat dictionary attacks.  A longer password helps to combat brute force attacks.
SEC-3	All communication between server and application should be done using HTTPS	High	Prevents Man-in-the-middle attacks
SEC-4	Users' inputs should be sanitised when registering.	Medium	This ensures that the data inputted into the server only contains safe content and does not result in security breaches.
SEC-5	All databases hosted on the cloud should have appropriate access control enabled (e.g. IP access control)	Medium	Prevents unauthorised access to cloud databases

### 4.2.2 Performance

S/N	Non-functional Requirements	Priority	Justification
PER-1	Once users are matched, matched users should be added into the room within 2 seconds	High	Provides better user experience as users do not have to wait too long before getting into the room
PER-2	Navigation between different pages of the application should take less than 3 secs and less than 6 secs under load.	High	Improves user experience and reduces bounce rate
PER-3	The latency between users while typing in the code space should be under 1 second	Medium	Lower latency of code updates will lead to better user experience and more relevant feedback from interviewers
PER-4	The delay for voice and video chat should be less than 3s	Medium	Lower latency of voice and video chat will lead to better user experience and more

			relevant feedback from interviewers
<b>PER-5</b>	The audio quality of the voice chat between users should be at least a bitrate of 64kbps	Medium	Better user experience as it is easier to communicate
<b>PER-6</b>	The video quality of the video chat between users should be at least 720p 30fps	Low	Allows observation of the interviewee's non verbal cues during the mock interview and better user experience

#### 4.2.3 Robustness

S/N	Non-functional Requirements	Priority	Justification
<b>ROB-1</b>	An error should be shown when running time of the code exceeds 5s	High	Prevents infinite loops in faulty code from crashing the code executor

#### 4.2.4 Portability

S/N	Non-functional Requirements	Priority	Justification
<b>POR-1</b>	The system should be able to run on all major browsers (Safari, Chrome, Firefox)	High	Accommodates most users
<b>POR-2</b>	The system should be deployed using docker	Medium	Allows application to be portable and easily deployed on systems with different OS



## 4.3 External Interface Requirements

### 4.3.1 Hardware Interfaces

This product is designed to run as a web application and should be run on a **laptop or desktop** running any mainstream OS (Windows x32/x64, OS X, Linux) which supports a web browser.

Although a web application could technically be run on a mobile device, we have decided to go for a “Desktop-First” design approach as technical interviews are most likely to be conducted on a laptop or desktop instead of a mobile device. Ideally, the application would allow for a mobile friendly interface to accommodate more users.

### 4.3.2 Communication Interfaces

Being a web application, this product communicates using **HTTPS** for secure communication between server and the web interface, as well as using **WebSockets** events for any real-time textual communication.

For video/voice chat communication between users, **WebRTC** is used as a real-time video conferencing framework which builds on the **RTP** and **RTCP** protocols.

### 4.3.3 Software Interfaces

The [tech stack breakdown](#) will give a detailed explanation of each library/software used.

This product is built on top of several libraries and databases and this section will only serve to give a brief summary of the technologies used to achieve different functions of the application.

Firstly, WebSocket events used by the browser and server as one of the main forms of communication is built upon **Socket.IO**.

The real-time voice/video chat function is enabled using **Agora**.

The ability to compile and execute code written in the mock interview is achieved using a self-hosted instance of **Judge0**.

Lastly, as there is a need to persist data such as users’ account information, users’ history and question bank, there is a need for us to use databases such as **MySQL**, **MongoDB** and **SQLite**.

## 4.4 Feature List

We have implemented all must-have features and three nice-to-have features in the application

- User service manages the authentication and maintaining information of users of PeerPrep
- Matching service is responsible for matching users together based on the difficulty they select
- Question service is responsible for storing a question bank indexed by difficulty level and retrieve a question of appropriate difficulty level to the mock interview room
- Coding service provides concurrent collaborative code editing between the matched users in the room
- Our frontend provides basic UI for user interaction
- Our frontend is reactive and user-friendly for users to easily use all functionalities of the application
- Deployment of the application on our local machine
- Communication service provides the video and voice call feature between the matched users in the room
- History service maintains records of the past interview attempts for users to review
- Room service manages and maintains status of rooms including questions chosen and reconnection of users
- Deployment of application to the (local) staging environment (Docker-based)
- API gateway allows a single point of entry for all requests to then be redirected to the respective relevant microservices

## 5. Developer Documentation

### 5.1 Tech Stack

#### 5.1.1 Frontend

##### **React**

We chose react as we have decided that we wanted to implement a Single Page Application for a smoother user experience. The application is loaded once at the start and users can navigate between pages without having to wait for them to load. Additionally, React supports many frontend libraries such as Material UI, Bootstrap, etc to build a rich user interface.

Additionally, React was the simplest to learn as we had to account for group mates who have never worked with any web development frameworks before.

React also offers us the ability to reuse every component that we build, allowing us to easily extend our project if needed.

##### **Material UI (MUI)**

MUI was chosen as it provides a consistent design and experience throughout the application. Since it follows the design principles and templates used by Google it will also provide users with a familiar style and experience when using our application.

By using the pre-designed user interface components provided by MUI, we can take the onus of designing the components off ourselves which can be both challenging due to our lack of user-testing to construct proper user-centric designs and time-consuming due to our unfamiliarity with such a task. This allows us to channel our time towards building the functionality of the application instead.

Additionally, MUI allows for themes to be easily created and applied to all user interface components throughout the application which allows for a consistent look. This along with the pre-built components that MUI provides, allows for a quicker and smoother development process as these components can be easily reused.

#### 5.1.2 Backend

##### **NodeJS**

NodeJS is perfect for real-time applications. We are employing web sockets to support real-time communication between the interviewer and interviewee, matching of participants and code synchronisation. The single-thread functionality makes it very suited for real-time communication and allows us to have low latency for our communication services.

NodeJS also has extensive support for the microservice architecture which we have decided to employ.

Additionally, it supports the creation of Single Page Applications since NodeJS allows for server-side rendering of pages before its data is sent to the client which is perfect for our use case.

## ExpressJS

As a performant web application framework for NodeJS, it allows us to easily implement it without much additional work such as providing a simple routing for requests made by clients and a middleware that is responsible for providing correct responses to clients. This also takes off some workload and worries of efficiency from us and allows us to simply focus on working on the functionalities of the project.

Allows for easy and efficient communication between frontend and backend to create RESTful APIs as it only relies on JavaScript which is vital for the microservices architecture to function.

### 5.1.3 Third-Party Libraries

This section contains some of the notable libraries that were used in the development of PeerPrep.

- **Sequelize:** Sequelize is an Object-relational mapping (ORM) library which provides an object-oriented layer between relational databases and object-oriented programming languages such as Javascript which was what PeerPrep was built upon. It helps to describe the relationship between an object and data without knowing how the data is structured. This helps to abstract away low level SQL queries by using standardised API calls to interface with the underlying database, thereby speeding up development time.
- **Agora:** Agora is a framework that provides ultra-low latency and high quality video and voice real-time communication services. It serves as the backbone for the video conferencing used for both parties of the mock interview to communicate with each other.

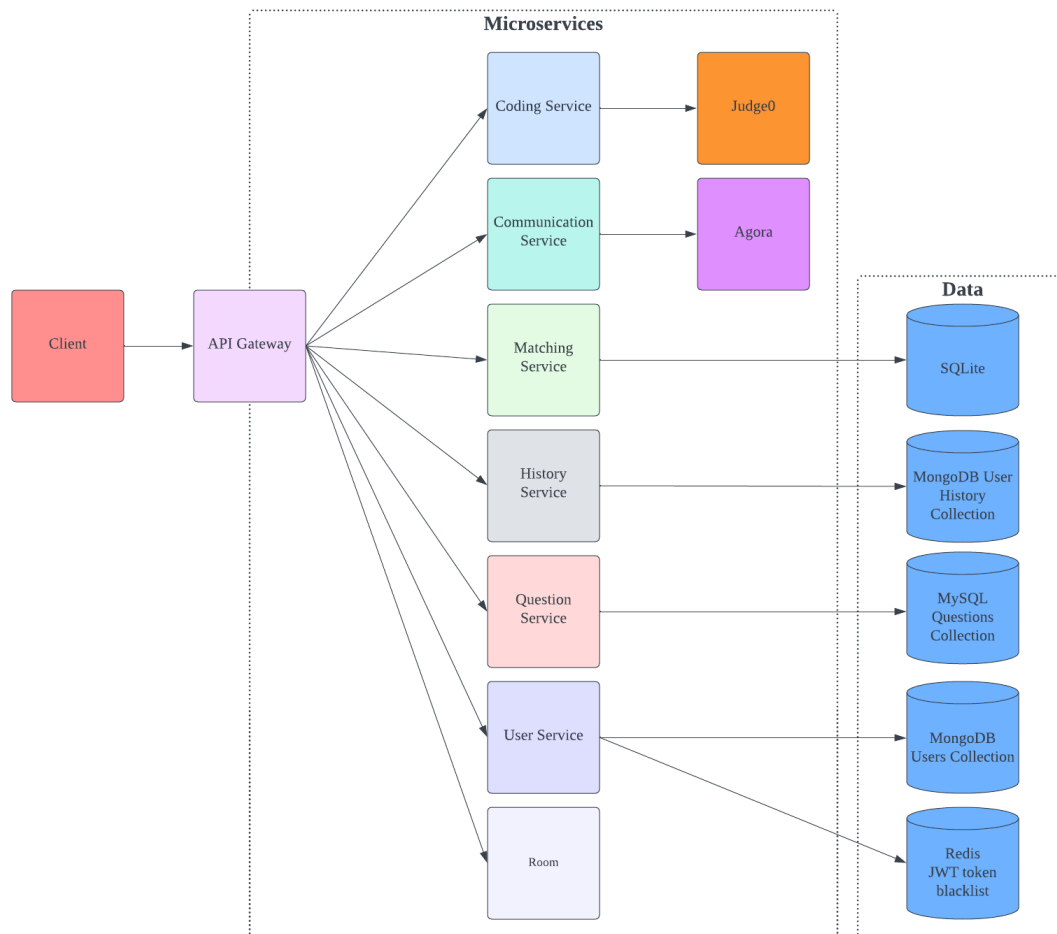
It additionally promises 99,99% uptime along with a global coverage which is in line with our availability requirement, ensuring that our application is highly available to anyone across the globe.

- **Judge0:** Judge0 is an open-source online code execution system that provides a sandboxed compilation and execution environment which isolates it from the rest of our codebase. It comes with the support for 60+ popular coding languages which allows users to run and test code in the programming language of their choice.

It comes with the added benefit of being easily scalable by increasing the number of worker nodes assigned to Judge0 if it ever comes under load. This is made possible as they provide a docker image which could easily be used in a Kubernetes Horizontal Pod Autoscaler to scale Judge0 according to workload so that more code submitted by users can be executed at once.

- **Automerger:** Automerger is a network-agnostic Conflict-Free Replicated Data Type (CRDT) which allows concurrent changes on different devices to be merged automatically without a central server, offloading the load on our server to the edge devices. It is used in the coding service since it allows the two users to collaboratively edit in the code editor concurrently without any conflicts.

## 5.2 Architecture



*Figure 5.2a: Overall Architecture Diagram*

We decided to employ the Microservice architecture. We have thoroughly discussed this and ultimately came to the conclusion that the microservice architecture is more well-suited for our use case than the monolithic architecture.

Firstly, all services can be deployed and updated independently which provides more flexibility over a monolithic architecture. A bug in one microservices would only affect that specific service and would not influence the entire application which helps isolate or contain the spread of the ill-effects of bugs. Thus, changes and experiments can be implemented with less risks.

Secondly, due to the incremental nature of the application's software development lifecycle, new use cases and features could be proposed midway through development. A microservice architecture allows for easy extensibility as new services could be easily developed in isolation of the current application. Technologies are also not restricted by decisions made at the start and can be changed as we deem fit for different services.

Thirdly, a microservice architecture allows for better scalability. As this has a very wide audience and very applicable use case, we believe that it may attract more users in the future. A microservice architecture allows for scaling of each independent component rather

than the application as a whole. This allows for scaling only where needed which is both cost and time efficient, both of which our team lacks.

Despite all this, we are also aware of the disadvantage of using a microservice architecture over a monolithic one. A monolithic architecture would be more suited to a small team such as ours as a microservice architecture is more complex to implement. Complexities such as having to deploy, test, and maintain multiple services are present. However, we recognise these drawbacks and concluded that it is beneficial for us to take on the toll of a more complex implementation and use the microservices architecture instead. The benefits of ease of development and deployment of the monolithic architecture are heavily outweighed by the advantages of scalability and decoupling of services offered by microservices due to the importance of scalability in our use case. Additionally, the disadvantages of the monolithic architecture such as the heavy coupling and rigidity to making changes are also detrimental to our use case since we are working in a team and have delegated different parts of the project to different team members. Having to work with a monolithic application would cause a lot of confusion as it would prevent us from working on each part assigned to ourselves in relative isolation of other parts due to the heavy coupling.

## 5.3 Frontend

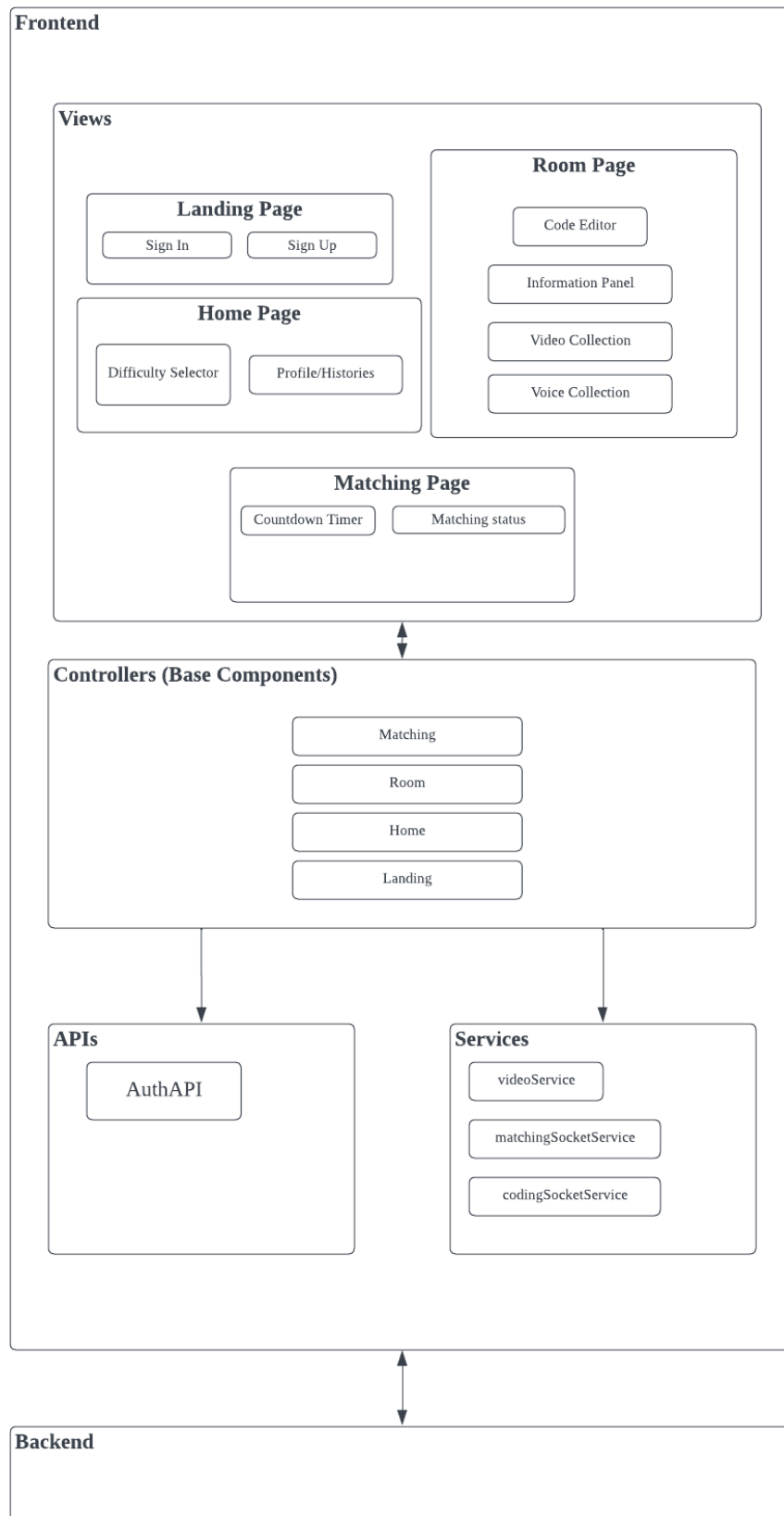


Figure 5.3a: Frontend Client Architecture Diagram

Our frontend is roughly modelled off the Model-View-Controller pattern. However, a small amount of it goes against the principles of the pattern as certain user event logic is handled in the frontend such as form submissions and button clicks.

## 5.4 Microservices

The series of subsections below serves to introduce and explain what each of our microservices does as well as to elaborate on some implementation choices we made.

### 5.4.1. User Management Service

The user management service handles all user account information as well as the authentication and authorization of the users. For all accounts, the user's email must be unique to prevent a particular user from setting up multiple accounts with the same email. And we also require unique usernames to allow for ease of querying of our user database.

When a user signs up, the user account information is sent to the user service via POST request made to the sign up API endpoint, and this information is then stored into a MongoDB Atlas database. Do note that the password entered by the user is salted and hashed before it is stored. Also, we chose MongoDB Atlas as our user service database mainly due to the security features and encryption that it provides as well as the ease of set-up and use.

Once a user has signed up for an account, they will be able to login. When a user logs in, the user service first verifies the entered account credentials by checking against the credentials stored in the database. If verified, the user service generates a JWT token using the user's user id and username and has an expiration of 7 days. This JWT token is stored in the user's cookies and will be sent back to the user service for authentication and authorization purposes. For security purposes, the cookie stored has the 'HttpOnly' and 'Secure' attributes enabled as well as the 'SameSite' attribute set to 'Strict'. By setting these attributes, it helps to mitigate cross-site scripting (XSS) and cross-site request forgery (CSRF) attacks.

For authorization within the user service, the 'authorise' middleware intercepts HTTP requests sent to selected endpoints and verifies the JWT token in the request cookies. If successfully verified, the request is then forwarded to the respective endpoint.

The API gateway also has an 'authorise' middleware that utilises the 'user/auth' route of the user service. The 'authorise' middleware of the API gateway intercepts works the same as that of the user service but instead intercepts and forwards HTTP requests sent to selected microservices.

The user service also supports various other endpoints that are used to manage the user's account information. These endpoints are listed with diagrams in [Section 5.6.1](#)



## 5.4.2 Question Service

The question service provides questions from our database for the users in the room. The question selected will be aligned with the difficulty chosen from both participants. When a match is found, the difficulty of the questions required is sent to the question service and a random question is pulled from its database to be provided to the room. We employed the use of an SQL database for this, specifically MySQL.

We decided to use an SQL database as the questions have a predefined structure which is more suited for SQL databases which have a schema and follows a table-like structure. Additionally, there are only a limited and relatively small number of questions which do not grow much over time. The question database is only called once per room to get a random question and is unlikely to see a high load on the Database attached. Given the above considerations, we have decided not to use a NoSQL database since there is no need for the high horizontal scalability, and keeping all questions in one database would allow the random questions to be chosen somewhat uniformly which lessen the chance of encountering the same question during practice.

We settled for MySQL as our chosen database. Initially we were deciding between PostgreSQL and MySQL. After research, we found out that MySQL is simpler to use, faster and more reliable. It is more suited for simple operations such as reading and writing which is all the question service needs. The question service does not require the implementations of complex data types that PostgreSQL offers. We do concede that MySQL is less secure than PostgreSQL. Security is a major factor but it is arguable that high amounts of security is not needed for the question service as it does not contain nor work with any confidential information such as passwords or users' personal information.

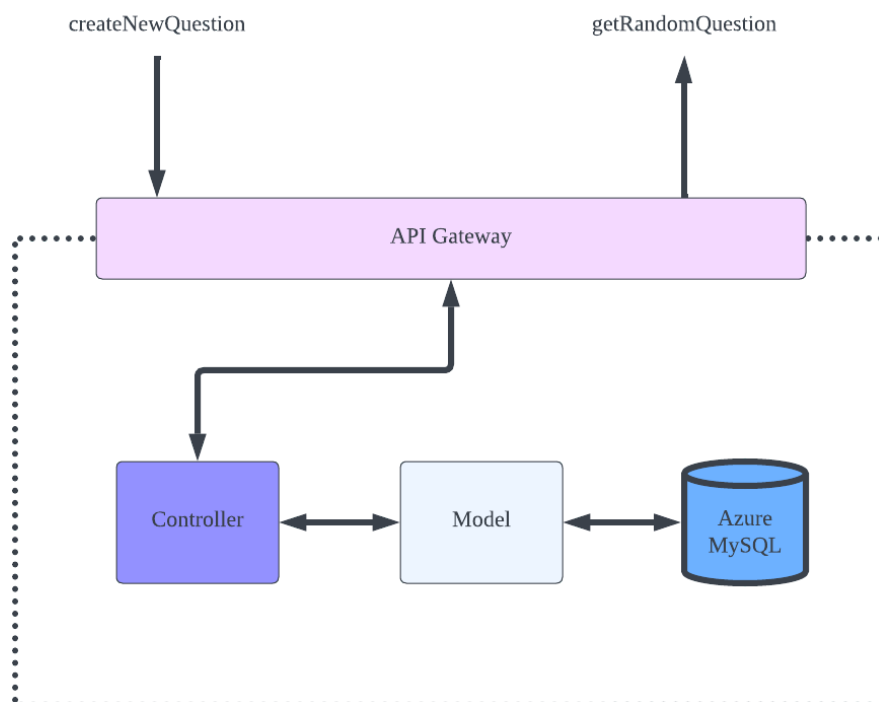
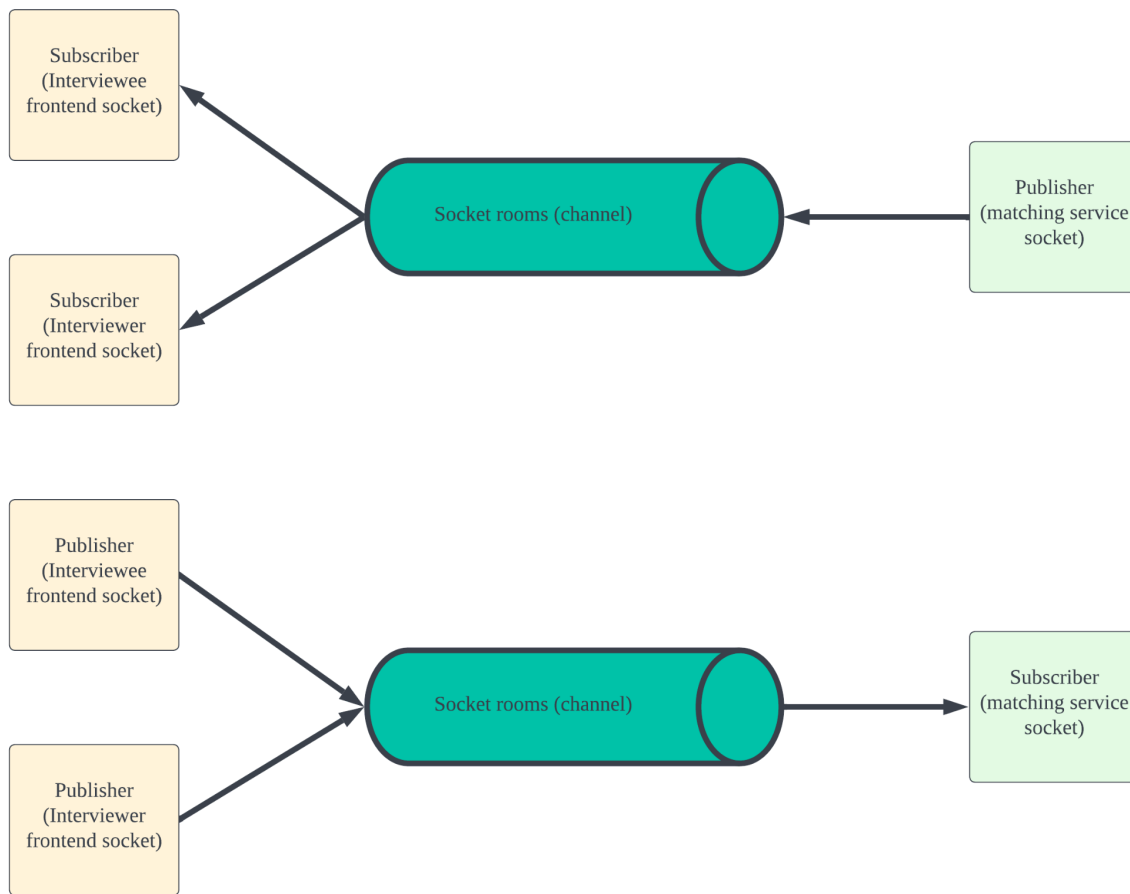


Figure 5.4.2a: Question Service Architecture Diagram

### 5.4.3 Matching Service



*Figure 5.4.3a: Matching Service Pub-Sub diagram*

The matching microservice helps to match users of the application together. Users will only be matched together if they have selected the same difficulty of questions. As the matching is done in real-time, no data is persisted and instead runtime structures are used to maintain which users are currently in queue to be matched.

We have also set a time-out for the matching. After thirty seconds in the queue, players will be forcefully removed from the queue. They will be given the option to rejoin the matching queue or return to the dashboard. This was done to indicate to the users that there is currently a lack of matches for the current difficulty. Additionally, this also takes some workload off our servers as the user is not perpetually using the matching services despite there being no one to match with.

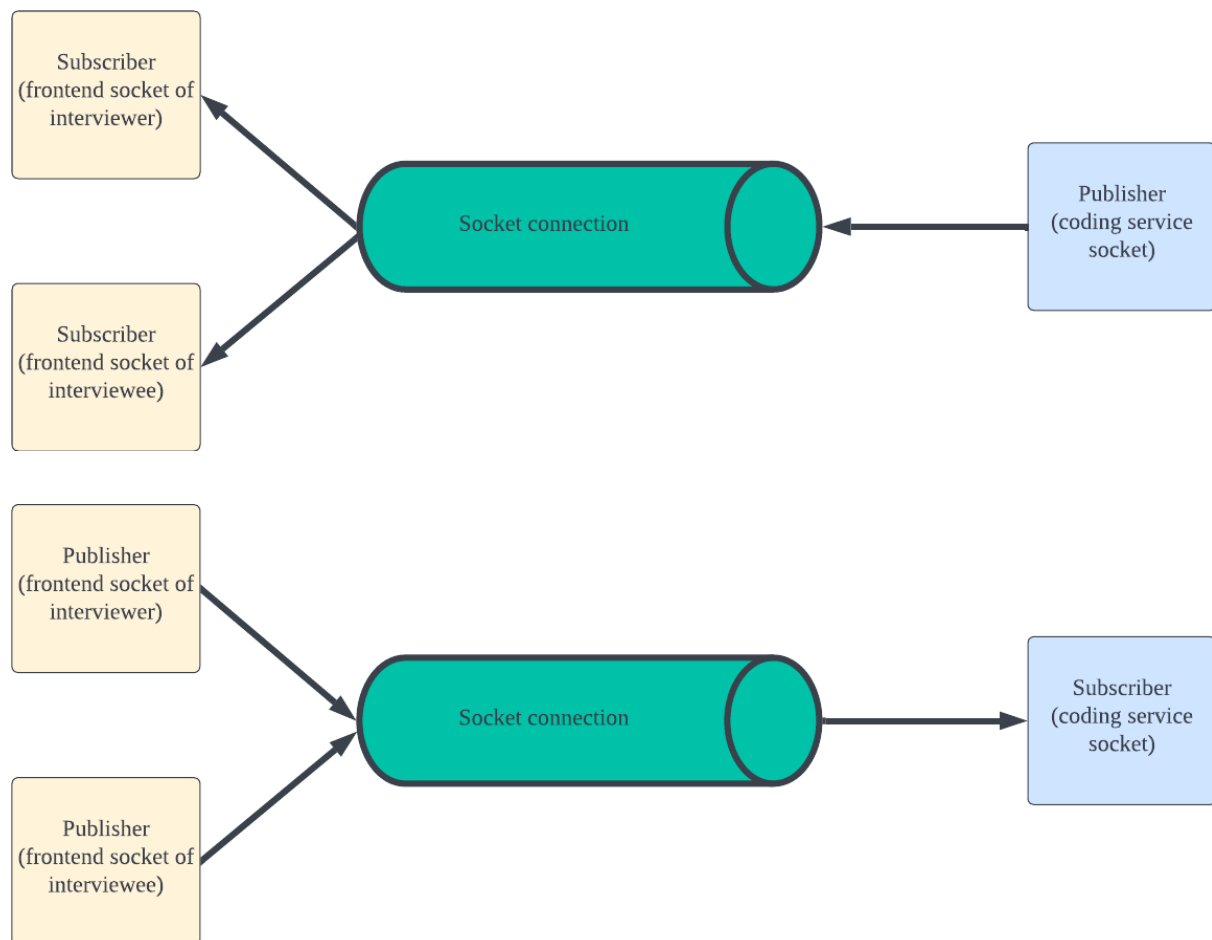
The matching microservice functions as follows:

- Client selects a difficulty of question to tackle and emits an event to a socket and will be added to a pool of available matches
- When two clients choose the same difficulty, they will then be matched together

- The matched clients' will be removed from the pool of available matches and put into a room together
- A thirty second timeout is set on the match event emitted when clients connect
- If the client cancels the match or match event times out, clients will emit a match fail event which will remove them from the pool of available matches

#### 5.4.4 Coding Service

The coding service helps users to maintain their code on screen as well as execute their code when necessary. This service addresses two concerns: code collaboration and execution.



*Figure 5.4.4a: Coding Service Pub-Sub diagram*

**Code collaboration:** This allows for a conflict-free resolution of edits made to the code by both users. It enables for real-time collaboration of code between both parties in the room. This is enabled through the use of socket communications, where any changes made in the code editor are packaged into the Conflict-free Replicated Data Types (CRDT), sent through the socket connection, and any conflicts are resolved at the other end.

**Code execution:** This executes the written code. It packages the code up and sends it to external code executors that are spawned on-demand. The result is obtained from these

executors, be it compilation or runtime errors or the result of the code, and displayed on screen for the users to see.

### 5.4.5 Room Service

The room service is responsible for managing the users' roles during the interview. It coordinates the events that may occur in the room, specifically the swapping of interviewer/interviewee roles and disconnection of client, etc.

The room service accepts both HTTP requests from other microservices and the frontend. The room service follows a publisher-subscriber model. Both parties will be subscribed to each other's changes such as request to swap roles and exiting the room. Upon the request to swap roles or exit the room, an event will be published to inform the other party. The other party can then be notified of the exit room event and will not have to stay in an empty room.

Alternatively, they will also be notified of the request to swap roles and may accept or decline the request. Their decision will also be published to the other party. The room service will also handle the actual role swap.

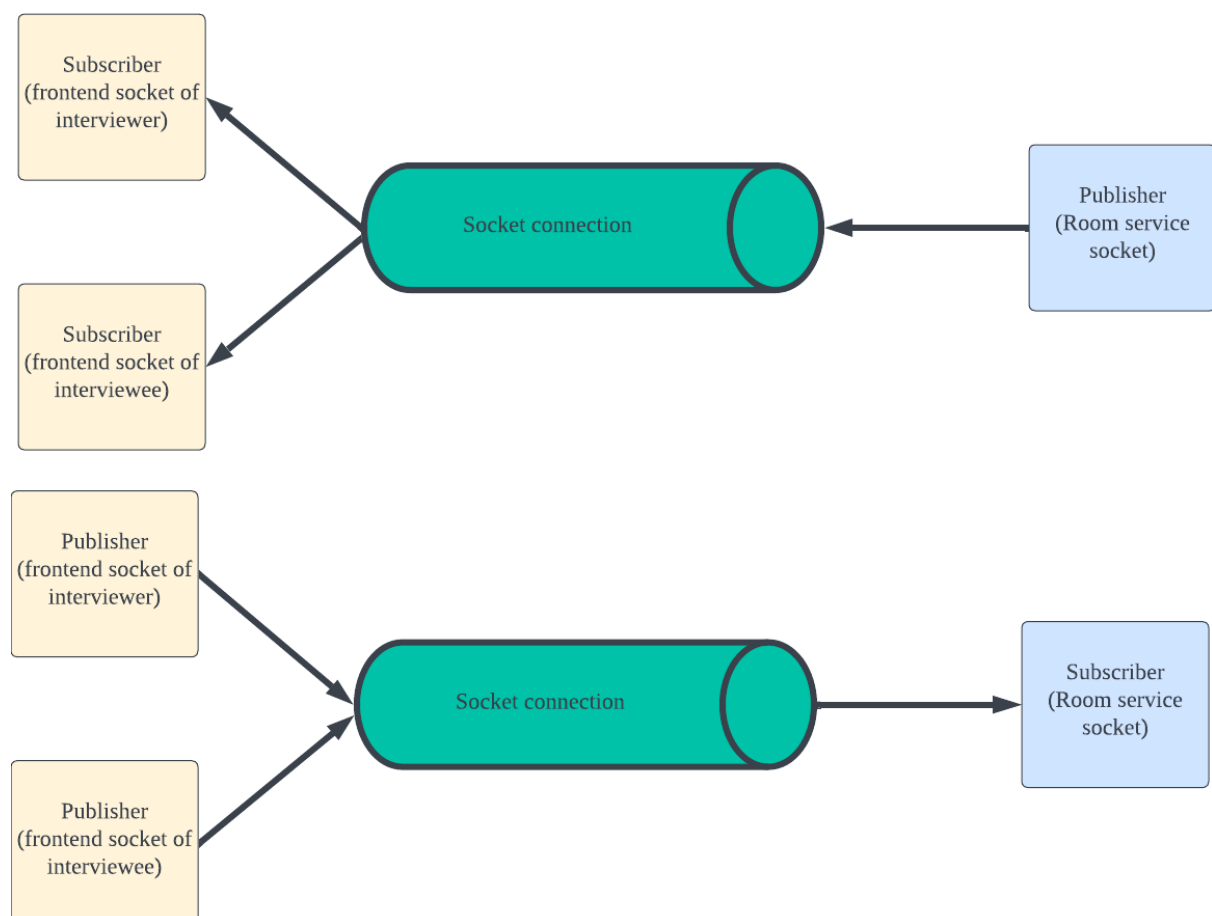


Figure 5.4.5a: Room Service Pub-Sub diagram

### 5.4.6 History Service

The history service stores the data from the mock interviews. All past data from the interviews will be stored in this microservice, allowing users to retrieve and review their interview records.

Process	
<b>id</b>	<b>string</b>
user	string
title	string
code	string
interviewer Notes	string
personal Notes	string
question	string
difficulty	string
interviewer	string
timestamp	DateTime

*Figure 5.4.6a: History Service DB Schema*

### 5.4.7 Communication Service

The communication service handles user authentication when a user joins the video call channel. It is an authentication microservice that sits between our third party video conferencing provider, Agora and our frontend application.

When the user joins the call, the frontend makes an API call to the communication service to authenticate the user. Subsequently, the communication service generates a unique token specific to the current session and returns it to the frontend. The frontend then connects to the Agora Server using the token, joining the video call channel.

The sequence diagram below shows the interactions between the user and the communication service.

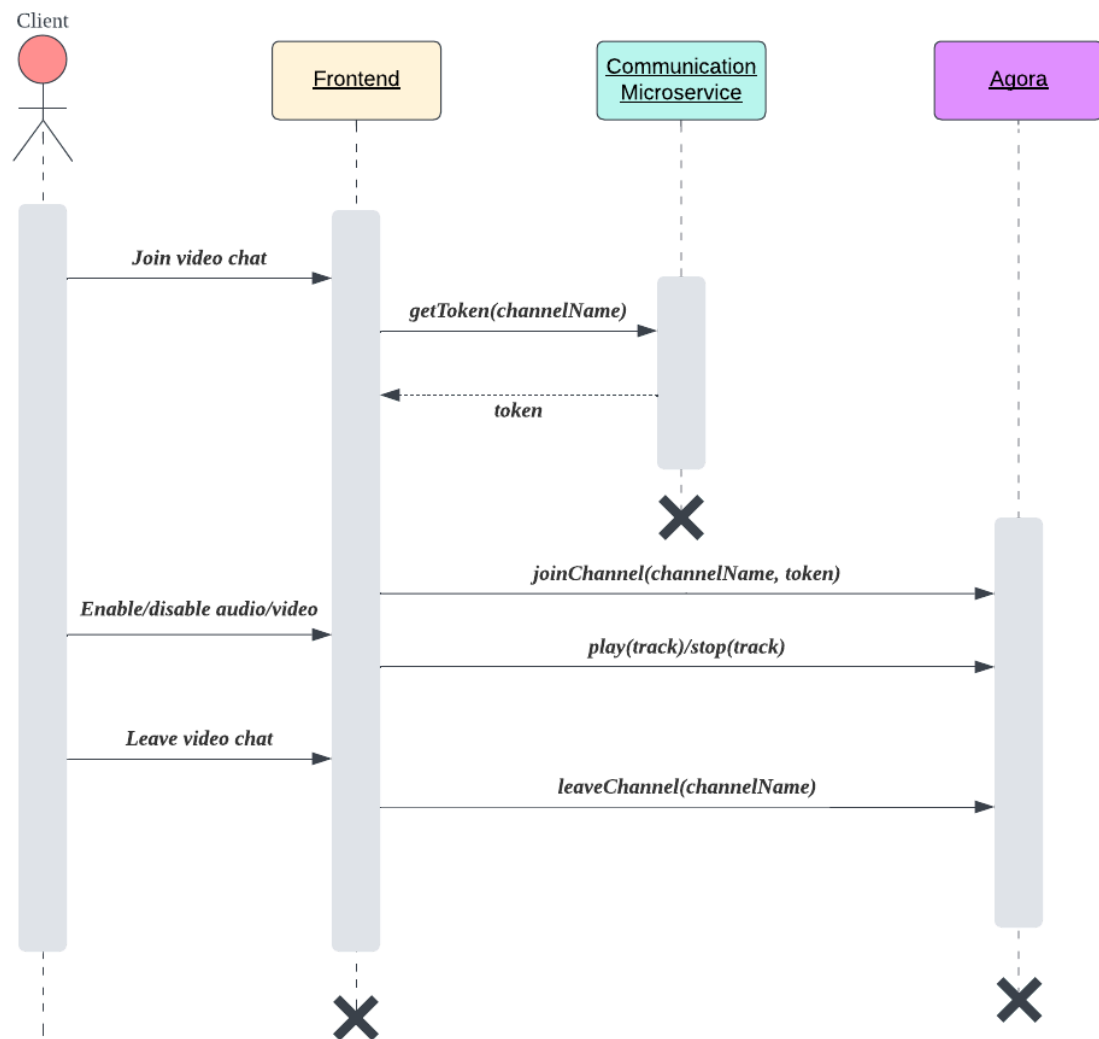


Figure 5.3.7a: Sequence diagram for video chat

## 5.4.8 Microservices Intercommunication

The diagram below briefly summarises the intercommunication between the various microservices.

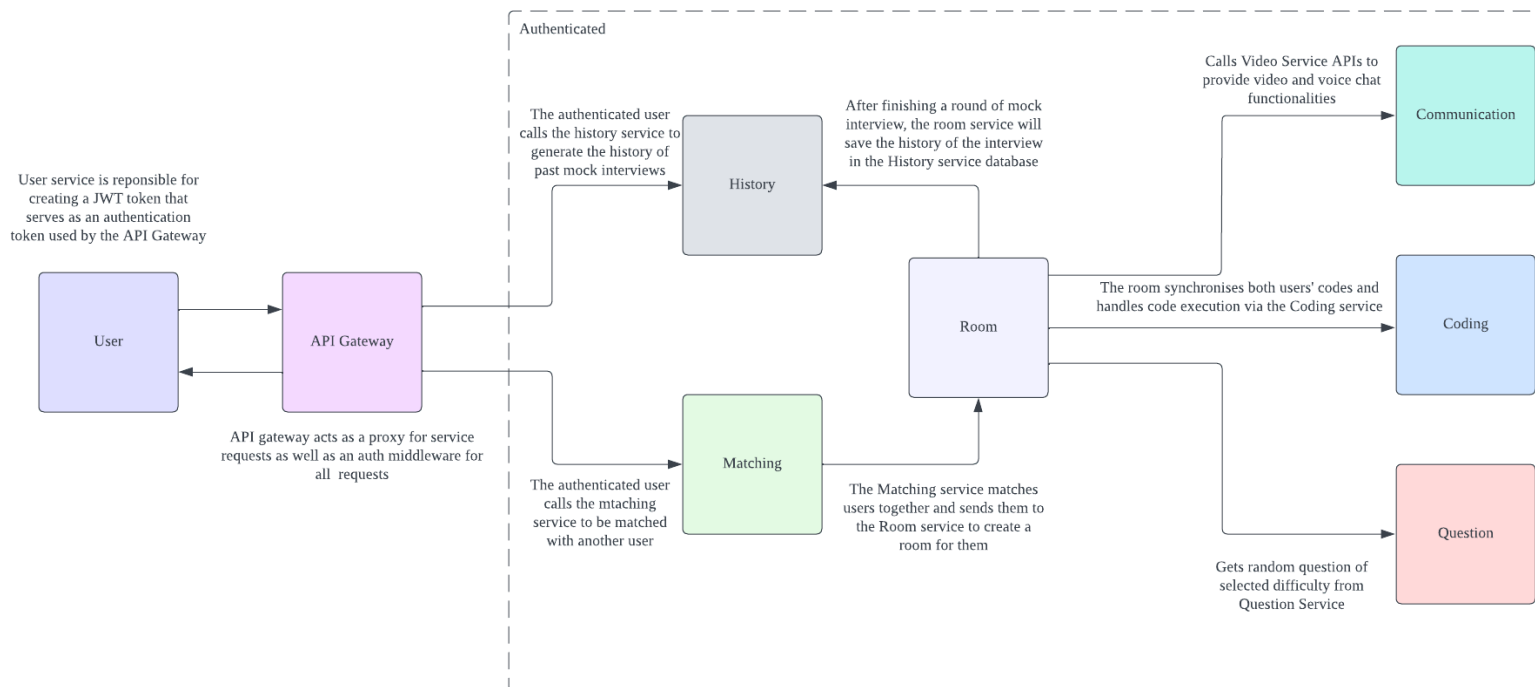


Figure 5.4.8a: Summary of Microservices Intercommunication

## 5.5 Design Patterns

### 5.5.1 MVC Pattern

We chose to follow the MVC pattern for the entirety of our application to separate the software's business logic from the presentation layer. It follows the "separation of concerns" design principle which improves maintenance of the codebase. Since PeerPrep was designed to be a web application, it naturally fell under the MVC pattern due to the separation of frontend from backend.

Each of our backend services implements a controller which serves as an intermediary between the presentation layer (frontend) and the logic layer (backend). It is responsible for processing the business logic and incoming requests, manipulating the data through the model layer, and then updates the view with the output. This output is sent to the frontend main in the form of synchronous HTTP response or asynchronously through Axios responses. It is important to note that the controller does not process the data in any way, merely telling the model layer what to do with the data based on requests from the frontend.

Most of our backend services also implement a model layer with a mix of persistence and ephemeral storage options, each suited to their respective functions. They are responsible for manipulating, processing and transforming the data which will then be propagated back to the controller and finally frontend for the presentation layer to be updated.

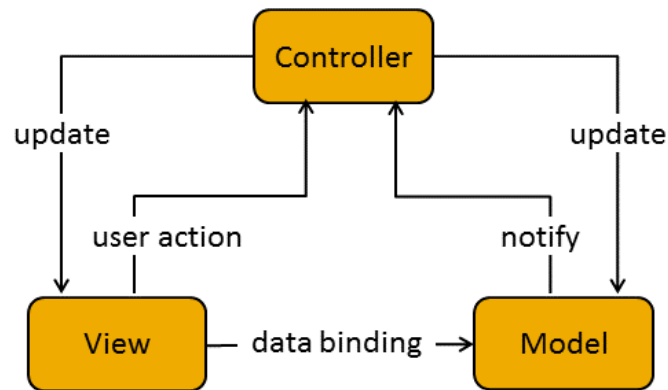


Figure 5.5.1a: MVC pattern

### 5.5.2 Pub-Sub Pattern

We have extensively employed the use of websockets in our application, especially so for our coding and matching microservices. They utilise websockets to broadcast and receive real time updates to and from the frontend, as well as to other clients. For instance, our coding microservice publishes changes from one client's coding space to its subscribers which is the frontend of the other participant in the room, allowing for real time updates of the code being written.

Our matching service also utilises the Pub-Sub pattern. As mentioned in the explanation of our [matching service](#), users who request to be matched will publish a match find event. Upon successful match, the matching service will publish a match success event to the frontend socket clients.

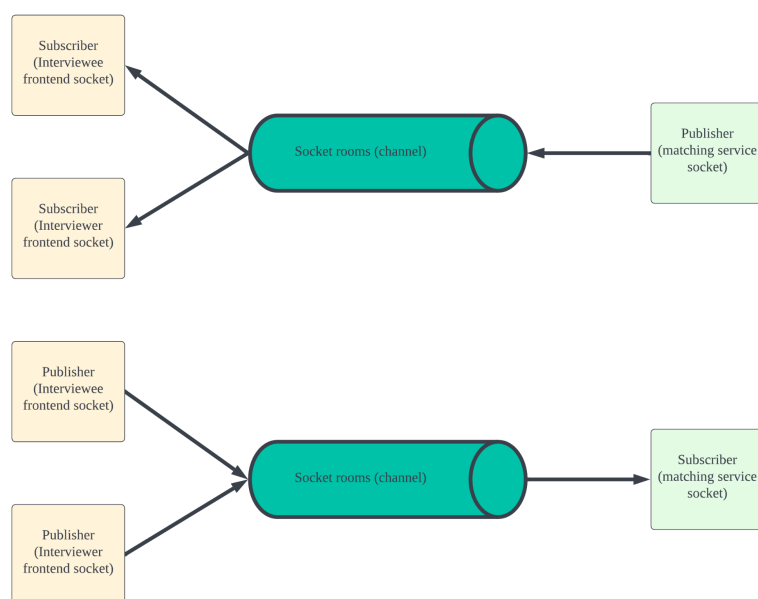
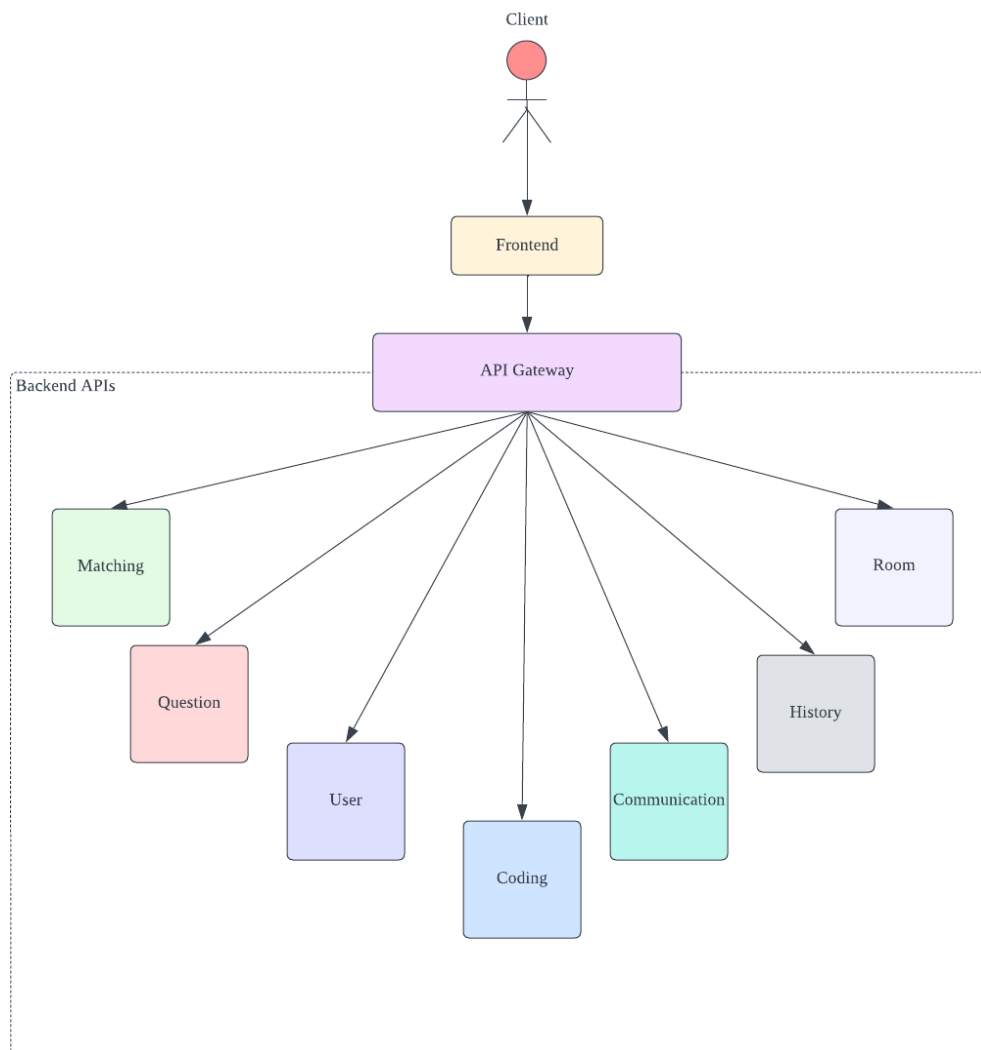


Figure 5.2.2a: Pub-Sub pattern



### 5.5.3 Facade Pattern



*Figure 5.5.3a: API Gateway*

We utilised the Facade Pattern to implement the API gateway that provides a single entry point to the subsystems for all clients.

The facade pattern is particularly suited for our application which employs the [Microservices Architecture](#) as our services provide fine-grained APIs, which means that clients will need to interact with the services in different ways. Thus, the existence of the API gateway service insulates the clients from the internal working of the microservices by providing a single entry point for the clients. In short, the facade pattern makes clients loosely coupled with the microservices and we can make changes to the microservices without affecting the client code.

The API gateway handles requests by proxying/routing them to the appropriate service. In particular, the API gateway utilises the User Management service to perform authentication on all incoming requests. It acts very much like a middleware – it intercepts HTTP requests sent to the other microservices, validates the JWT token and redirects the request to the appropriate microservice.

### 5.5.4 Repository/DAO Pattern

The repository pattern is used in most of our microservices to provide an abstraction over data storage, allowing us to decouple our model from the data layer. It hides the lower level complexities of the database. The implementation of the repository pattern is also an extension of the Data Access Object (DAO) which aims to isolate the business logic from the persistence logic.

This use of this pattern provides greater abstractions of the data persistence layer which leads to improved scalability.

### 5.5.5 Data Transfer Object (DTO)

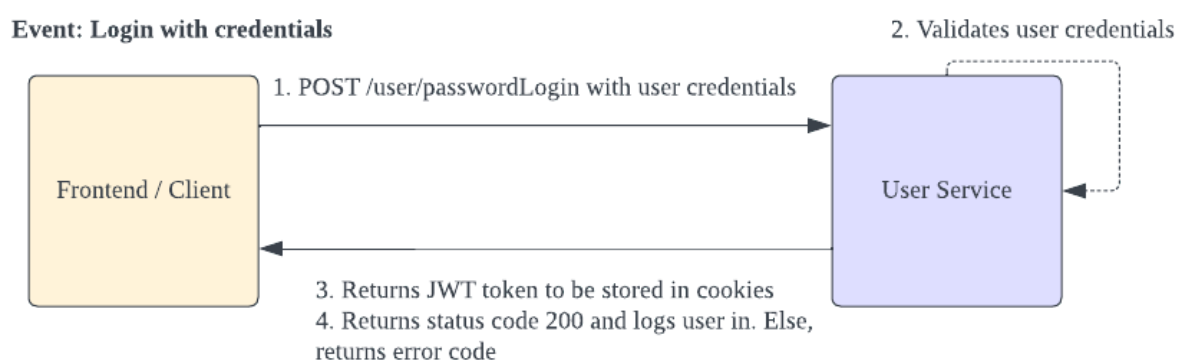
We employed the concept of DTOs. During the querying of the user history, all user history is queried for at the same time, batched up and stored locally. To view full details of a history, the data is queried from locally rather than making additional API calls to the history service. This reduces the number of API calls to the question service and thus follows the general structure of a DTO.

## 5.6 API Calls

PeerPrep adopts a microservice architecture, hence, the application requires inter-service communication between the microservices and also communication between the frontend and the backend services.

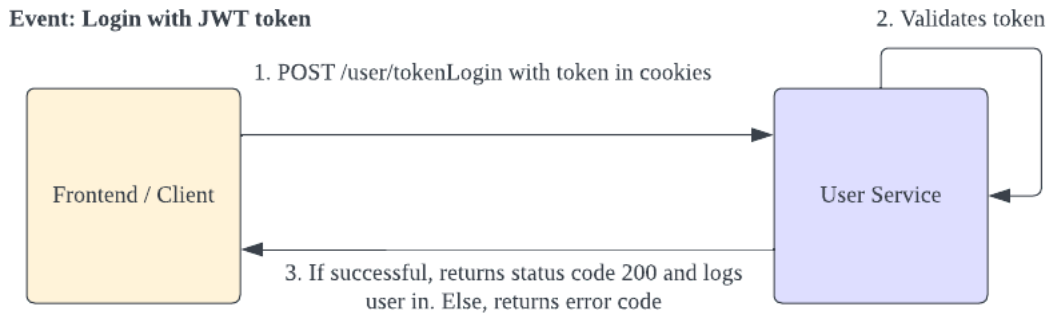
The application mainly leverages the HTTP request/response communication and Pub-sub messaging based on event-driven communication to facilitate communication between the microservices.

### 5.6.1 Auth Events



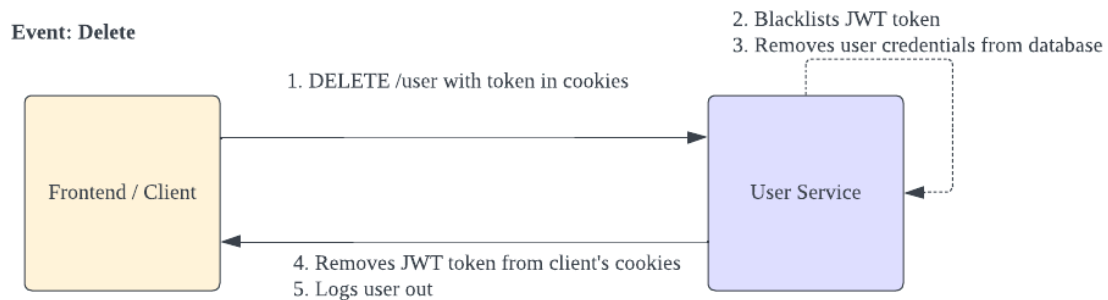
*Figure 5.6.1a: Login events (with credentials)*

When the user logs in with their user credentials, an API Post request is made to the user service. The user service validates the entered user credentials. If valid, it will send a response with the JWT token which is stored in the client's browser along with a success response that allows the client to log in. The client stores the JWT token in their browser and logs in. Else, an error code is sent as response instead and the client is not logged in.



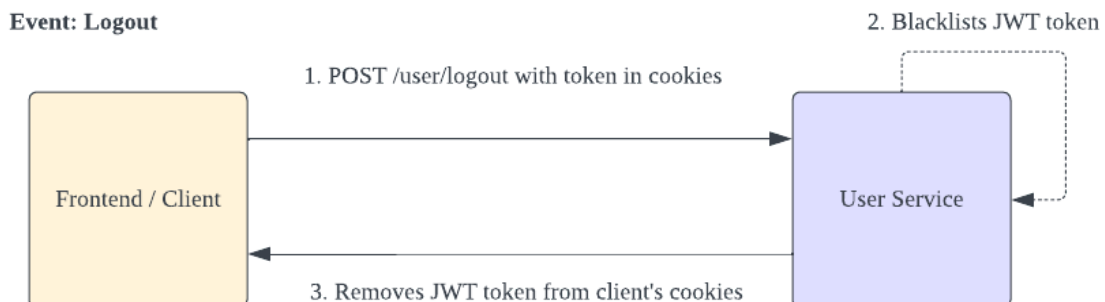
*Figure 5.6.1b: Login events (using JWT)*

In this scenario, the user would already have their JWT stored as a cookie in their browser. When the webpage is rendered, an API Post request will be automatically sent to the user service to request a login with their JWT. The user service validates this JWT and if valid returns a status 200 code and logs the user in. Else, an error code is sent as a response instead.



*Figure 5.6.1c: Delete account events*

The delete account scenario involves an API Delete request to be sent with their JWT in the cookies. The user service blacklists this JWT and removes the user credentials from its database. A response is then sent which removes the JWT from the client's cookies and logs the user out.



*Figure 5.6.1d: Logout events*

For logout, an API Post request is sent to the user service with their JWT in the request cookies. The user service blacklists the JWT and returns a response which removes the JWT from the client's cookies.

## 5.6.2 Dashboard Events

For user sign up and sign in, the frontend client communicates with the User Service to sign up or log in using his email address and username.

As mentioned in [Section 5.3.1](#), the User service will generate and return a JWT token to the client if the login credentials entered are valid.

### Event: Load User History

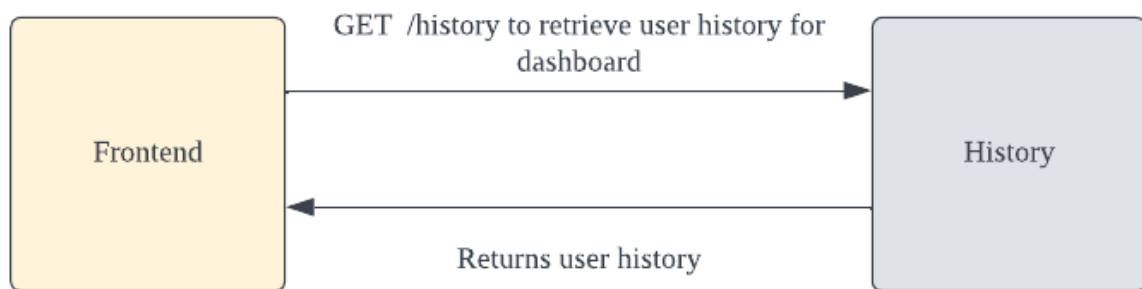


Figure 5.6.2a Get history records API call

After successful sign in, the frontend client sends a request to the History Service to fetch data for the practice history on our Dashboard display.

### Event: Loading Profile

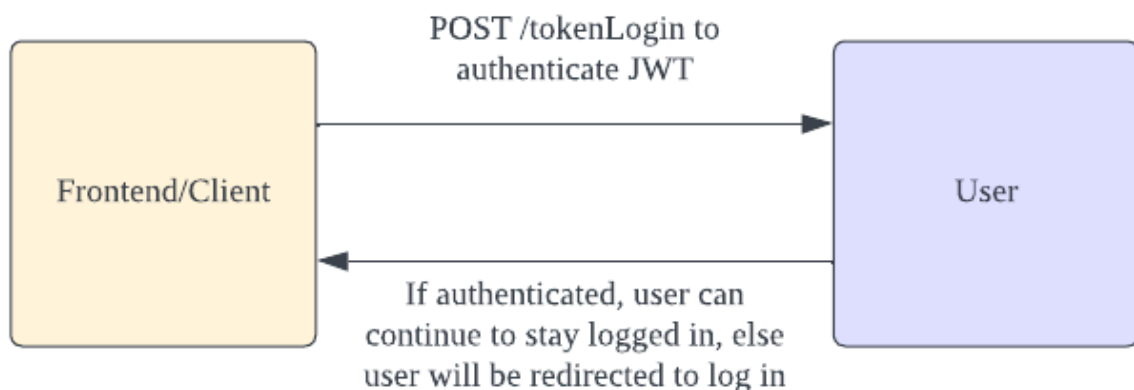


Figure 5.6.2b Authentication check API call

On each render of the dashboard, the frontend client sends a request to the User service to authenticate the JWT of the current user. If authenticated, user information is fetched to be sent back with the response to populate the dashboard's profile page with and the user can remain to use the application. Else, the user will be redirected back to the log in page.

### 5.6.3 Matching Events

The matching stage starts when the user clicks on the 'Practice' button in the dashboard. The user then be redirected to the matching page where the matching process begins. The diagram below shows the API calls during a successful matching process. At the end of the process, the user will be redirected to the coding page.

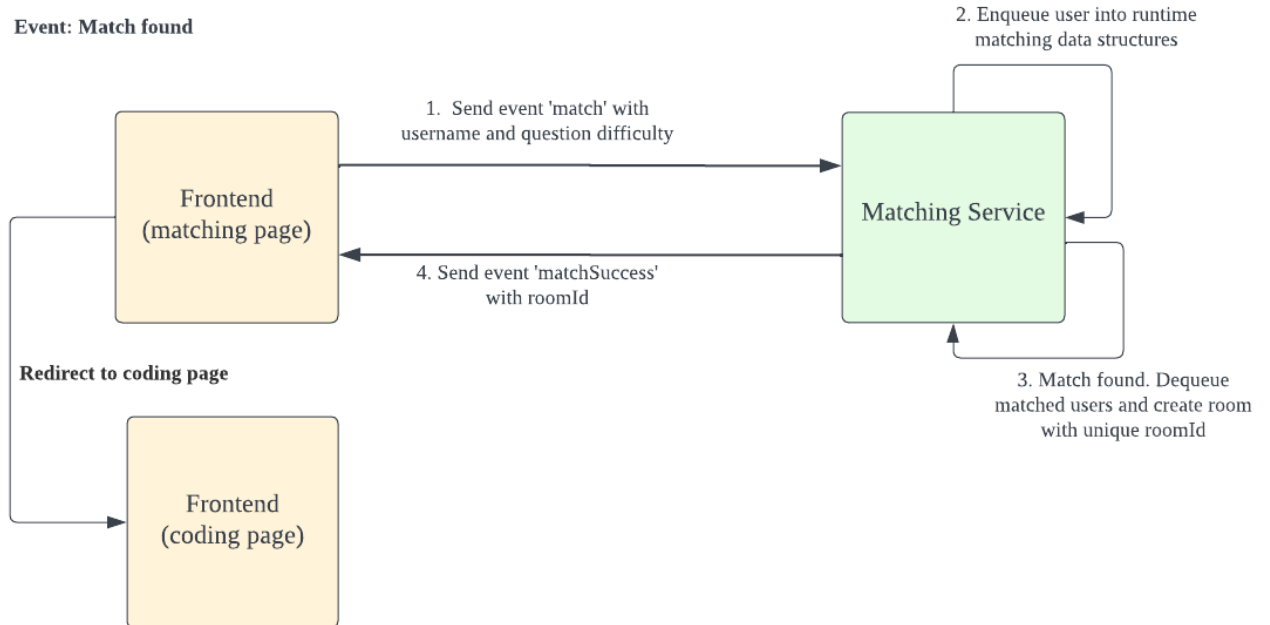
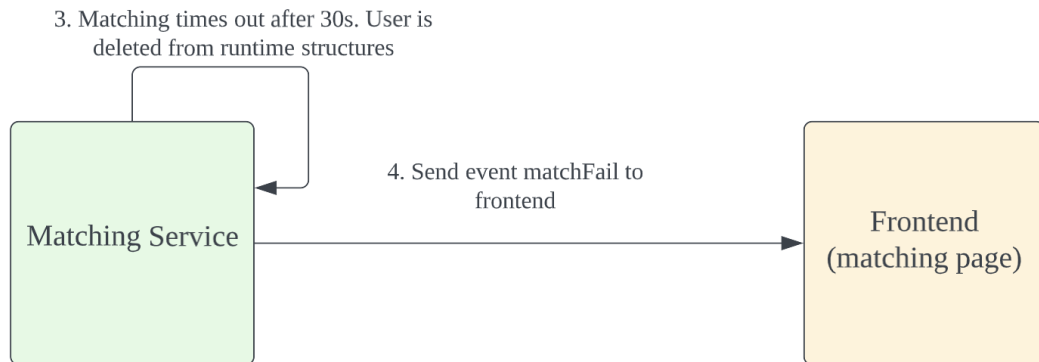


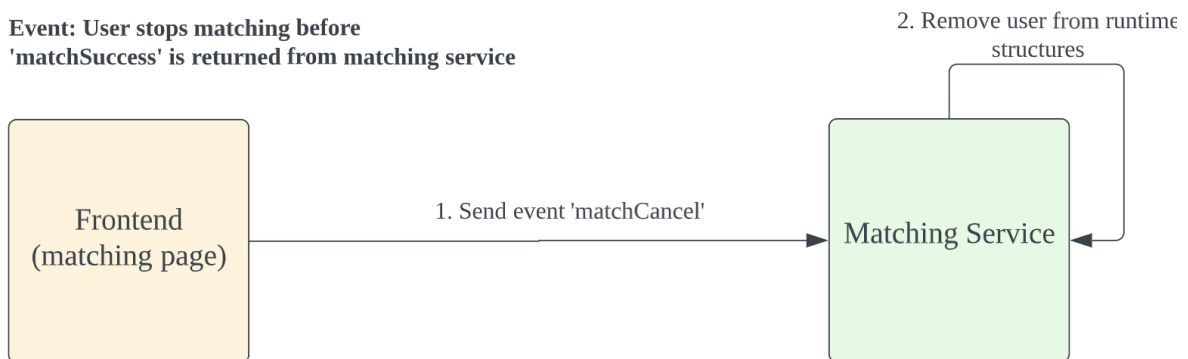
Figure 5.6.3a: Successful matching process

The following two diagrams illustrate the potential events of failure in the matching process. Termination of the matching process can be initiated by the user on the frontend, or simply due to the matching service being unable to find a match, leading to the 30 seconds time-out.

**Event: Cannot find match (continuing from step 2 in Figure 5.5.3a)**



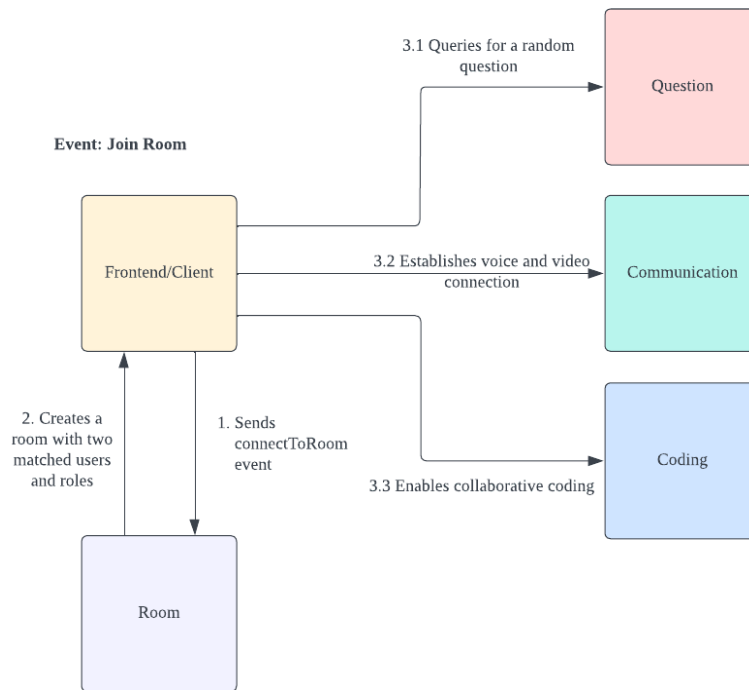
**Event: User stops matching before 'matchSuccess' is returned from matching service**



*Figure 5.6.3b: Two possible cases of failure during matching process*

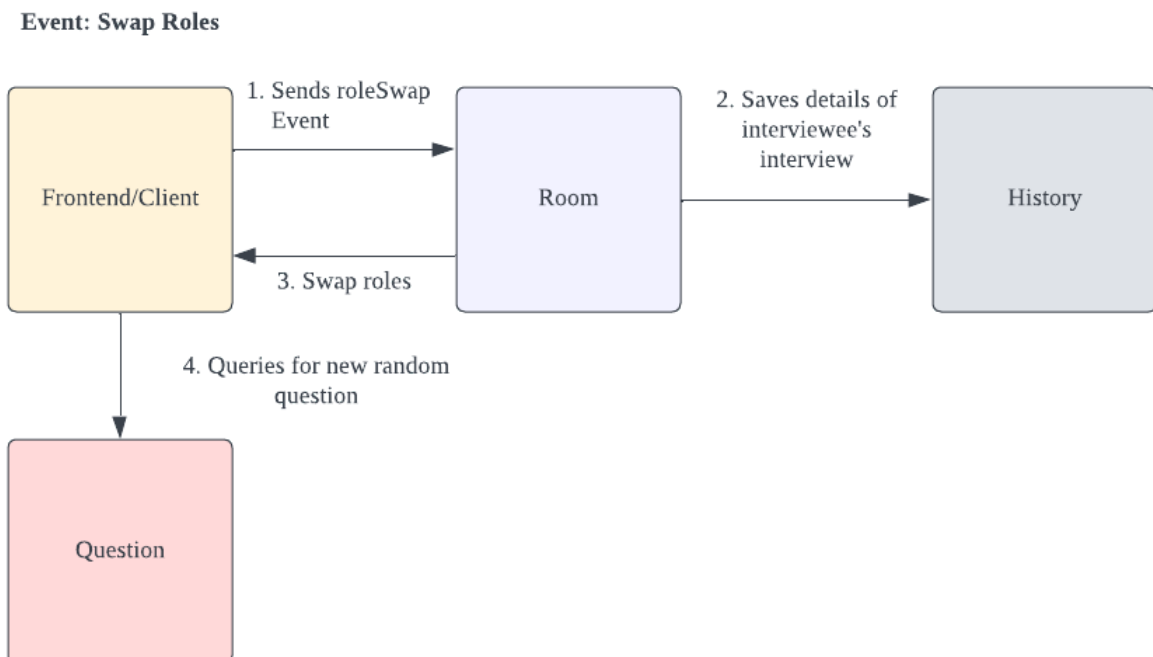
## 5.6.4 Room Events

Upon a successful matching event as in Figure 5.5.3a, the client will send a `connectToRoom` event to the Coding service, together with the `roomId` that was generated in the Matching service. The interviewer in the room will then simultaneously query the question service for a random question, may choose to connect with the communication service and will use the coding service to ensure collaboration in the code.



*Figure 5.6.4a: Join room event*

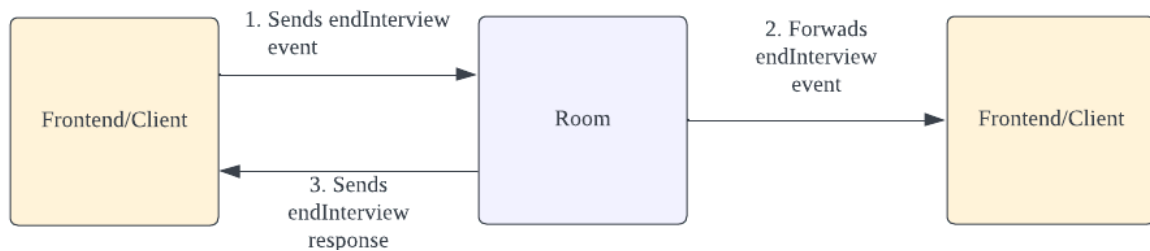
One user may request to swap roles that will inform the other user of this decision. The other user may then accept the request. The details of the interviewee's interview, including the question and interviewer's notes will then be saved in the History service for review in the future and the room service will swap the roles of the users.



*Figure 5.6.4b Swap role event*

When one user leaves the room, an endInterview event is sent to the other user to inform them of it. This is so the other user will not be left alone in a room without any feedback that the other user has left. The user may then choose to leave or remain in the room.

**Event: End Interview**

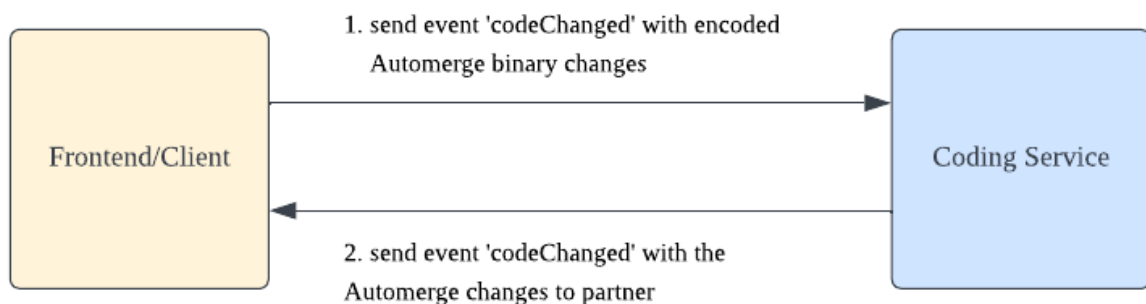


*Figure 5.6.4c: End Interview event*

### 5.6.5 Coding Events

During the interview, the interviewer and interviewee share a real-time collaborative code editor which updates the code content and programming language chosen. The coding area provides very basic syntax highlighting the programming language chosen.

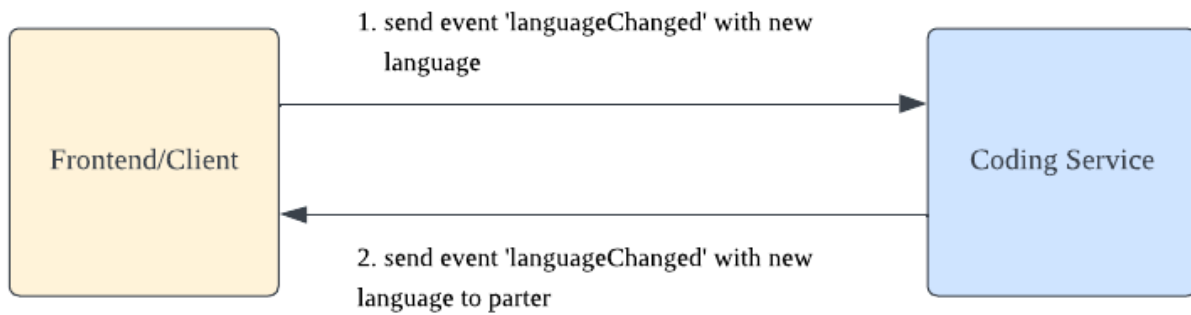
**Event: Code Updated**



*Figure 5.6.4a: Code Update event*



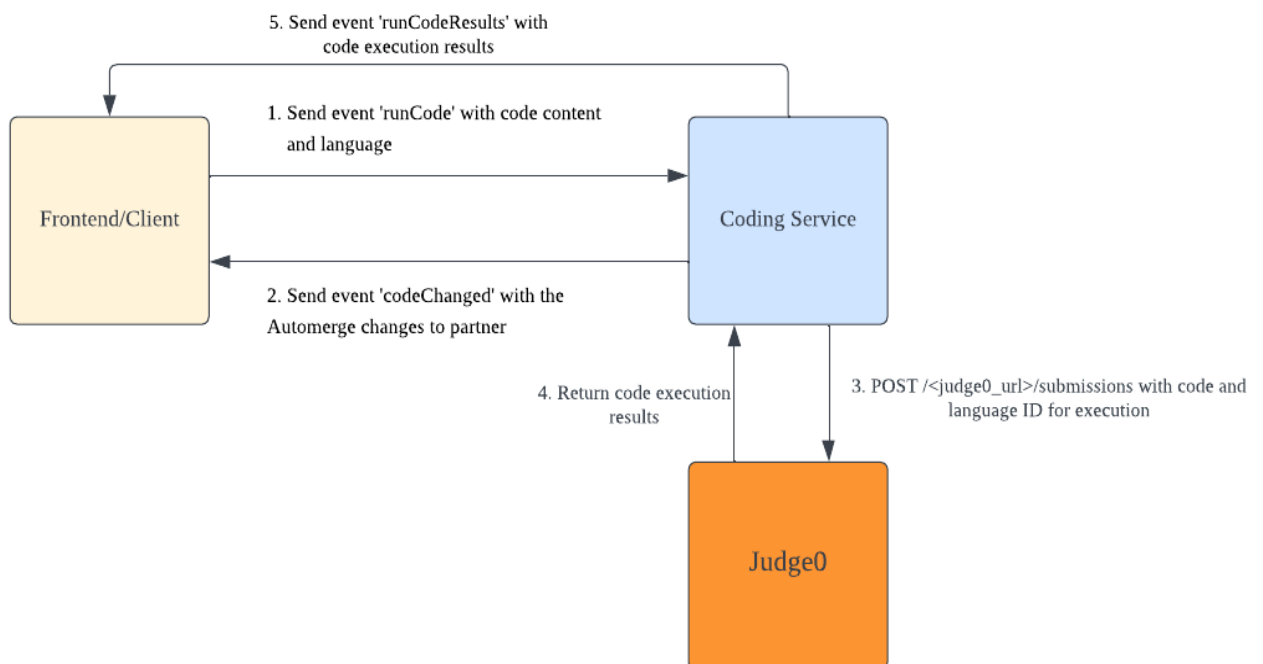
### Event: Language Changed



*Figure 5.6.5b: Language change event*

Clients are able to run their code by an event 'runCode' to the Coding Service, which pings the third-party library Judge0 for code execution output. Both clients (interviewee and interviewer) will receive the code execution output upon completion of the execution

### Event: Run Code



*Figure 5.6.4c: Run code event*

## 5.7 Other Design Considerations

### 5.7.1 Communication Service

We have decided to exclude the functionality of an instant message service in our application, opting instead to restrict our users to using their **webcams and microphones** to communicate verbally. This was a conscious decision made to actively replicate a real technical interview. As this application strives to be a simulation of actual technical interviews, it is crucial that we adhere to the communication standards set by them which is to communicate verbally, and most of the time with webcams on, rather than simply typing as such interviews not only assess one's coding proficiency but also their communication skills as well.

However, we understand that some participants will not be comfortable with turning on their webcams for a mock interview. As such, we have provided the option to turn it off. Microphones will also be given the option to mute themselves.

We chose Agora as our third party video conferencing service provider as it uses a token-exchange based approach to control access to audio and video channels, providing secure authentication.

### 5.7.2 Email and password-based authentication

We have decided to use email and password for authentication. It is arguable that using some form of external authentication such as Github would be more secure and reliable as we are employing the technology of an established company rather than making it ourselves. It would also provide more accountability as it would be directly tied to their Github account which can be seen as a repertoire of one's work.

However, the availability of our application is our priority, and the integration of a third-party authentication will make our system vulnerable to any failures in the third party, which will render our application inoperational as authentication is fundamental to our application. Additionally, the integration of third party services such as Github is relatively simple, thus, we decided to focus on building our in-house authentication service to maintain availability of the system.

## 6. DevOps

### 6.1. Sprints

Our team has mainly adopted the Agile software development methodology. We have broken up our development timeline into 3 major sprints, with each sprint lasting about 3 weeks. At the start of each sprint, we conducted thorough sprint planning to concretise the workflow of that sprint and came out with a sprint backlog that contained all actionables in a shared document. During each sprint, a weekly sprint meeting would be conducted for all developers to voice out what they have completed for the week, any problem they faced during the development process and what they are going to complete in the following week. At the final sprint meeting of each sprint cycle, we would dedicate time to integrate all new features developed by the team, run unit and regression tests, and ensure that we have a working product with the new features implemented.



Figure 6.1a Sprint Cycles of Agile Methodology

	<b>Sprint 1 (Week 4 - Recess)</b>	<b>Sprint 2 (Week 7- 10)</b>	<b>Sprint 3 (Week 10 - 13)</b>
<b>Nicholas</b>	Design frontend  Implement skeleton of frontend	Create all relevant screens and components  Implement private routing so that only authenticated users may access these routes  Integrate user service, matching service and code execution with frontend	Improve user interface of application  Integrate history and question services with frontend  Create profile page and integrated it with profile picture from user service for further customisation of account
<b>Xing Wei</b>	Implemented User	Set up Docker	Implemented the

	service and authentication	compose to deploy the application locally  Integrated code execution with the build within Docker	Room service  Implemented an API gateway
<b>Yi Guan</b>	Set up SQLite Database for Matching Service  Implement backend Matching Service	Set up Docker Compose to deploy application locally  Implement Question database and backend API	Implement History Service  Set up linting CI
<b>Sin Yee</b>	Set up MongoDB Database for User service  Implement hashing and salting of passwords	Create matching page and logic for integration with matching service	Implement video call features on frontend and set up Agora token server for communication service

## 6.2 CI/CD

Ensures that our application still runs smoothly after each change via regression testing. As manual regression testing carries a very heavy workload and is rigorous, we have employed the use of continuous integration via Github Actions.

We decided on Github Actions as it is already built into Github which is our chosen form of version control and requires no additional setup such as with Jenkins.

Continuous Integration is conducted upon every push and pull request to ensure that regression testing is properly conducted and any error can be detected early and rectified.

## 6.3 Continuous Deployment

We chose not to employ continuous deployment as this would affect the status of the production server. The production server would have to go down every change and it would affect user experience. We would prefer to have the choice of when to update the production build ourselves as we can notify users that the servers may be down prior to it.

## 7. Reflections

### 7.1 Future Enhancements

#### 7.1.1 Integrating Third-Party Authentication Services

The current version of PeerPrep only uses our own in-built password based authentication system. Although some security factors were considered and mitigations have been implemented, using a third-party authentication such as Firebase and GitHub authentication could lead to tighter security over the user's account. Furthermore, allowing a third-party authentication also simplifies the onboarding process by skipping the registration step. We could additionally add more social login options such as Google and Facebook in the event that users do not have a GitHub account.

#### 7.1.2 Collaborative Whiteboard

In the current version of PeerPrep, users are only allowed to convey their ideas either textually or verbally explaining to the other party. However, there are some concepts in Algorithms and Data Structures that benefit greatly from a visual representation. For example trees and graphs are notoriously difficult to represent using a textual representation but are easy to discuss using diagrams and sketches. This functionality could be implemented using Agora which provides an SDK for a real-time whiteboard feature.

#### 7.1.3 Specific Role Matching

We aim to implement the ability for a user to choose specifically a role to match as. We believe that users would enjoy the freedom of choice as some may not be open to interviewing others. This would also expand our demography as those who simply want to help others prepare for interviews and not prepare themselves are able to do so without being an interviewee themselves.

#### 7.1.4 Rating System

In the current version of PeerPrep, the only matching criteria is that users select the same difficulty. However, we recognise that there may be a difference in aptitude or experience of users matched. Perhaps in the future we could implement a system that rates both the interviewer and interviewee. In line with the above specific role matching, by doing this, we can match similarly-rated users who choose to partake in both interviewer and interviewee roles as helping out someone of the same level would benefit the users the greatest. On top of that, when only selecting a specific role such as an interviewer, stronger interviewers could be matched with weaker interviewees to expedite the interviewee's learning process and the interviewer would have a lot more to offer the interviewee.

## 7.2 Learning Points

### 7.2.1 Pros and cons of Microservice Architecture

Being introduced to Microservice Architecture and applying it to an actual project has led us to having deeper insights on what are the benefits and difficulties that come with it.

One of the greatest benefits of having a microservice architecture is that it allows for the development process to be highly parallelizable which is perfect for the sprint cycles that we have been following. It allows us to develop multiple functionalities concurrently without having to worry about having many dependencies on the others' work. This is possible due to the microservice architecture following the Single Responsibility Principle inherently which greatly reduces the coupling between modules.

However, the microservice architecture comes with the drawback of having a much higher complexity when compared to a monolithic architecture. We felt that one of the most challenging parts of using the Microservice Architecture is in secret management. This is due to each microservice being developed with their own unique techstacks which requires their own environment variables such as cloud database URLs and private container registries authentication details. Without a central secret manager we often fumbled when deploying our app locally since we did not have a secret environment variable for another service being developed. We also faced challenges trying to deploy our application to the cloud as we were all new to the concepts of Kubernetes and Docker.

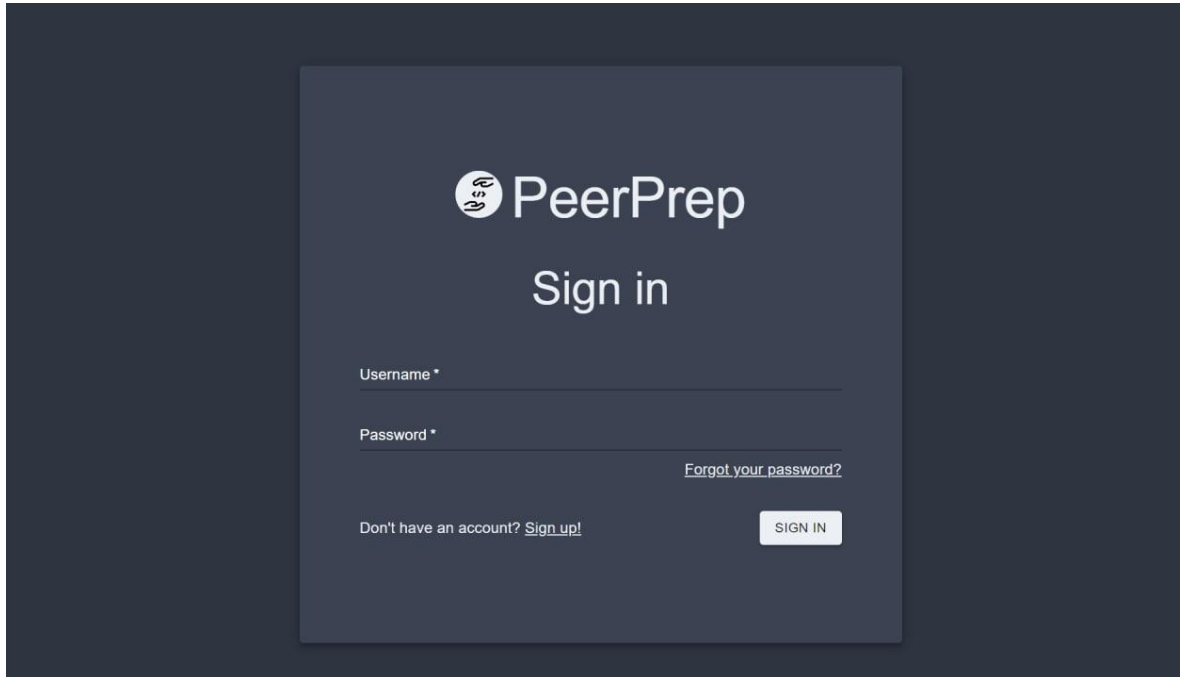
### 7.2.2 Importance of a well-defined requirement document

During the initial phase of the project, the importance of a well-defined requirement document was emphasised by the teaching team. Although we were all exposed to the concepts of use cases and user stories, this module goes a step further and requires us to produce a specification requirement early in the project development cycle.

Having such a rigorous and well-defined requirement specification document has helped us scope the project and allowed us to plan our development process early on in the project. It served as an important guide for us as we know exactly when and for what purpose each feature is built for so as to achieve the requirements we set out to complete. With this document being produced early on in the development process, it allowed us to focus on the development of the product but yet ensured that we kept the development to a set of specifications, helping us not to sidetrack from the purpose of building the product.

## 8. Product Screenshots

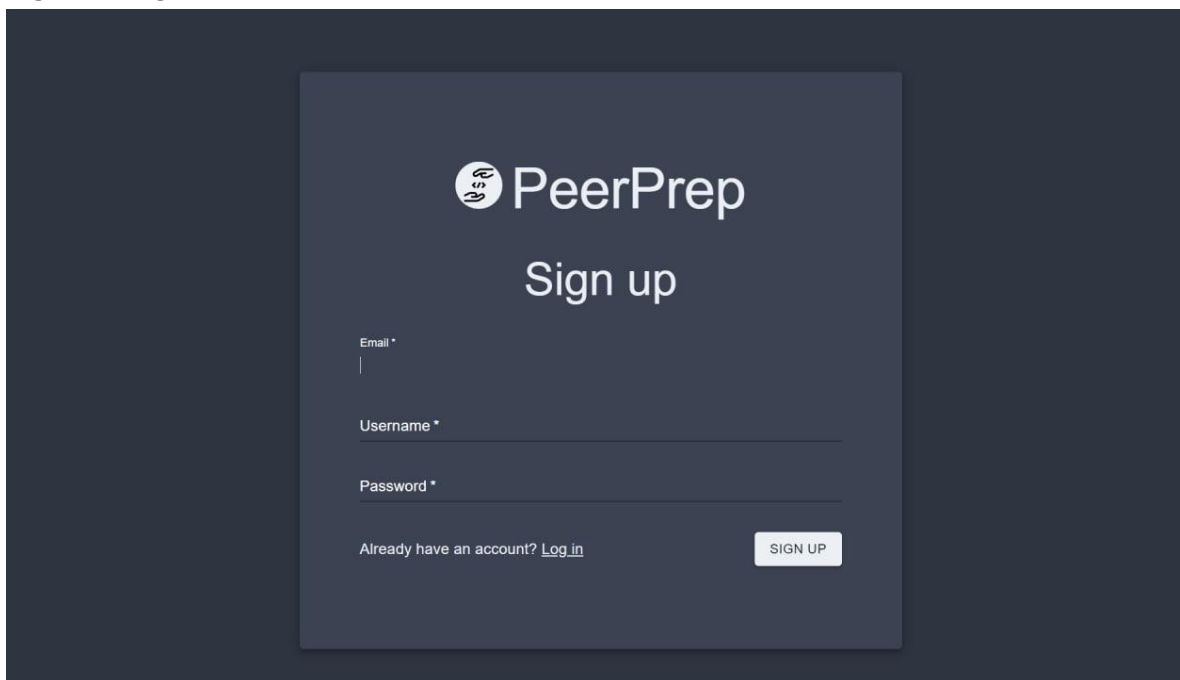
### Login Page:



The screenshot shows the PeerPrep 'Sign in' page. It features a dark blue background with a central white card. The card contains the PeerPrep logo (a stylized 'P' inside a circle) and the text 'Sign in'. Below the title are two input fields: 'Username \*' and 'Password \*'. To the right of the password field is a link that says 'Forgot your password?'. At the bottom left of the card is the text 'Don't have an account? [Sign up!](#)'. At the bottom right is a white button with the text 'SIGN IN'.

*Figure 8a: Landing/authentication page (sign in)*

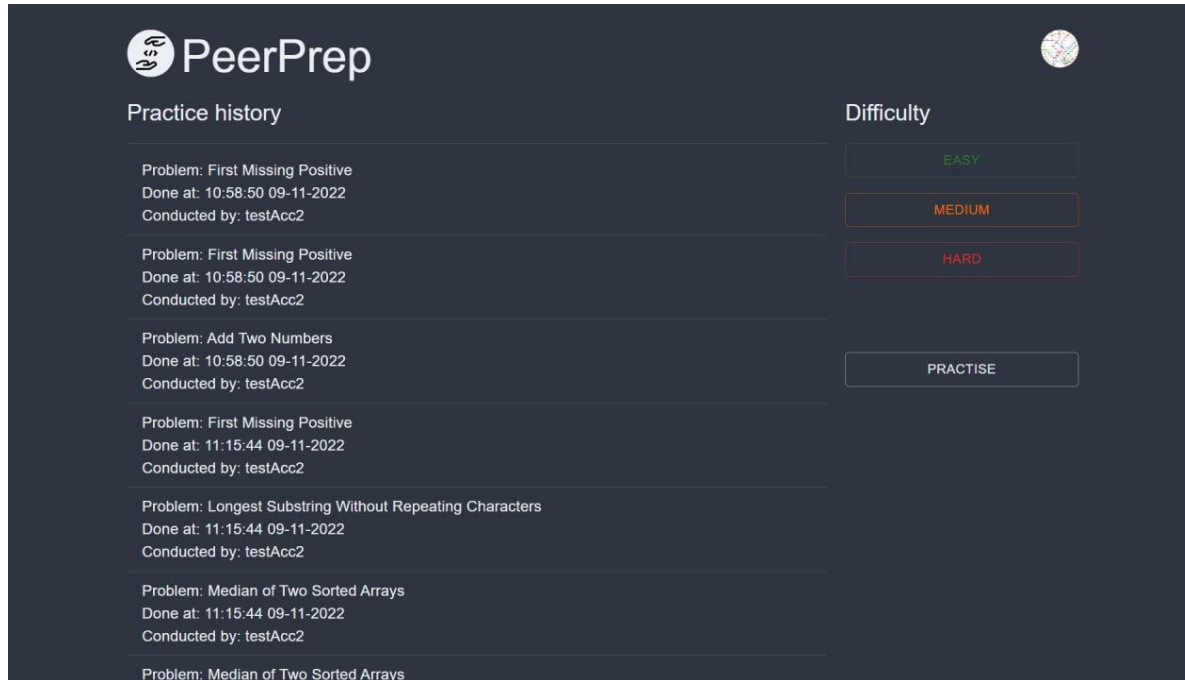
### Sign up page:



The screenshot shows the PeerPrep 'Sign up' page. It features a dark blue background with a central white card. The card contains the PeerPrep logo (a stylized 'P' inside a circle) and the text 'Sign up'. Below the title are three input fields: 'Email \*', 'Username \*', and 'Password \*'. At the bottom left of the card is the text 'Already have an account? [Log in](#)'. At the bottom right is a white button with the text 'SIGN UP'.

*Figure 8b: Landing/authentication page (sign up)*

## Dashboard Page:



The dashboard page features a dark blue background. At the top left is the PeerPrep logo, and at the top right is a circular profile picture placeholder. The main content is divided into two sections: 'Practice history' on the left and 'Difficulty' on the right.

**Practice history**

Problem: First Missing Positive Done at: 10:58:50 09-11-2022 Conducted by: testAcc2
Problem: First Missing Positive Done at: 10:58:50 09-11-2022 Conducted by: testAcc2
Problem: Add Two Numbers Done at: 10:58:50 09-11-2022 Conducted by: testAcc2
Problem: First Missing Positive Done at: 11:15:44 09-11-2022 Conducted by: testAcc2
Problem: Longest Substring Without Repeating Characters Done at: 11:15:44 09-11-2022 Conducted by: testAcc2
Problem: Median of Two Sorted Arrays Done at: 11:15:44 09-11-2022 Conducted by: testAcc2
Problem: Median of Two Sorted Arrays

**Difficulty**

EASY

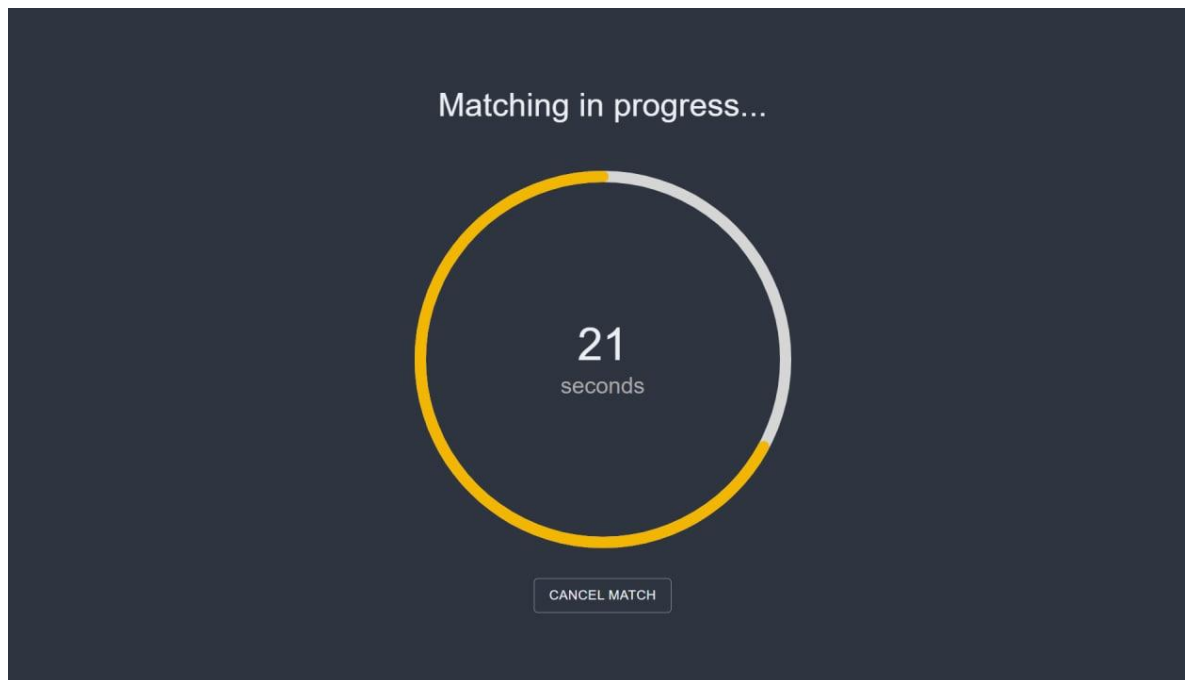
MEDIUM

HARD

PRACTISE

Figure 8c: Dashboard page

## Matching Page:



The matching page has a dark blue background. At the top, it says 'Matching in progress...'. In the center is a large circular progress indicator with a yellow segment and a white segment. Inside the circle, the text '21 seconds' is displayed. At the bottom, there is a button labeled 'CANCEL MATCH'.

Figure 8d: Matching modal