



**NUS**  
National University  
of Singapore

**AY23/24 Semester 1**  
**CS3219: Software Engineering Principles and Patterns**

Group 58 PeerPrep Final Report

Written by:  
Tang Bo Kuan (A0231051B)  
Ryan Chua Hao Jian (A0233569W)  
Wong Jia Jun (A0233598R)  
Alexander Lee (A0233939U)  
Teo Hao Yu (A0217790Y)

Submission Date: 15 Nov 2023

# Table of Contents

<b>1. Introduction</b>	<b>4</b>
1.1 Background	4
1.2 Purpose	4
<b>2. Contribution</b>	<b>5</b>
<b>3. Requirements</b>	<b>6</b>
3.1 Functional Requirements	6
3.2 Non-Functional Requirements	9
3.2.1 Quality attributes	9
3.2.2 Demonstration of NFRs	10
3.2.2.1 Compatibility	10
3.2.2.2 Usability	13
3.2.2.3 Security	17
<b>4. Developer Documentation</b>	<b>20</b>
4.1 Architecture	20
4.1.1 Overall User Flow	22
4.2 Design Patterns	23
4.2.1 Request/reply	23
4.2.1.1 Other Pattern Considerations	23
4.2.1.2 Rationale for this pattern	24
4.2.2 Pub-sub	25
4.2.2.1 Rationale for this pattern	25
4.3.1 Rationale for Tech Stack	27
4.3.1.1 React	27
4.3.1.2 Material UI (MUI)	28
4.3.1.3 MongoDB	28
4.3.1.4 Express.js	28
4.3.1.5 Third-Party Libraries	29
4.4 Microservices	31
4.4.1 User Service	31
4.4.1.1 User Management	31
4.4.1.2 Authentication	32
4.4.1.3 Authorization	33
4.4.1.4 Admin Invitations	33
4.4.2 Question service	35
4.4.2.1 Technologies used	35
4.4.2.2 View all questions in Question Bank	35
4.4.2.3 Question Filtering	36
4.4.2.4 Question History	37
4.4.3 Matching service	38
4.4.4 Collaboration service	39
4.4.4.1 Code Editor	40

4.4.4.2 Video/Audio communication	40
4.4.4.3 Interviewer/interviewee roles	41
4.4.4.4 Other Features	42
4.4.5 Email service	45
4.4.5.1 Technologies used	45
4.4.5.2 How it works	45
4.4.5.3 Security Measures	45
4.4.6 Compiler service	46
4.4.7 AI Chatbot service	51
4.5 Frontend	52
4.5.1 React's Context-Provider	52
4.6 Gateway	53
4.7 Deployment	54
4.7.1 Frontend Deployment	54
4.7.2 Microservices Deployment	54
4.8 CI/CD	57
<b>5. Reflections</b>	<b>59</b>
5.1 Further Enhancements	59
5.1.1 Integrating third-party authentication services	59
5.1.2 CI with Comprehensive Tests	59
5.1.3 Enhanced Collaboration features	59
<b>5.2 Reflections and Learning Points</b>	<b>60</b>
5.2.1 Pros and cons of Microservice Architecture	60
5.2.2 Importance of planning and well-defined requirements document	60

# **1. Introduction**

## **1.1 Background**

In recent years, students/developers have faced increasingly challenging interviews when applying for jobs which many have difficulty dealing with. Their weaknesses range from a lack of communication skills, articulating their thought process out loud to an inability to understand and solve the given problem. Moreover, simply practicing questions alone can be quite tedious and monotonous. To solve this issue, we have implemented a collaborative interview preparation platform called PeerPrep where students and junior/amateur developers can find peers to practice interview questions with.

PeerPrep is a platform that allows its users to match with each other and allow them to work together on a problem. In doing so, users can also simulate interviews by assuming the role of interviewer-interviewee and give each other a realistic interview experience.

## **1.2 Purpose**

Our purpose is to create a web application that helps students/amateurs in the tech industry better prepare themselves for mainly technical interviews. We aim to achieve this by using a peer learning system where users can learn from each other and curb the boredom and dread of practicing solo.

## **1.3 Target Audience**

Students, amateur/junior developers who wish to practice algorithmic questions and improve their communication skills

## 2. Contribution

Sub Group	Name	Contribution (Individual)	Contribution (Subgroup)
1	Tang Bo Kuan	<ul style="list-style-type: none"> <li>• Continuous Integration of services, gateway and frontend</li> <li>• Compiler Service</li> <li>• AI Service</li> <li>• Collaboration Service</li> <li>• Frontend</li> <li>• Wrote leetcode question descriptions and driver code</li> </ul>	<ul style="list-style-type: none"> <li>• Compiler Service</li> <li>• AI Service</li> </ul>
1	Wong Jia Jun	<ul style="list-style-type: none"> <li>• Collaboration service <ul style="list-style-type: none"> <li>◦ Implementation of interviewer/interviewee roles</li> </ul> </li> <li>• Frontend integration</li> </ul>	<ul style="list-style-type: none"> <li>• Video (collab page)</li> <li>• Audio communication (collab page)</li> </ul>
1	Teo Hao Yu	<ul style="list-style-type: none"> <li>• User Service <ul style="list-style-type: none"> <li>◦ User authentication and authorization</li> </ul> </li> <li>• Collaboration Service <ul style="list-style-type: none"> <li>◦ Code editor</li> </ul> </li> <li>• Implement frontend <ul style="list-style-type: none"> <li>◦ UI for collaboration page</li> <li>◦ UI for login page</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Syntax highlighting in code editor</li> </ul>
2	Ryan Chua Hao Jian	<ul style="list-style-type: none"> <li>• User Service <ul style="list-style-type: none"> <li>◦ User Management</li> </ul> </li> <li>• Email Service</li> <li>• Matching Service - frontend</li> <li>• Dockerization of User, Question and Email Service</li> <li>• Continuous Deployment of Services to AWS</li> <li>• Frontend</li> </ul>	<ul style="list-style-type: none"> <li>• Question History</li> <li>• Question Filtering</li> <li>• Deployment</li> </ul>
2	Alexander Lee	<ul style="list-style-type: none"> <li>• Matching Service <ul style="list-style-type: none"> <li>◦ Set up queue system and socket connections</li> </ul> </li> <li>• Gateway implementation</li> <li>• Question Service <ul style="list-style-type: none"> <li>◦ Set up schemas, models and query functions</li> <li>◦ Set up router for api calls</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Question History</li> <li>• Question Filtering</li> </ul>

## 3. Requirements

### 3.1 Functional Requirements

Requirement	Priority	Sprint
<b>F1 – User Management (M1)</b>		
F1.1 – User Registration		
F1.1.1 User should be able to create an account with an email	High ▾	1
F1.1.1 User should be able to create an account and set a password	High ▾	1
F1.2 – User Profile		
F1.2.1 User should be assigned roles to allow them to update and delete questions e.g admin, user	Medium ▾	1
F1.2.2 Only registered and authenticated users should be able to view questions	Low ▾	1
F1.2.3 Only registered and authorized users should be able to add, edit and delete questions	High ▾	
<b>F2 – User Matching (M2)</b>		
F2.1 – Matching Service		
F2.1.1 User should be able to match another user in real-time	High ▾	1
F2.1.2 User should be able to select difficulty and match another user of similar difficulty	High ▾	3
F2.1.3 User should time out after a set time and be prompted to match again if unsuccessful	High ▾	3
F2.1.4 Users that start matching first should be matched first e.g queue	Medium ▾	3
F2.1.4 Users should be matched based on their coding language preference	High ▾	3
<b>F3 – Question Service (M3)</b>		
F3.1 - Question repository CRUD		
F3.1.1 The system should be able to store/read questions with their corresponding information to/from the repository	High ▾	1

F3.1.2 The system should be able to add questions with their corresponding information	High ▾	1
F3.1.3 The system should be able to edit questions' information	High ▾	1
F3.1.4 The system should be able to delete questions from the repository	High ▾	1
F3.1.5 User should be able to filter the list of questions by a certain criteria (difficulty, topic, company, etc.)	Medium ▾	4
F3.2 Indexing questions from the repository		
F3.2.1 The system should be able to index questions based on one or more criteria	Medium ▾	
<b>F4 – Collaboration Service (M4)</b>		
F4.1 Concurrent code editing		
F4.1.1 Matched users should be able to see each other's code and changes on the collaborative space (like google docs)	High ▾	3
F4.1.2 Matched users should be able to edit the code on the collaborative space	High ▾	3
F4.2 Collaboration matching		
F4.2.1 Matched users should have the same coding language preferred	High ▾	4
<b>F5 – UI/UX (M5)</b>		
F5.1 User-friendly UI		
F5.1.1 Users should be able to intuitively use the app without a detailed tutorial	Low ▾	5
F5.1.2 The function of buttons represented by icons should be intuitively understood by users	Low ▾	5
F5.2 GUI based		
F5.2.1 The app should have no CLI interfaces and have a GUI	High ▾	1
<b>F6 – Deployment (M6)</b>		
F6.1 The app should be deployed on Docker	Medium ▾	3
<b>F7 – Communication (N1)</b>		
F7.1 Video chat		

F7.1.1 The app should allow users to communicate through a video call while in the collaborative space	Low ▾	4
F7.1.2 Users should be able to mute themselves and turn off video stream of themselves while in the collaborative space	Low ▾	4
<b>F8 – History (N2)</b>		
F8.1 Question attempt history		
F8.1.1 The app should maintain ALL past attempts and their relevant information along with its date/time on a question for a user	Low ▾	4
F8.1.2 Users should be able to view the attempt itself along with the date/time of it	Low ▾	4
<b>F9 – Compilation (N3)</b>	Low ▾	4
F9.1 Compilation of attempts		
F9.1.1 Users should be able to have the option to compile their code	Low ▾	4
F9.1.2 Users should be able to see the stdout, time taken, memory used, stderr of their code	Low ▾	4
F9.1.3 Users should be able to see how many test cases their code passed if there are test cases available	Low ▾	4
<b>F10 – Managing questions (N4)</b>		
F10.1 User should be able to choose to view questions of a particular topic	Low ▾	4
<b>F11 – AI helper (N7)</b>		
F11.1 AI Chat Bot assistance		
F11.1.1 Users (interviewers) should be able to send queries to an AI Chat Bot	Low ▾	4
F11.1.2 Users should be able to view responses of their queries from the AI Chat Bot	Low ▾	4



## 3.2 Non-Functional Requirements

### 3.2.1 Quality attributes

The following trade-off matrix demonstrates how we prioritized software quality attributes:

	Availability	Security	Usability	Performance	Compatibility	Scalability
Availability		-	-	-	-	-
Security			-	+	-	+
Usability				+	-	+
Performance					-	-
Compatibility						+
Scalability						

From the table, these are the 3 most prioritized quality attributes we had for PeerPrep, in this order:

1. Compatibility
2. Usability
3. Security

## 3.2.2 Demonstration of NFRs

### 3.2.2.1 Compatibility

We define compatibility as the capability of PeerPrep to operate on different platforms or operating environments. This is particularly important in a diverse user base using different types of devices and operating systems.

Requirement	Priority
<b>NFR 1 – Compatibility</b>	
NFR1.1 User should be able to access PeerPrep on any desktop browser	High ▾
NFR1.2 User should be able to access PeerPrep on any desktop Operating System (OS)	High ▾

#### NFR1.1 - User should be able to access PeerPrep on any desktop browser

We decided to prioritize this as 'High' as we aim to achieve broad user accessibility. By supporting all desktop browsers, PeerPrep becomes accessible to a wider audience. Users have different preferences and constraints regarding browser choice, and accommodating this diversity is crucial for maximizing reach and inclusivity. Prioritizing universal browser compatibility for PeerPrep aligns with user experience enhancement, technical feasibility, and market expansion goals, making it a high-priority non-functional requirement.

To support this NFR, we deployed PeerPrep on AWS and Vercel, accessible on [peerprep.ryanchuahj.com](https://peerprep.ryanchuahj.com). The deployment ensures that PeerPrep is compatible with a wide range of desktop browsers, such as Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, and others. This cross-browser compatibility eliminates barriers for users who may have different preferences or constraints when choosing a browser.

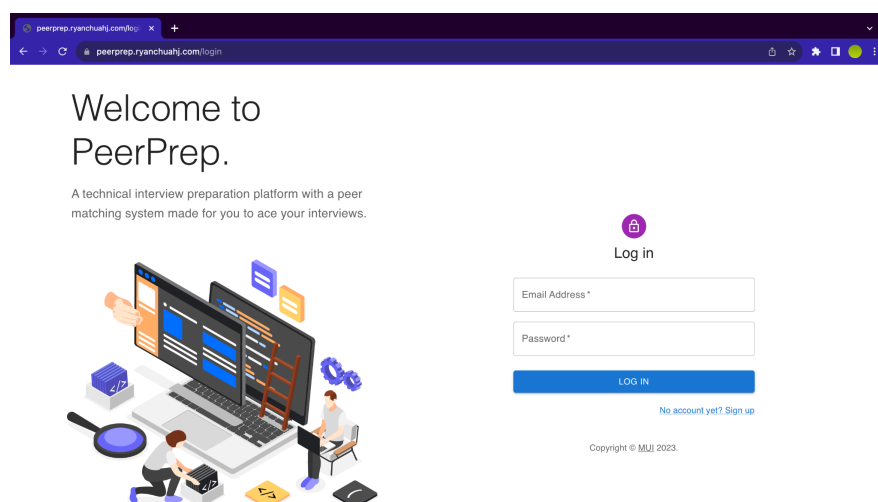


Figure 3.2.2.1a: Screenshot of compatible browser

## **NFR1.2 - User should be able to access PeerPrep on any desktop Operating System (OS)**

While NFR1.1 includes this, this NFR is coming from the development perspective. Having PeerPrep to be OS-agnostic improves development efficiency, as our development team can focus on creating a single, unified codebase that works across multiple operating systems, rather than maintaining separate versions for each OS.

We used Docker, which can create containers to package applications into a standardized unit for software development. PeerPrep will run the same way, regardless of where the Docker container is deployed – be it on different operating systems, cloud environments, or different developers' machines.

Docker also ensures that PeerPrep is run in a consistent and isolated environment. This isolation ensures that the application is not affected by the host operating system, leading to fewer compatibility issues.

Moreover, using Docker during PeerPrep's development is entwined with a number of fundamental DevOps procedures. As was already mentioned, the environment's consistency throughout the development process reduces "works on my machine" issues and increases team agility and efficiency. Additionally, Docker offers the flexibility needed to scale the application effectively and quickly - a crucial component of responsive DevOps practices.

To sum up, the DevOps philosophy of optimizing development and operations, guaranteeing reliability, scalability, and maintainability, while also improving team cooperation and workflow efficiency, is embodied by Docker's inclusion into PeerPrep's development pipeline.

```

1  version: '3.8'
2
3  x-logging:
4    &default-logging
5  logging:
6    driver: json-file
7    options:
8      max-size: 100m
9
10 services:
11   frontend:
12     depends_on:
13       - gateway
14     container_name: frontend
15     build:
16       context: .
17       dockerfile: ./frontend/Dockerfile
18     ports:
19       - "3000:3000"
20     env_file:
21       - ./frontend/.env
22     environment: # Running locally has different url than running on docker (not localhost)
23       - NEXT_PUBLIC_MATCHING_SERVER_URL=http://matching:3004
24       - NEXT_PUBLIC_COLLAB_SERVER_URL=http://collaboration:3005
25       - GATEWAY_SERVER_URL=http://gateway:8080
26
27   gateway:
28     depends_on:
29       - users
30       - questions
31       - matching
32       - collaboration
33       - compiler
34       - ai
35       - email
36     container_name: gateway
37     build:
38       context: .
39       dockerfile: ./gateway/Dockerfile
40     ports:
41       - "8080:8080"
42     environment:
43       - USER_SERVICE_URL=http://users:3001
44       - QUESTION_SERVICE_URL=http://questions:3002
45       - RABBITMQ_URL=amqp://user:password@rabbitmq:5672
46
47   users:
48     depends_on:
49       - email
50     container_name: user-service
51     build:
52       context: .

```

Figure 3.2.2.1b: Screenshot of Docker-compose file

```
1  version: '3.8'
2
3  x-logging:
4    &default-logging
5    logging:
6      driver: json-file
7      options:
8        max-size: 100m
9
10 services:
11   gateway:
12     depends_on:
13       - users
14       - questions
15       - matching
16       - collaboration
17       - compiler
18       - ai
19       - email
20     container_name: gateway
21     build:
22       context: .
23       dockerfile: ./gateway/Dockerfile
24     ports:
25       - "8080:8080"
26
27   users:
28     depends_on:
29       - email
30     container_name: user-service
31     build:
32       context: .
33       dockerfile: ./backend/user-service/Dockerfile
34     env_file:
35       - .env
36     ports:
37       - "3001:3001"
38
39   questions:
40     container_name: question-service
41     build:
42       context: .
43       dockerfile: ./backend/mongodb-database/Dockerfile
44     ports:
45       - "3002:3002"
46
47   matching:
48     container_name: matching-service
49     build:
50       context: .
51       dockerfile: ./backend/matching-service/Dockerfile
```

Figure 3.2.2.1c: Screenshot of Docker-compose file

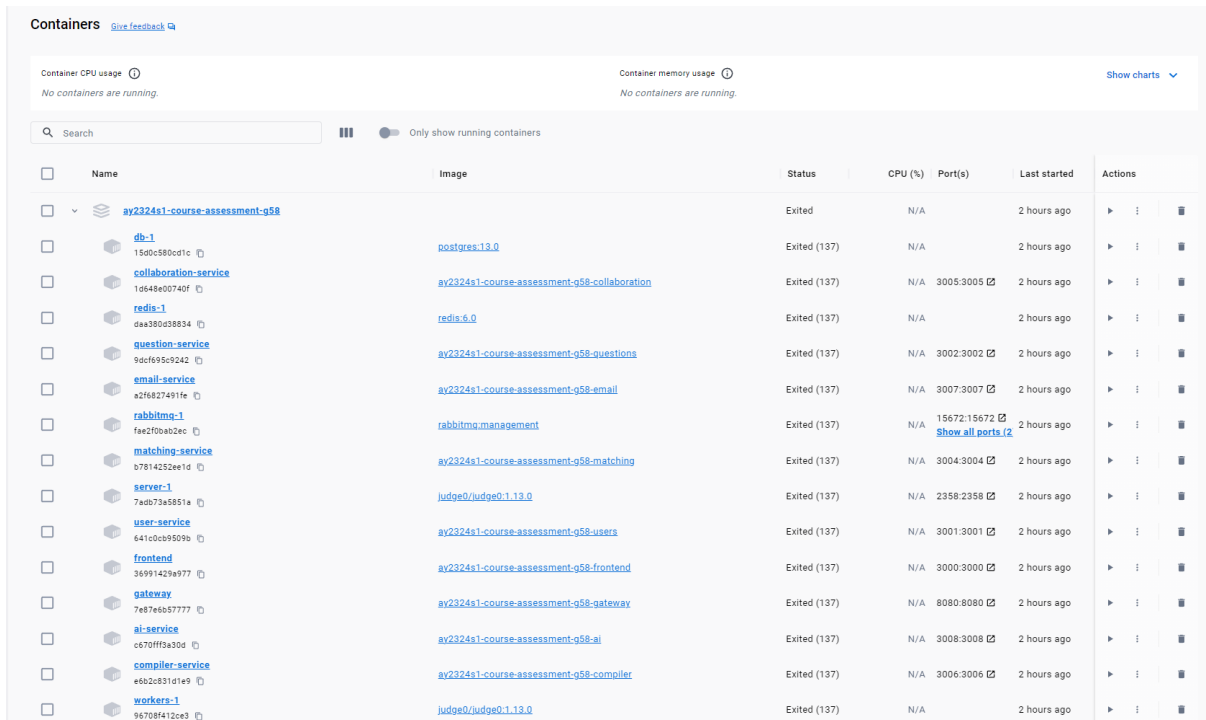


Figure 3.2.2.1d: Screenshot of Docker containers

### 3.2.2.2 Usability

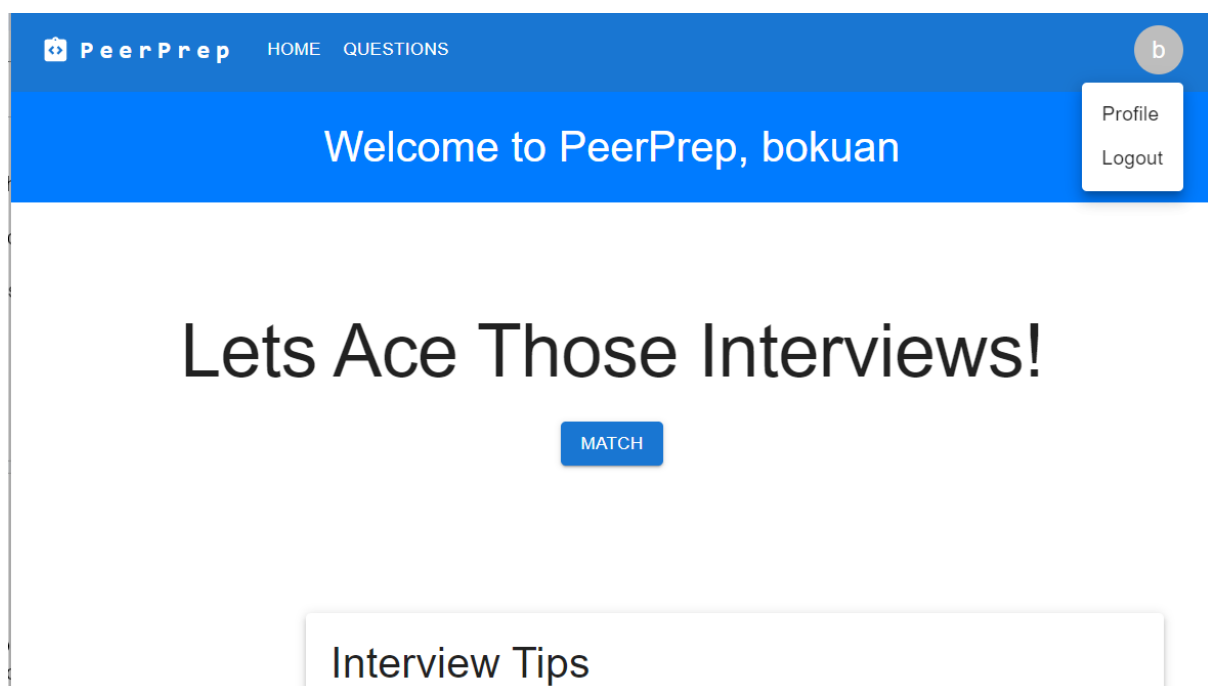
We aim to deliver a user-friendly and intuitive platform. PeerPrep should be easy to learn to use, efficient to use, minimize the frequency of errors for users, and maximize user satisfaction. It's particularly important in consumer-facing applications where user experience can significantly impact adoption and retention.

Requirement	Priority
<b>NFR 2 – Usability</b>	
NFR2.1 Users using different window sizes should be able to use PeerPrep	Low
NFR2.2 Users in the same collaboration room should only proceed to the next question or end the session with the other users' consent	Medium
NFR2.3 Users should be able to see changes with little delay in the collaboration room	High
NFR2.4 Users should know when they perform an illegal action and avoid them in the future	High
NFR2.5 Users should be able to view the GUI easily, regardless if they are in light or dark mode	Low

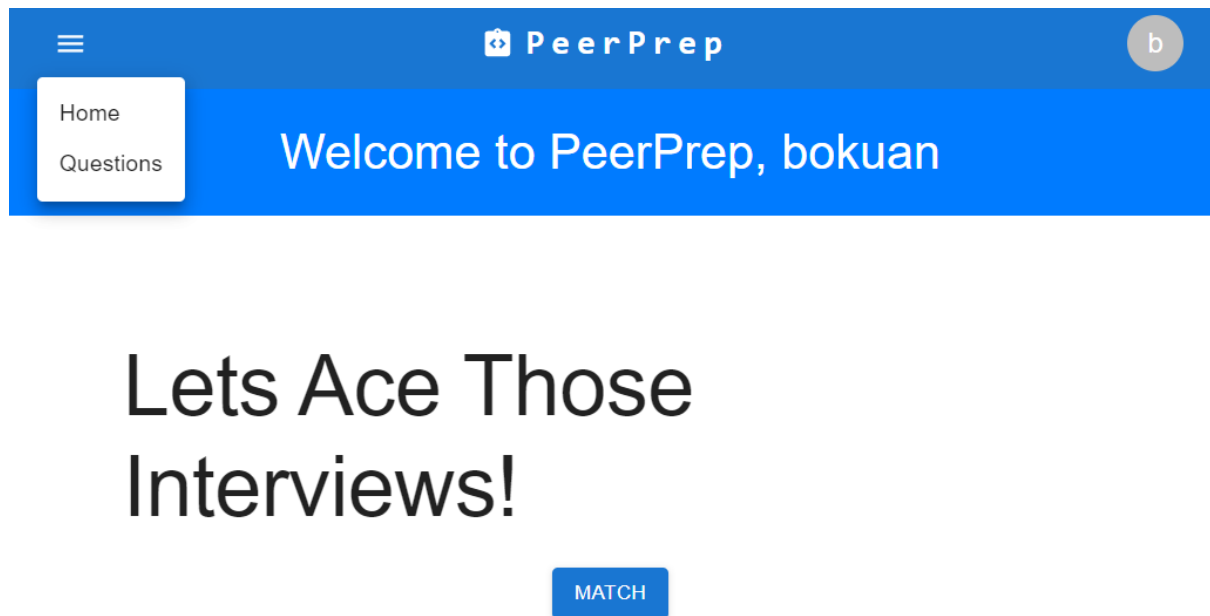
### **NFR2.1 Users using different window sizes should be able to use PeerPrep**

Although ensuring visual and functional consistency across various window sizes is crucial for user retention, as it caters to diverse hardware preferences and configurations, we have assigned a 'Low' priority to this NFR. This decision stems from the understanding that the primary usage scenario involves PeerPrep being operated in a minimized window on desktops, considering that full mobile compatibility is not yet implemented.

The implementation of this NFR is demonstrated through adaptive CSS styling within our React components. We have designed a dynamic navigation bar that adjusts based on screen resolution. At wider resolutions, the navigation bar is displayed in its complete form, offering full functionality. Conversely, at narrower resolutions, the navigation bar transitions to a more condensed format, optimized for limited screen space, though it's primarily targeted at minimized desktop windows rather than mobile devices.



*Figure 3.2.2.2a: Screenshot of navigation bar*



*Figure 3.2.2.2b: Screenshot of navigation bar*

**NFR2.2 Users in the same collaboration room should only proceed to the next question or end the session with the other users' consent**

In a collaborative learning and peer programming environment such as PeerPrep, maintaining a sense of control and mutual respect among users is nice to have. Therefore, we have identified NFR2.2, which mandates users' consent to proceed to the next question or end the session, as a medium priority.

When a user initiates a request to proceed to the next question or to conclude the session, PeerPrep ensures that both participants in the collaboration room are in agreement by displaying a confirmation prompt to each user. This action will only be executed if there is mutual consent from both users. In the event that either user is not ready to advance to the next question or to end the session, the request is automatically declined. This system is designed to respect the readiness and preferences of both users, ensuring a synchronized and collaborative experience.



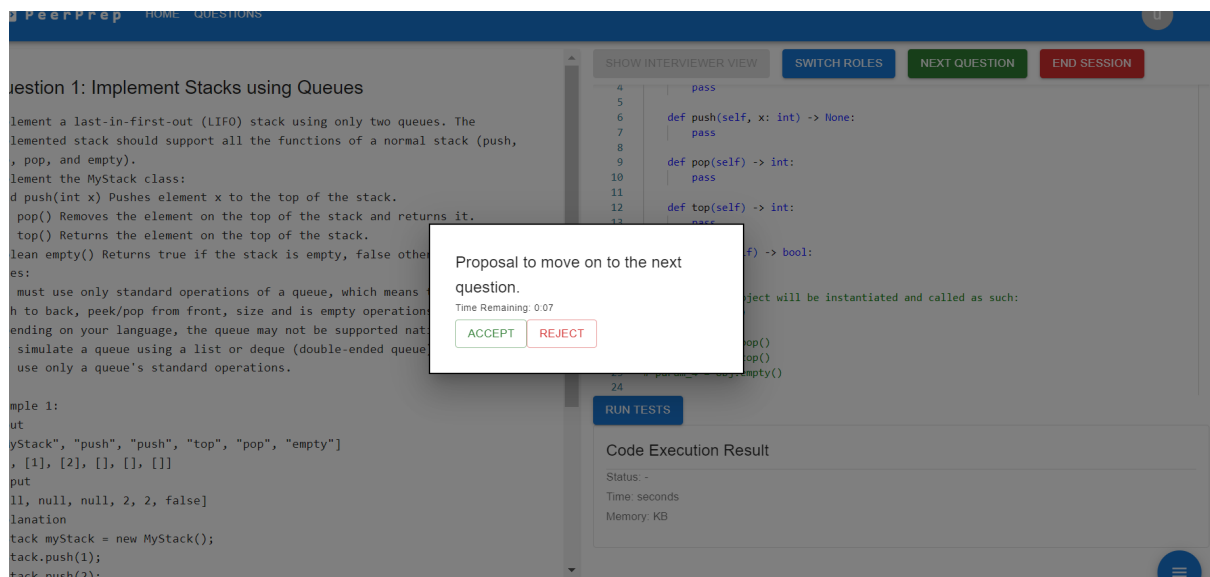


Figure 3.2.2.2c: Confirmation to move to next question

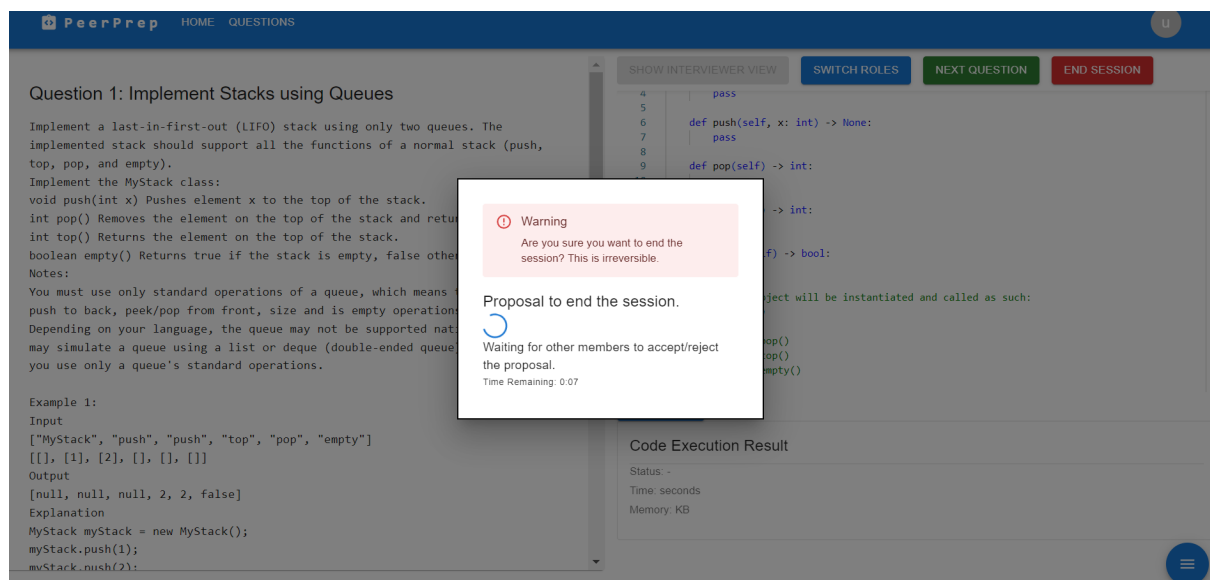


Figure 3.2.2.2d: Awaiting confirmation from other user to end session

### NFR2.3 Users should be able to see changes with little delay in the collaboration room

We have assigned a high priority to NFR2.3, highlighting the necessity for users to experience minimal delay in observing changes within the collaboration room. We recognized that a highly responsive and low-latency environment is pivotal to an enhanced user experience, so we implemented this feature with great care. To achieve this, we have utilized Socket.IO, leveraging the capabilities of WebSockets. This technology choice is instrumental in ensuring real-time, bidirectional communication, which is essential for maintaining a smooth and interactive collaborative experience. By minimizing delays in the reflection of changes, we facilitate a more seamless and efficient collaboration process, closely mirroring the dynamics of in-person interaction. This commitment to a low-latency environment is central to our goal of providing an intuitive and user-friendly platform that meets the high standards of our users' expectations for real-time collaboration.

## NFR2.4 Users should know when they perform an illegal action and avoid them in the future

We aim to improve the UI/UX of our platform by ensuring that users receive clear feedback whenever an action cannot be performed. Understanding that comprehensive feedback is crucial for a satisfying user experience, we've focused on effectively communicating why certain actions are not feasible. To achieve this, we have employed Material UI's snackbar component, which allow us to present feedback in a visually appealing and user-friendly manner.

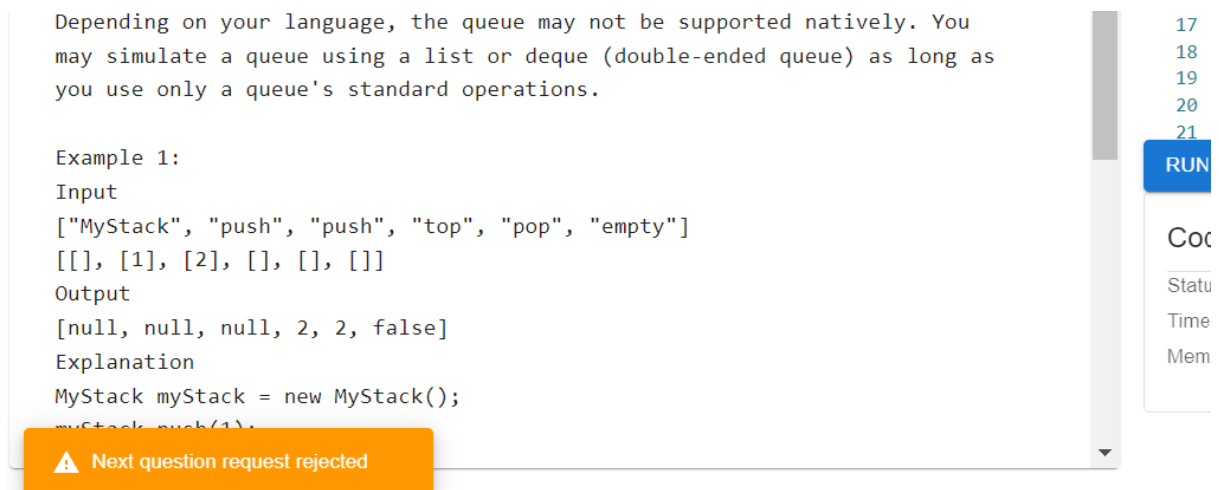
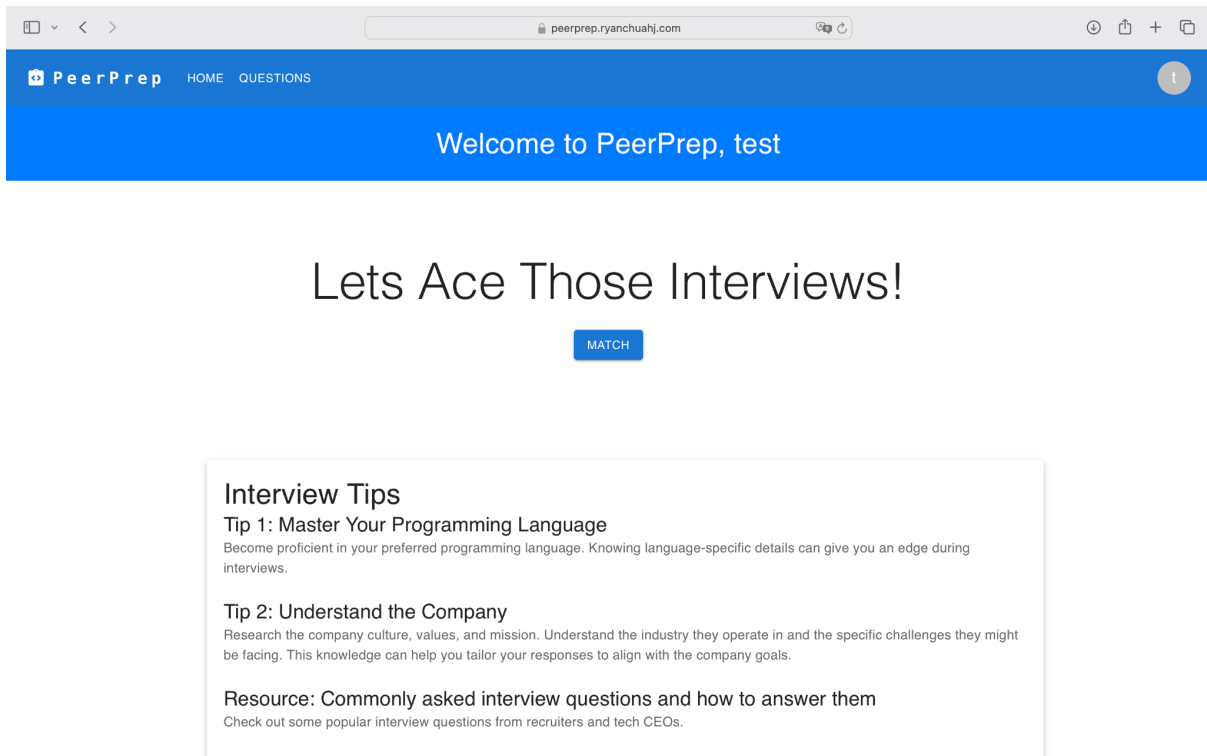


Figure 3.2.2.2e: Snackbar showing feedback why not advanced to next question

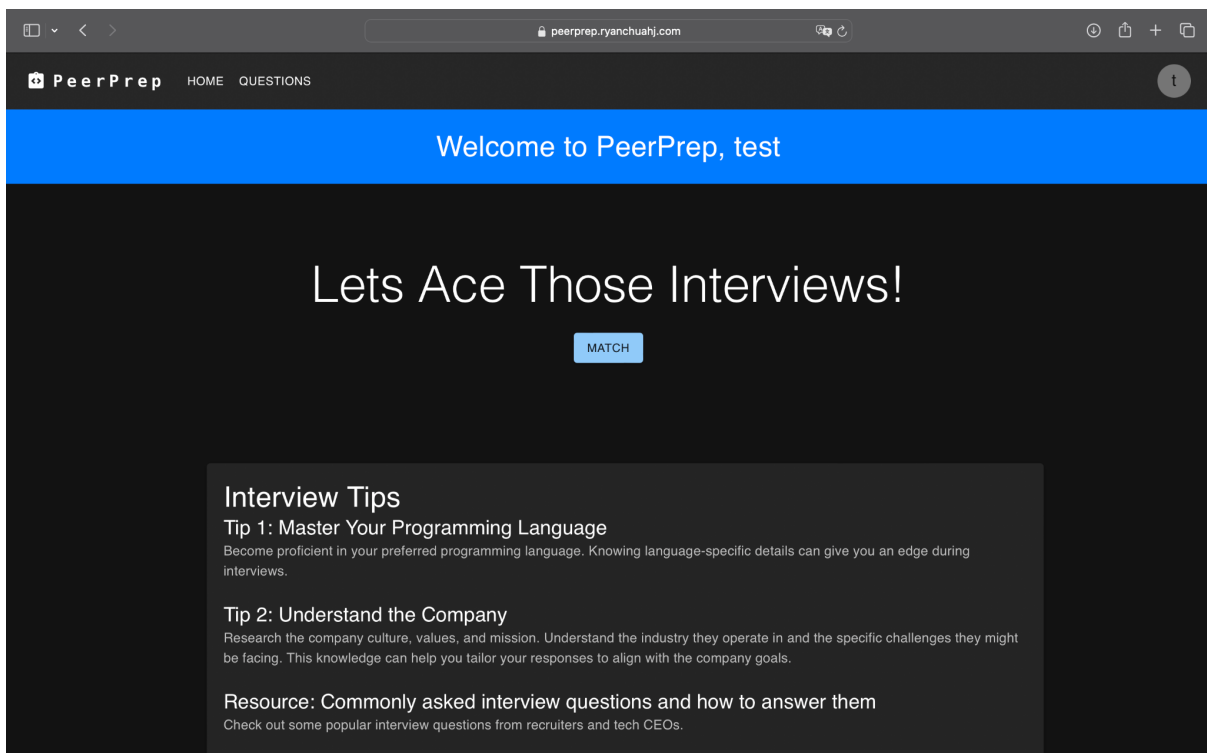
## NFR2.5 Users should be able to view the GUI easily, regardless if they are in light or dark mode

We aim to make the UI/UX of our platform seamless and minimalist. Hence, we implemented a theme-adjusting functionality within the frontend of our web application which detects whether the user's system is in light mode or dark mode. Upon landing on the page, the application detects the user's light/dark mode preference and changes the theme to follow that of the user's. When the user changes their preference, the frontend, which is subscribed to such changes, also propagates the change to the theme of the application.

This functionality ensures that the page always follows the color preference of the user, ensuring a seamless and user-friendly experience.



*Figure 3.2.2.2f: UI while user is in light mode*



*Figure 3.2.2.2g: UI while user is in dark mode*






### 3.2.2.3 Security

Security in PeerPrep pertains to the protection of information and systems from unauthorized access, use, disclosure, disruption, modification, or destruction. It encompasses various aspects including data confidentiality, integrity, and authentication.

Requirement	Priority
<b>NFR 3 – Security</b>	
F3.1 User passwords stored in the database must be securely hashed to ensure robust protection against unauthorized access and breaches.	High ▾
F3.2 Only admins of PeerPrep can access services to add and edit questions	High ▾
F3.3 JWT tokens should expire after 24 hours	Medium ▾
F3.4 Users should only be able to become an admin from a secure email invitation from another admin	Low ▾

#### **F3.1 User passwords stored in the database must be securely hashed to ensure robust protection against unauthorized access and breaches**

To protect user data and uphold confidence, it is essential to prioritize the security of user passwords using strong hashing algorithms. By adding a crucial layer of security, hashing passwords makes them very challenging to crack in the case of a database breach. This strategy not only demonstrates our dedication to user privacy and data protection, but it also conforms with best practices in cybersecurity.

	 id int8	 mail text	 created_at timestampz	 password text	 admin bool
<input type="checkbox"/>	2	okuan2000@gmail.com	2023-11-14 14:45:16.571+00	5dff644d28ccd903ff1fbb40b069c8c:12a1	TRUE

*Figure 3.2.2.3a: Passwords are hashed*

### F3.2 Only admins of PeerPrep can access services to add and edit questions

One important security precaution is to limit the ability to add and edit questions to PeerPrep admins only. By limiting the possibility of fraudulent or unauthorized content updates, this restriction protects the validity and dependability of the questions that users are asked. We protect the platform from possible tampering by strictly limiting access to these services.

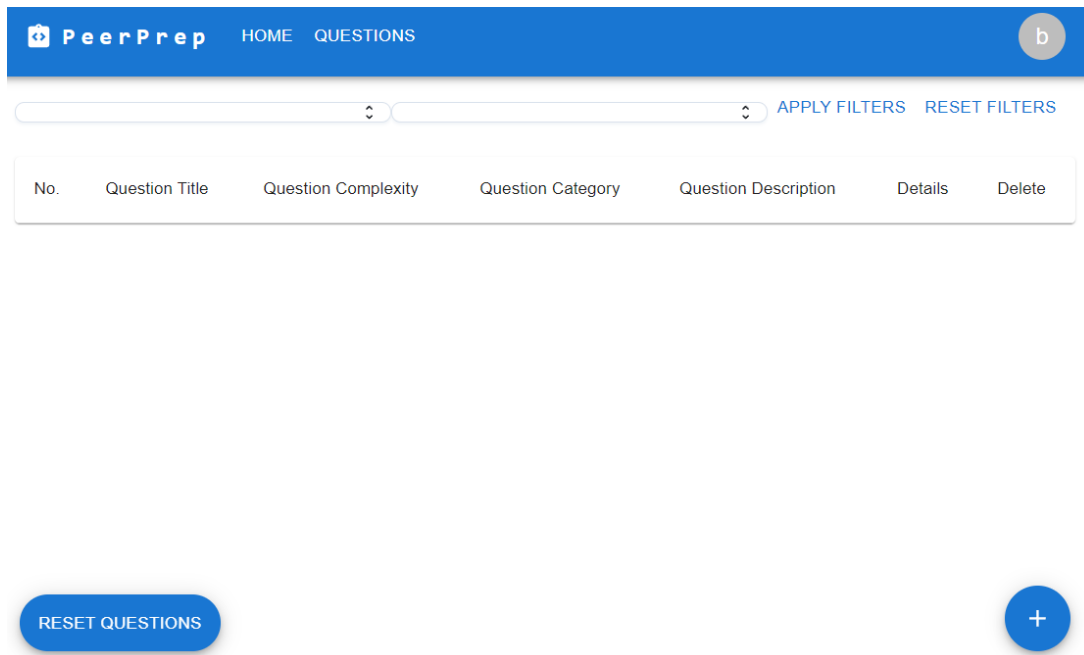


Figure 3.2.2.3b: Admin interface to add questions

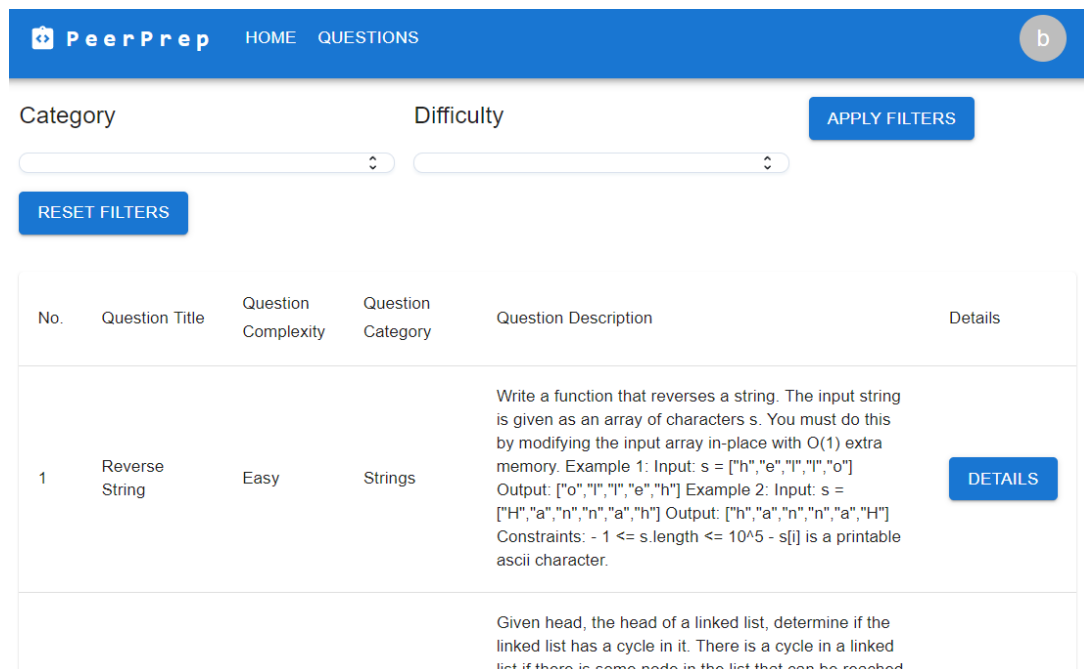


Fig 3.2.2.3c: Non-admin interface unable to add and edit questions

### F3.3 JWT tokens should expire after 24 hours

Adding a 24-hour expiration policy for JWT tokens strikes a balance between user convenience and security risks. Short-lived tokens reduce the window of opportunity for unauthorized access in case a token is compromised. This policy ensures regular token renewal, continuous verification of user credentials and maintaining a secure session state.

```
* The function generates a token using the user object and a secret key.
* @param {object} user - The `user` parameter is an object that represents the user for whom the token
* is being generated. It typically contains information such as the user's ID, username, and any other
* relevant data that needs to be included in the token.
* @returns a JSON Web Token (JWT) generated using the `jwt.sign()` method.
*/
export function generateToken(user: object) {
  return jwt.sign(user, process.env.JWT_ACCESS_TOKEN || "", { expiresIn: "1d" });
}
```

Fig 3.2.2.3d: 1 day expiry for JWT tokens

### F3.4 Users should only be able to become an admin from a secure email invitation from another admin

Enforcing that users can only attain admin status through a secure email invitation from an existing admin is a critical safeguard for maintaining the integrity and security of PeerPrep. This process ensures that administrative privileges are granted deliberately and only to trusted individuals.

We are able to ensure that the email invitation link is secure through encrypting parts of the url, authorization checks during creation of email and guarding the webpage from loading.

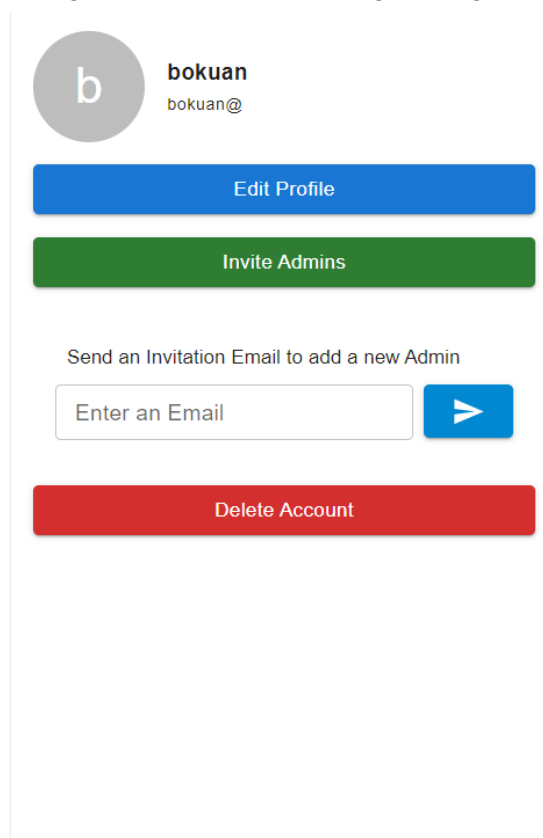
The image shows a user profile interface for an admin. At the top, there is a circular profile picture with the letter 'b', followed by the username 'bokuan' and the email 'bokuan@'. Below this, there are three buttons: 'Edit Profile' (blue), 'Invite Admins' (green), and 'Delete Account' (red). Under the 'Invite Admins' button, there is a text prompt 'Send an Invitation Email to add a new Admin'. Below this prompt is a text input field labeled 'Enter an Email' and a blue button with a right-pointing arrow. The entire interface is enclosed in a light gray border.

Fig 3.2.2.3e: Admin invite UI

## **4. Developer Documentation**

### **4.1 Architecture**

Our group decided to implement PeerPrep using the microservice architecture. Through discussion, we came to the conclusion that a microservice architecture is more suitable for PeerPrep as compared to a monolithic architecture.

Firstly, a microservice architecture allows for speedier development. A microservice architecture affords greater separation of concerns, where each service is tasked with responsibilities pertaining to a certain set of functions. The separation of concerns provided by the architecture means that each of our subteams can be responsible for the development, deployment, and maintenance of specific microservices.

Secondly, the use of a microservice architecture allows for better fault isolation. This is due to the loose coupling of a microservice architecture. Loose coupling of services contributes to better fault isolation by containing changes and faults. In the event that one microservice experiences a failure, it is less likely that the whole application would fail. In addition, in the event of bugs, they can be easily pinpointed to a specific microservice, without affecting the operations of the other services. Hence, new features and changes can be implemented with less deleterious effects.

Lastly, a microservice architecture provides good scalability. With a microservice architecture, we are able to scale each microservice independently based on its specific resource requirements. In the event that a certain service experiences heavier load, we are able to scale that particular service in a cost-efficient way, without scaling other services unnecessarily.

## Overall Architecture

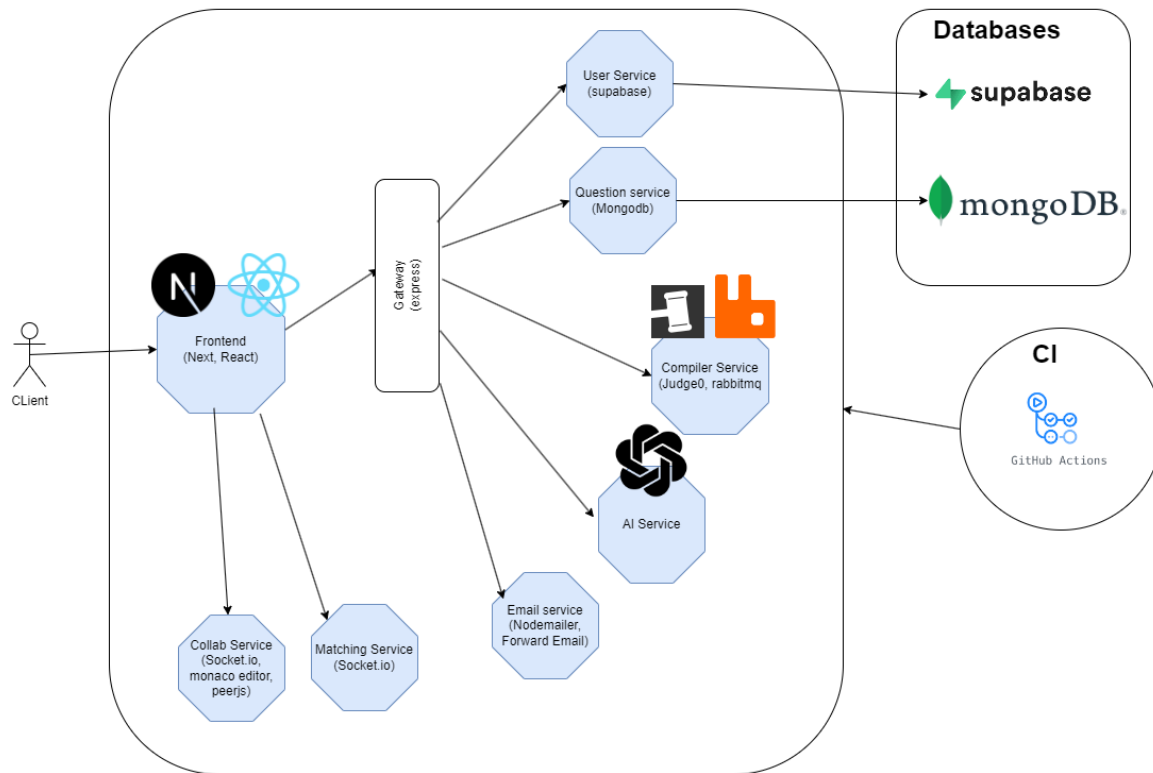


Figure 4.1a: Overall architecture diagram



## 4.1.1 Overall User Flow

User Flow Diagram

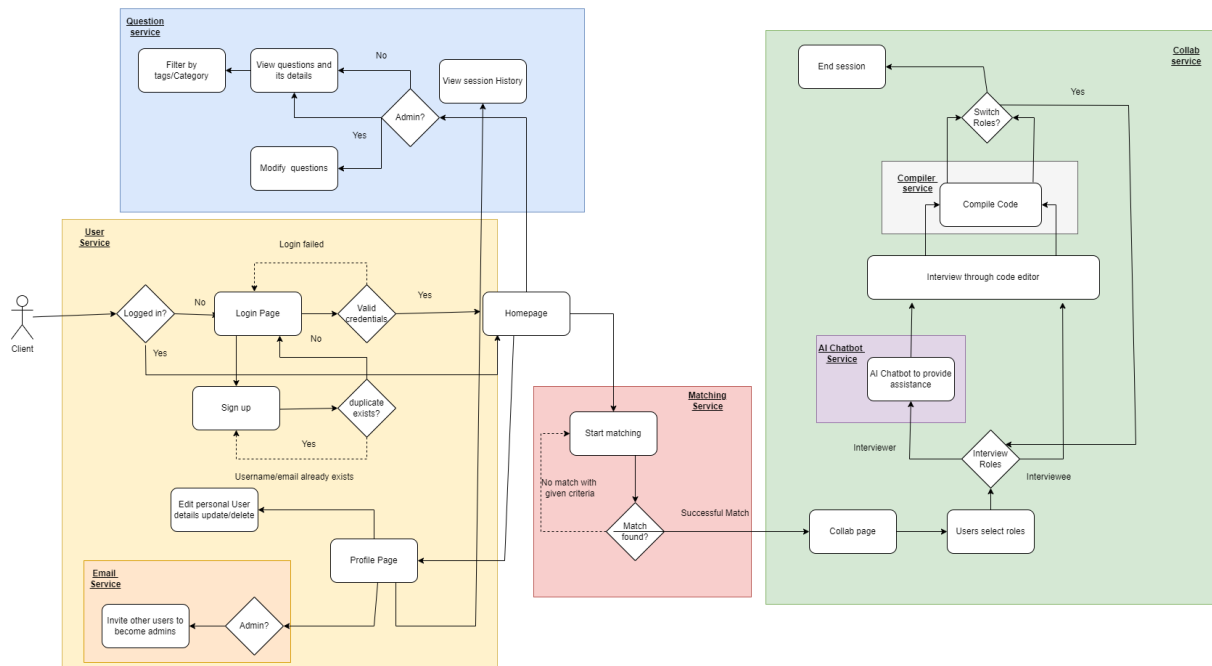


Figure 4.1.1a: Overall user flow diagram

## 4.2 Design Patterns

### 4.2.1 Request/reply

This section examines the Request/Reply pattern used in our **Compilation Service** and **AI Service**. This pattern facilitates a two-way conversation between the frontend and respective backend services, aligning perfectly with our requirements. We implemented this pattern using RabbitMQ, leveraging the Advanced Message Queuing Protocol (AMQP).

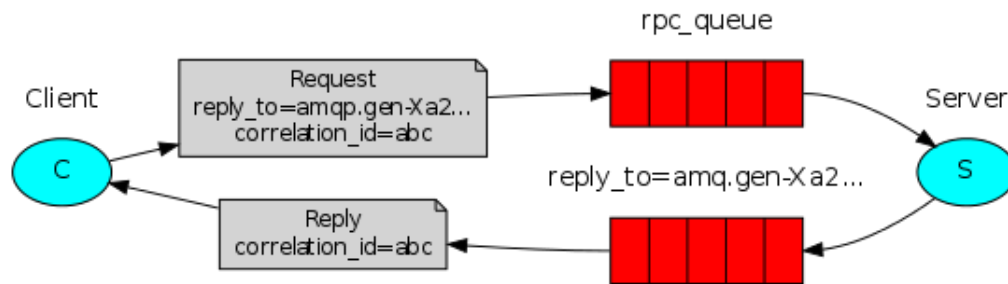


Fig 4.2.1a: RabbitMQ request/ reply pattern

#### 4.2.1.1 Other Pattern Considerations

Before settling on the Request/Reply pattern, we considered the Publisher-Subscriber (Pub-Sub) pattern. The Pub-Sub pattern, following an event-driven structure, designates event producers as publishers and consumers as subscribers.

##### How it would work

- Frontend requests for compilation or user prompts for GPT-3.5 turbo would be passed to our event manager.
- The Compilation Service and the AI Service, acting as subscribers, would consume these events.
- Post-processing, these microservices would publish the results (compilation or AI response) back to the event manager.
- The frontend, subscribed to events pertinent to each collaboration room, would receive these updates.

However, we concluded that this approach would overcomplicate our workflow. The nature of our operations aligns more closely with the Request/Reply pattern, where each collaboration room sends a specific request and awaits the corresponding response.

#### 4.2.1.2 Rationale for this pattern

The choice of the Request/Reply pattern for our Compilation and AI Services was driven by the unique communication demands of these systems. Utilizing RabbitMQ, our reasons for selecting this pattern include

Benefit	Justification
Two-way communication	Essential for scenarios where the frontend sends a request (such as code compilation or AI processing) and anticipates a specific, related response
Correlation of Requests and Responses	This pattern simplifies the correlation process in our environment that has concurrent requests, from unique collaboration rooms to two different services
Decoupling	Having RabbitMQ as a message broker provides decoupling as changes in one service doesn't affect the other as long as the message format is maintained.
Load Balancing	RabbitMQ effectively manages load balancing, which is crucial for our services that are prone to high traffic, like frequent compilation requests or AI prompts.

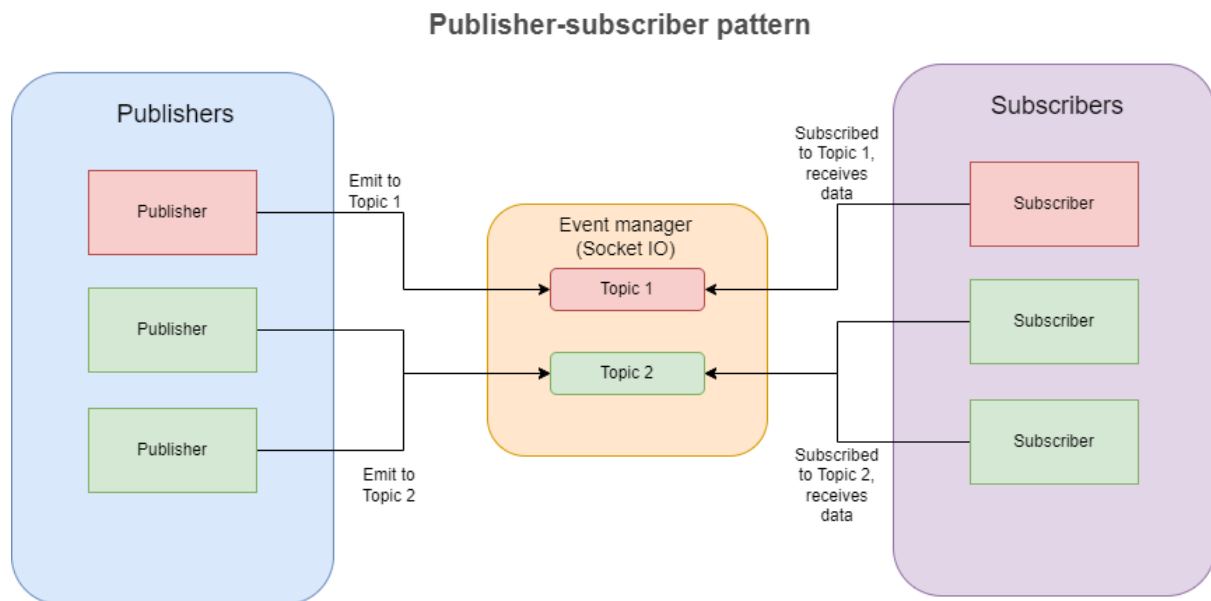
### 4.2.2 Pub-sub

The publisher-subscriber pattern was used in both our **Matching Service** and **Collaboration Service**. The pattern was upheld with the Socket.IO library, which establishes a TCP connection between the frontend and the respective backend service, enabling low-latency, duplex, event-driven communication.

#### 4.2.2.1 Rationale for this pattern

The publisher-subscriber pattern was strategically chosen for our Matching and Collaboration services to meet the requirements of a fast and reliable microservice architecture. This pattern offers several benefits:

Benefit	Justification
Scalability	The pub-sub pattern can easily accommodate an increasing number of subscribers without significantly impacting existing services. This is vital for PeerPrep as it grows in user base, ensuring that the system can handle a higher volume of simultaneous connections and interactions.
Real-Time Updates	Users receive near-immediate feedback and updates, which is essential for maintaining an interactive and dynamic collaboration environment. The pattern's event-driven nature ensures that actions in one client are promptly reflected in the other client's view.
Decoupling	Publishers and subscribers are loosely coupled. This allows for easier maintenance and updates to the Matching and Collaboration services without disrupting the overall system.
Efficiency	The pattern is efficient in terms of network usage and data distribution. Only relevant data is sent to subscribers, reducing unnecessary data transmission and processing. This efficiency is crucial in a high-traffic environment like PeerPrep.



*Fig 4.2.2.1a: Pub-sub pattern*

The diagram above visualizes the application of the publisher-subscriber pattern within our specified microservices:

1. **User Connections as Subscribers:** Upon connection, a pair of PeerPrep users are modeled as subscribers to a shared topic – typically representing their collaboration room or the match-making queue..
2. **Users as Publishers:** Simultaneously, these users are also considered publishers. They can publish events – such as a match request or an action within the collaboration room – to their shared topic
3. **Event Broadcasting:** Any event published by a user is instantly broadcasted to the other user, who is a subscriber to the same topic. This ensures real-time, synchronous communication between users
4. **Efficient Communication Model:** This model enables low-latency, duplex, event-driven communication between the frontend and backend services. Establishing a TCP connection through Socket.IO ensures that data exchange is prompt and reliable, enhancing user interaction and experience

This implementation of the publisher-subscriber pattern with Socket.IO is fundamental to achieving the desired performance and scalability in PeerPrep. It allows for efficient management of real-time data flows and user interactions, crucial for the platform's collaborative and dynamic environment.

## 4.3 Tech Stack

Component	Application or Framework	Comments (if any)
Frontend	React, NextJS	
Backend	Express	
Database	Supabase (user-service) MongoDB (question-service)	
Cloud Providers	Amazon Web Services (AWS) Vercel	Vercel is used for frontend deployment.  AWS is used for microservice deployment.  List of services used for AWS: Application Load Balancers Elastic Compute Cloud (EC2) Elastic Container Registry (ECR) Elastic Container Service (ECS) Message Queue (MQ) Route 53
CI/CD	GitHub Actions	
Project Management Tools	GitHub Issue Tracker	

### 4.3.1 Rationale for Tech Stack

We choose the MERN (MongoDB, Express.js, React.js, Node.js) stack for deploying PeerPrep as it offers several advantages suitable for our project. The MERN stack is preferred as all the components are JS-based. Hence, we will be able to integrate the different services easily. In addition, the MERN stack is [optimised to be used in a microservices architecture](#).

#### 4.3.1.1 React

We chose React as a frontend framework due to its component-based architecture and virtual DOM. This makes it well-suited for dynamic and interactive user interfaces, which is crucial for a coding collaboration web application, where there will be many real-time changes and updates to the frontend.

Additionally, React was the frontend framework of choice due to the group's familiarity with it and its ease of use.

#### 4.3.1.2 Material UI (MUI)

Initially, our group decided on Tailwind CSS as a frontend framework due to its flexibility and maintainability, as it would allow us to design the frontend however we want without any constraints.

However, we soon found out that Tailwind was giving us certain compatibility issues. As we decided to implement Material UI components in our frontend in order to save time, Tailwind caused frontend glitches to occur. As a result, our group decided to switch over to Material UI as a frontend framework and library due to its clean and consistent design, as well as wide range of frontend components available for use.

By using Material UI, our group was able to save development time on constructing our own frontend components from scratch, which can often be challenging and time consuming due to issues arising from browser compatibility and testing. This then allowed our group to channel our effort into building up main and nice-to-have features for PeerPrep.

#### 4.3.1.3 MongoDB

We chose MongoDB as our database for flexibility and scalability. MongoDB's schema is flexible and allows for easy addition of fields without compromising the functionality of the previous entries.

MongoDB also has a powerful query language that is simple to pick up. Queries to the database can be formed in fewer lines compared to SQL statements.

#### 4.3.1.4 Express.js

We chose Express.js as a backend framework due to its simplicity and ease of use. Being a lightweight and minimal framework, it allows for us to rapidly set up routes, handle HTTP requests and build up a set of RESTful APIs for PeerPrep features without unnecessary complexity.

Express.js also utilizes a middleware architecture that enables us to plug in various modules and functionalities when handling backend requests. For example, authentication and authorization can be implemented as a middleware function that requests are routed through, before a response is generated and sent back to the client.

#### 4.3.1.5 Third-Party Libraries

This section contains some of the notable third-party libraries that were used during our development of PeerPrep

Library used	Description of library
Prisma	<p>Prisma is an open-source database toolkit that simplifies database access in modern web application development. It provides a set of tools and libraries for working with databases, both relational and non-relational, making it easier for us to interact with our backend databases through a type-safe and auto-generated query builder.</p> <p>Prisma helped us with abstracting away the low level SQL queries and language, speeding up our development of PeerPrep.</p>
Socket.IO	<p>Socket.IO is a library that enables low-latency, bidirectional and event-driven communication between clients. Socket.IO automatically picks a transport mode to establish a socket connection, either by HTTP long-polling, WebSocket or WebTransport, abstracting away the low level details of handling websockets.</p> <p>Socket.IO serves as the backbone for the majority of our code collaboration functionalities.</p>
NextJS	<p>NextJS is a popular React.js framework that simplifies the process of building React-based web applications.</p>
Express Gateway	<p>Express Gateway is a microservices API Gateway built on Express.js.</p>
Monaco Editor	<p>Monaco Editor is an open-source, feature rich code editor developed by Microsoft. It is designed to be embedded in web applications, and has wide documentation for the various APIs it provides to interface directly with it.</p> <p>Monaco Editor serves as the backbone for the code editor component in our frontend.</p>
Judge0	<p>Robust, scalable, and open-source online code execution system that can be used to build a wide range of applications that need online code execution features.</p>
OpenAI API	<p>Provides access to GPT models that can follow complex instructions in natural language.</p>
peerjs	<p>peerjs is an open-source JavaScript library that simplifies WebRTC (Web Real-Time Communication) peer-to-peer data, audio, and video calls. WebRTC is a free, open-source project that provides web browsers and mobile applications with real-time communication via simple application programming interfaces (APIs).</p>



	<p>PeerJS abstracts much of the complexity involved in setting up and managing WebRTC connections and its simple API is used to handle matched users' communication through video and audio chat in our front end</p>
Nodemailer	<p>Nodemailer is a module for Node.js that allows developers to send email from Node.js applications. It is a popular and widely used library for sending emails using Node.js. Nodemailer supports various transport methods, including SMTP, sendmail, and even direct transport. It provides a simple and flexible API for sending emails with attachments, HTML content, and more.</p> <p>Nodemailer enables our email service to send emails from our domain to invite users to our application. This enhances and improves our admin invitation functionality.</p>
Forward Email	<p>Forward Email is a free and open-source email forwarding service focused on a user's right to privacy, and was launched in November 2017. It offers unlimited custom domain names, unlimited email addresses and aliases, unlimited disposable email addresses, spam and phishing protection, and other features. The service is maintained and owned by its original founding team of designers and developers. It is built with 100% open-source software using JavaScript, Node.js, DNS, HTTPS, TLS, and SMTP.</p> <p>Forward Email is used to forward emails created by nodemailer to the respective recipients via our custom domain.</p>

## 4.4 Microservices

### 4.4.1 User Service

The User Service forms the foundation of PeerPrep, as users are required to create accounts and be authenticated before entering the application. The User Service stores its information in a Supabase PostgreSQL database, and uses Prisma as an Object Relational Mapper. The Entity Relational Diagram below describes the relational database.

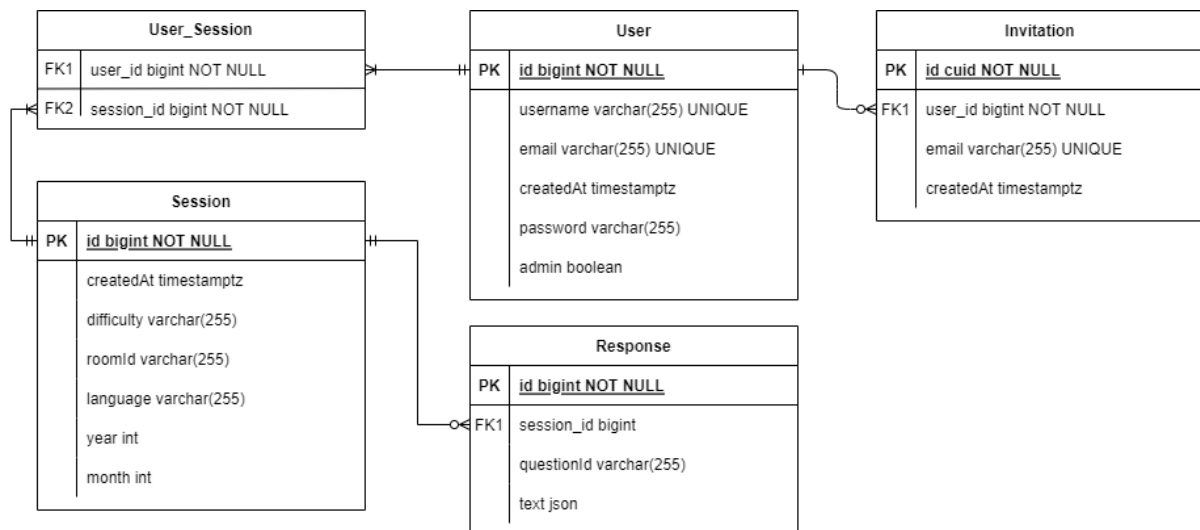


Figure 4.4.1.a: ER Diagram for relational database of user service

#### 4.4.1.1 User Management

User Service achieves this by exposing four endpoints for GET, POST, UPDATE and DELETE operations, to support user retrieval, creation, updating and deletion, respectively in the frontend.

The following fields are stored in the Users table:

- id
- username
- email
- createdAt
- password
- admin
- session
- invitation

Note that the password has been hashed and is not visible to developers. Also, the session field is linked to the session object discussed in section [4.4.2.1 Question History](#) and the invitation field is linked to the invitation object discussed in section [4.4.1.4 Admin Invitations](#).

For GET requests, the User Service returns all the users' emails and usernames.

For POST requests, the User Service first verifies if email or username already exists. If so, an error will be returned, informing that the respective email or username has already been taken. If an invitation is present, it will be deleted and an admin will be created in a single transaction. If no invitation is present, a non-admin user will be created.

For UPDATE requests, the User Service requires an email and username. It verifies that the user with specified email exists and no user with such a username exists. Thereafter, it updates the username of the specified user to the new username.

For DELETE requests, the User Service verifies that the user with the specified email exists before deleting the user.

#### 4.4.1.2 Authentication

Authentication is a crucial aspect of our MERN web application, ensuring that only authenticated users can access our question bank and web application. In this project, we have implemented a JSON Web Token (JWT) based authentication system. JWT is a compact, URL-safe means of representing claims between two parties. It allows us to authenticate users securely and manage sessions effectively.

When a new user signs up, the client sends a POST request to the user service via a POST request made to the respective endpoint. This request includes the user's registration details, such as username, email, and password. Upon receiving the signup request, the server securely hashes the user's password before storing it in the Supabase Postgres database. Hashing adds an additional layer of security by ensuring that plaintext passwords are not stored. We also decided on a PostGres SQL database for our user service, due to SQL's inherent flexibility and strength when it comes to storing relational data.

After a user has successfully signed up for an account, they will be able to login. When logging in, the user service first checks the user's credentials against the credentials stored in the database. If the credentials are verified, the server generates a JWT token using the user's details, such as their username and email. This JWT token is sent back to the user and stored in their local browser storage.

For security purposes, JWT tokens within our authentication flow are valid for a period of 24 hours. Within 24 hours, user's are able to stay logged in whenever they close the browser window without logging out. However, after a period of 24 hours, users will be then prompted to log in again. The validity period is variable, and can be adjusted, allowing for flexibility when it comes to security.

#### 4.4.1.3 Authorization

Authorization is also supported by JWTs in our user service. In the case of PeerPrep, a boolean field named “admin” in the JWT is set to either True or False, depending on whether the user in question has admin rights or not.

When a user has admin rights, they are able to add, update and delete questions from the question repository, as well as send invites to other users to become admins themselves (see section [4.4.5 Email service](#) for more information).

Certain API endpoints are also guarded through authorization. API endpoints which should only be called by admins are verified in the backend.

In the backend, there exists middleware functions, to authenticate and authorize users. We route token login requests (when users refresh the page) through the authenticate middleware function, which verifies the validity of the JWT token passed to the backend, and either grants or denies the login request. For admin invite requests, we route the requests through the authorize middleware function, to ensure that the user that made the API call is authorized to invite new admins to PeerPrep.

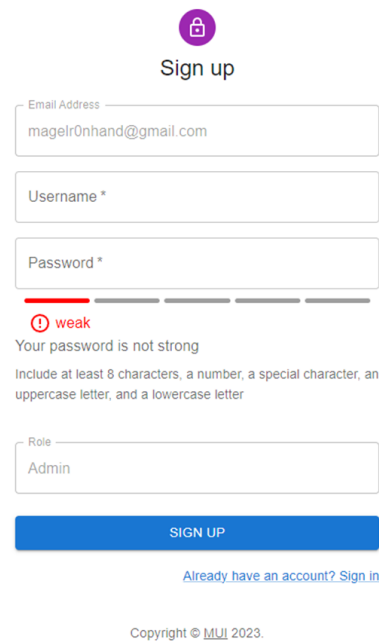
#### 4.4.1.4 Admin Invitations

Admin invitations are used as a method to allow existing admins to send invitation emails to invite new admins and prevent unwanted users from signing up as admins. An invitation has the following fields:

- id
- user\_id
- createdAt
- email

Note that the email specified will be the recipient of the email. Also, We limit the number of admins to 10, to prevent malicious actions by any existing admin. Specifically, during the API call, we sum the current number of admins with the current number of invitations to check if there are less than 10 invitations. Lastly, the invitation will expire in 7 days. This is to prevent any future cases of outdated invitations being accepted.

We protect the signing up of a new admin in three different areas. Firstly, only admins are able to create invitations in our database. This is done through the authorization of an endpoint as mentioned in the section above. Secondly, the invitation link generated has its id attached, making it difficult for malicious actors to forge the link. This is relevant as lastly, the admin signup page is protected by an API call that verifies that the invitation with the specified id and email exist in the database. Before loading the page, the frontend makes this API call and only loads the page if the invitation exists.



The image shows a web form titled "Sign up" with a purple lock icon above it. The form contains three input fields: "Email Address" with the value "mageir0nhand@gmail.com", "Username \*" (empty), and "Password \*" (empty). Below the password field is a strength indicator consisting of five bars, with the first bar highlighted in red and labeled "weak". Below this, a message states "Your password is not strong" and lists requirements: "Include at least 8 characters, a number, a special character, an uppercase letter, and a lowercase letter". There is a "Role" dropdown menu set to "Admin". At the bottom is a blue "SIGN UP" button. Below the button is a link: "Already have an account? Sign in". At the very bottom, it says "Copyright © MJJ 2023."

*Figure 4.4.1.b: Admin Signup page*

For invite route GET requests, the User Service returns all the invitations if present.

For invite route POST requests, the User Service first verifies if the inviter email is an admin. Next, the service will sum the total number of admins and invitations. If there are less than 10, the invitation will be created or updated if it already exists. It will then make a POST request to the email service to trigger the invitation email.

For invite route DELETE requests, the User Service first verifies if the inviter email is an admin. Thereafter, the service will delete the invitation with the specified email if it exists.

Lastly, the User Service also exposes a "/verify-invitation" POST endpoint, which verifies that the specified id matches the id of an existing invitation in the database, and is still valid.

## 4.4.2 Question service

### 4.4.2.1 Technologies used

For the question bank, we have decided to use MongoDB, a noSQL database, to store the questions.

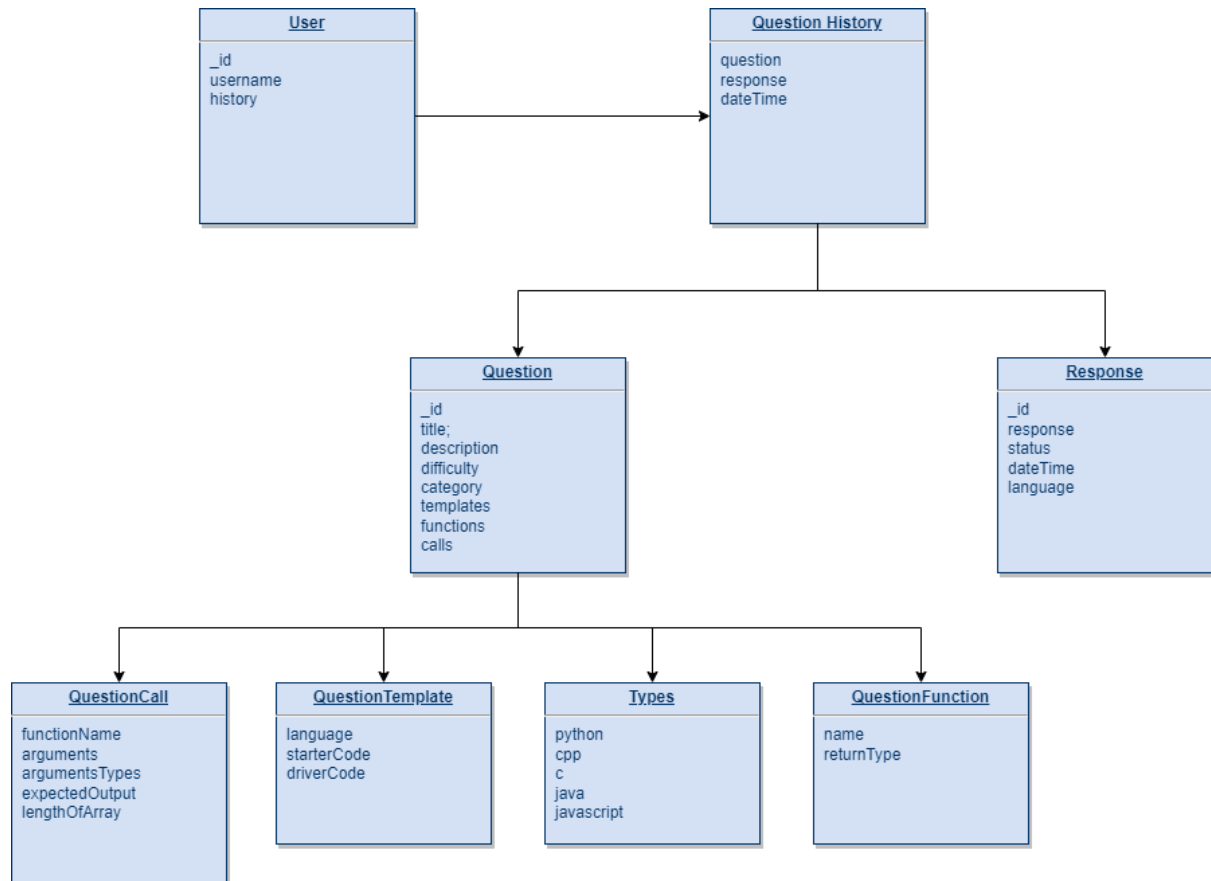


Figure 4.4.2.1a: Overview of schemas for question service

### 4.4.2.2 View all questions in Question Bank

Allowing users to view the questions in the Question Bank is an integral part of our application. In the Questions Page, users will be able to view a list of all the questions in the bank. Further details such as question templates can also be accessed via a pop-up. The preview of questions in the question bank gives users an idea of the questions which will be used in the mock interview.

### 4.4.2.3 Question Filtering

Category

Data Structures, Bit Manipulation

Difficulty

Medium, Easy

APPLY FILTERS

RESET FILTERS

Figure 4.4.2.3a: Screenshot of Question Filtering

Within the question page, users can use filters to query for topics of their interest. Our filters allow users to filter by categories and difficulties available in our question bank. With the large number of questions available in the question bank, users might not have time to scroll through all the questions to locate a practice question for their topic of choice. Therefore, filtering improves the user experience greatly to ensure more targeted practice for technical interviews. Users can also utilize the difficulty filter to compare the complexity of different questions within the same topic. This allows them to accurately gauge their skill level and attempt questions more suited to their technical abilities.

Upon entering the Questions Page, the filters will be shown above the Question Bank table. There will be 2 selection boxes, one for Category and one for Difficulty. Users can select their categories and difficulties of choice by opening the selection box of the respective filters. Users can select more than one option if they desire. If users want to reset their choices and re-select the filters from the start, they can click on the 'Reset Filters' button to clear their selections, then click 'Apply Filters' to get back the whole Question Bank. After choosing the topics and difficulties they want, users can click on the 'Apply Filter' button to filter out the questions. If users do not want any filters for category/difficulty, they can leave the selection box blank.

When the 'Apply Filter' button is clicked, a 'POST' request is sent to the MongoDB database containing the Question Bank through the api gateway. Details in the request include the topics and difficulties selected by the user stored in arrays of the respective filters. If no filters are chosen, the database service will respond by posting all available questions in the database. Otherwise, the database service will query the Question Bank for questions which match the corresponding filters and post all questions meeting the criteria. The questions posted will be reflected on the Questions Page.

#### 4.4.2.4 Question History

To encapsulate the PeerPrep interview process, we added session and response entities. The session object represents a single mock interview, composed of multiple users and multiple responses. The session object contains the following fields:

- createdAt
- difficulty
- language
- roomId
- year
- month

Each session object contains information about the mock interview session language, time of creation, difficulty, and the socket room id. We designed the session to be created once a match was successfully found, and the details would be part of the request made by the matching service to the user-service to create the session record. As the frontend is unaware of the session creation, the socket room id field allows the frontend to create responses tied to this session.

The response records the history of each question attempted by the users. It also contains a question id to map it to the appropriate question. The response object contains the following fields:

- text
- questionId

When a user enters the profile page, the questions are fetched from the question service and question history, including sessions, users and responses, is fetched from the user service. The user is then able to view their previous interview together with the name of the user, date, language, difficulty, and number of questions completed.

The user is also able to view each response given during the mock interview. When the user clicks on “view”, the responses from that session are loaded. The user can then view each response to the matching question.



### 4.4.3 Matching service

Our matching microservice helps to match users together for a mock interview. Users will be matched based on the difficulty they have chosen. We used a queue as a 'waiting room' for users who are in progress of matching such that no data would be persisted after they have found a match. We have also implemented a time-out after 30 seconds to prevent congestion. The flow chart from the user perspective is as follows:

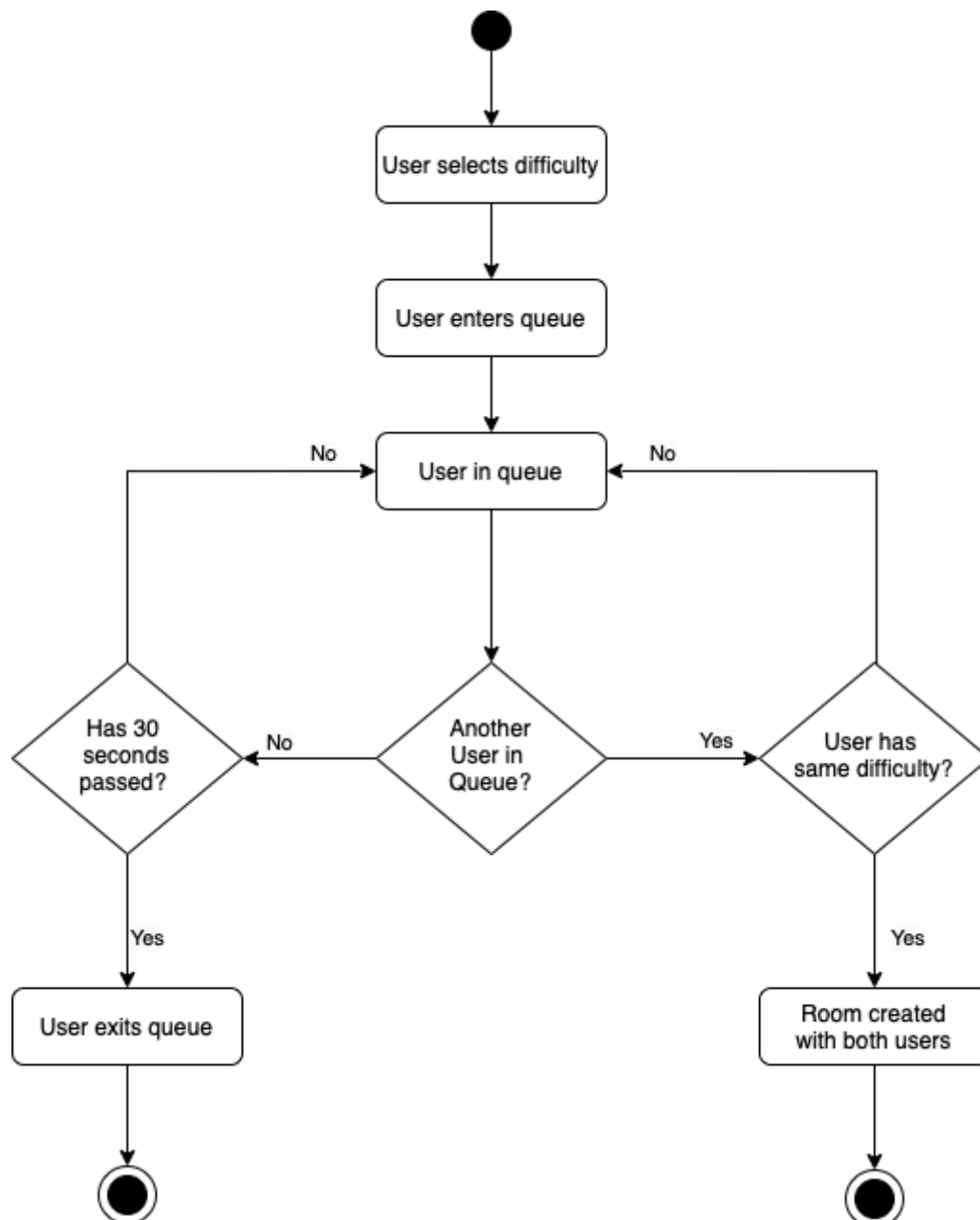


Figure 4.4.3a: Flow for matching event

A user will select their difficulty and language preference to begin matching by joining a queue. When a user joins a queue, a timer counting down from 30 seconds will be maintained by the frontend. After 30 seconds has passed and no match has been found, the

frontend sends a request to disconnect the socket. The service removes the user from the queue and disconnects the socket.

If there is a successful match, a room with a unique room ID will be created and the usernames of both users will be stored. Users can now collaborate and the submissions will be included in both user's question histories.

#### **4.4.4 Collaboration service**

As detailed in section [4.2.2 Pub-sub](#), our platform employs the Socket.IO API to facilitate real-time interaction between matched users. Upon being matched, users are subscribed to a common 'topic', which essentially represents the collaboration room they are in. This setup forms the core of our interactive environment.

Within these collaboration rooms, various user actions trigger real-time updates. For instance, when a user requests to move to the next question, makes edits in the code editor or initiates the stopwatch, these actions are published to their specific collaboration room topic. Consequently, the other user in the room, who is subscribed to the same topic, receives immediate notifications of these actions. This mechanism ensures that any change or interaction is promptly reflected on both users' interfaces.

This synchronous communication mode following the pub-sub pattern, powered by Socket.IO, is pivotal in creating a dynamic and responsive user experience. It allows for seamless, low-latency collaboration, mirroring the immediacy and interactivity of in-person coding sessions. Users can effectively code, discuss, and problem-solve in real-time, making the most out of their collaborative learning and interview preparation sessions.

#### 4.4.4.1 Code Editor

The code editor component supports the code collaboration functionality of PeerPrep. The component allows for code typed by one party to be seen and analyzed by the other party. The sharing functionality is enabled by the use of websockets, which introduces real-time communication capabilities, enabling multiple users to collaborate on code editing in a synchronized manner.

The websockets are managed by the collaboration service in our backend, which broadcasts changes made in the code editor to other sockets, such as if users type code or select some text on the editor.

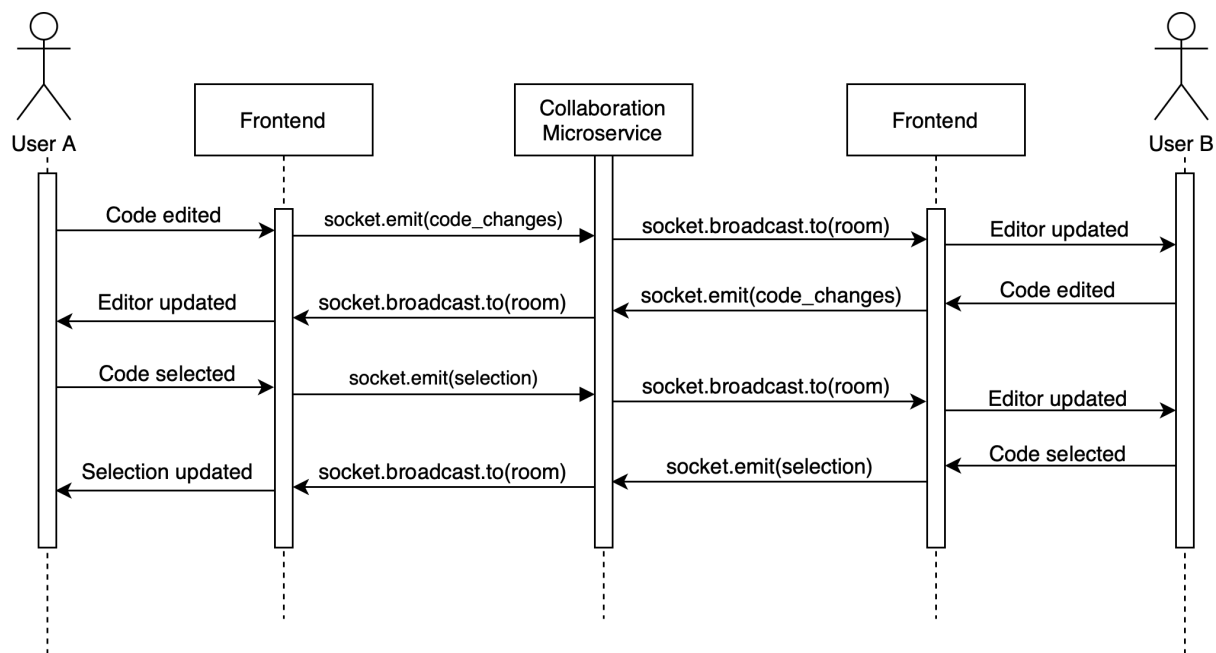


Figure 4.4.4.1a: Sequence diagram for code editor

Changes made by either user on the frontend of PeerPrep are then propagated to the other user, by websockets implemented in our Collaboration Service.

#### 4.4.4.2 Video/Audio communication

The Video communication component supports the real time communication functionality of PeerPrep. After the matched users join the room, they establish a connection with each other and communicate with each other via video and audio.

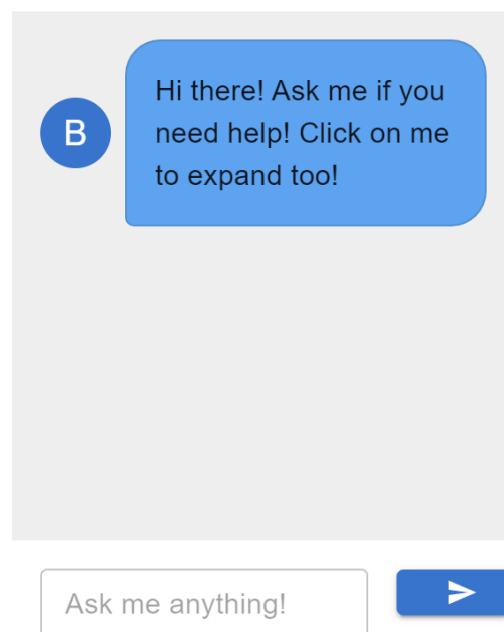
In our project we used PeerJS for an easier way of handling WebRTC (Web Real-Time Communication) peer-to-peer data, audio, and video calls. We used peerjs's API to send and receive video and audio data between the users in real time.

We chose not to implement a "text"/chat communication service as we felt that the collaboration service's code editor is more than sufficient to achieve that purpose. By utilizing comment blocks in the code editor can improve text-only discussions by commenting around code chunks that need to be discussed or simply having unimportant

short conversations that can be deleted after and need not be saved. Only important points can be noted down in comment blocks and be saved into our responses that can be viewed in our history.

#### 4.4.4.3 Interviewer/interviewee roles

With respect to the roles both users have namely Interviewer role and interviewee role, we wish to give some assistance to the users if they have trouble solving the questions. This interviewer specific panel together with [4.4.7 AI Chatbot service](#) aims to provide AI assistance to the interviewer so that he is prepared to give guidance to the interviewee. The interviewer also has a notepad in the panel to jot down any notes along the way. Moreover even after the technical aspect of the interview is done, interviewers can query for commonly asked non-technical interview questions to simulate a real-life interview giving both users some sort of confidence after the session.



*Figure 4.4.4.3a: Screen shot of AI chatbot assistance*

#### 4.4.4.4 Other Features

like the next question handshake, end session handshake, and the stopwatch also follow the pub-sub pattern using websockets.

Other Collaboration room events			
Feature	Event name	Triggered by	Publisher
Next question handshake	openNextQuestionPrompt	When a user clicks on the 'Next Question' button	Frontend
	aUserHasAcceptedNextQuestionPrompt	When a user clicks on the 'Accept' button for the next question handshake	Frontend
	aUserHasRejectedNextQuestionPrompt	When a user clicks on the 'Reject' button for the next question handshake	Frontend
	proceedWithNextQuestion	When all users in the room have pressed 'Accept'	Collaboration service
	dontProceedWithNextQuestion	When any user in the room have pressed 'Reject'	Collaboration service
End Session handshake	openEndSessionPrompt	When a user clicks on the 'End Session' button	Frontend
	aUserHasAcceptedEndSessionPrompt	When a user clicks on the 'Accept' button for the end session handshake	Frontend
	aUserHasRejectedEndSessionPrompt	When a user clicks on the 'Reject' button for the end session handshake	Frontend
	proceedWithEndSession	When all users in the room have pressed 'Accept'	Collaboration service
	dontProceedWithEndSession	When any user in the room have pressed 'Reject'	Collaboration service
Stopwatch	stopwatch_start_request	When a user presses the 'start' icon in the stopwatch	Frontend
	stopwatch_stop_request	When a user presses the 'stop' icon in the stopwatch	Frontend
	stopwatch_reset_request	When a user presses the 'reset' icon in the stopwatch	Frontend
	start_stopwatch	When a user presses the 'start' icon in the stopwatch	Collaboration service
	stop_stopwatch	When a user presses the 'stop' icon in the stopwatch	Collaboration service
	reset_stopwatch	When a user presses the 'reset' icon in the	Collaboration

		stopwatch	service
--	--	-----------	---------

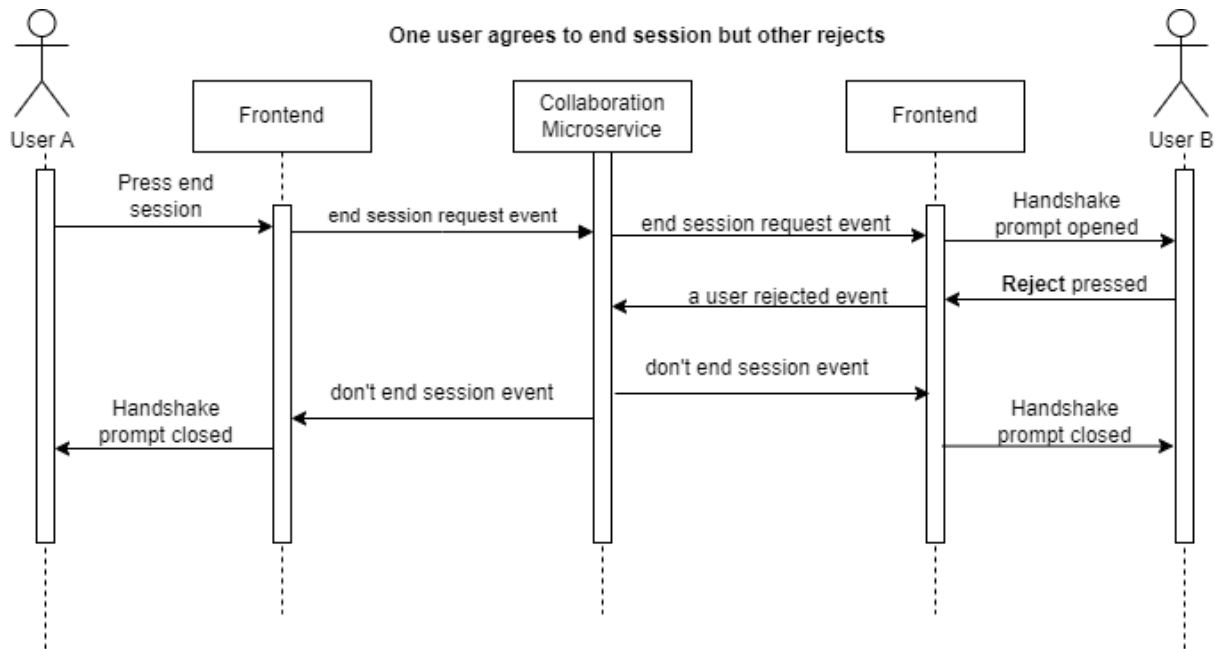


Figure 4.4.4.4a: Sequence diagram for moving on to next question

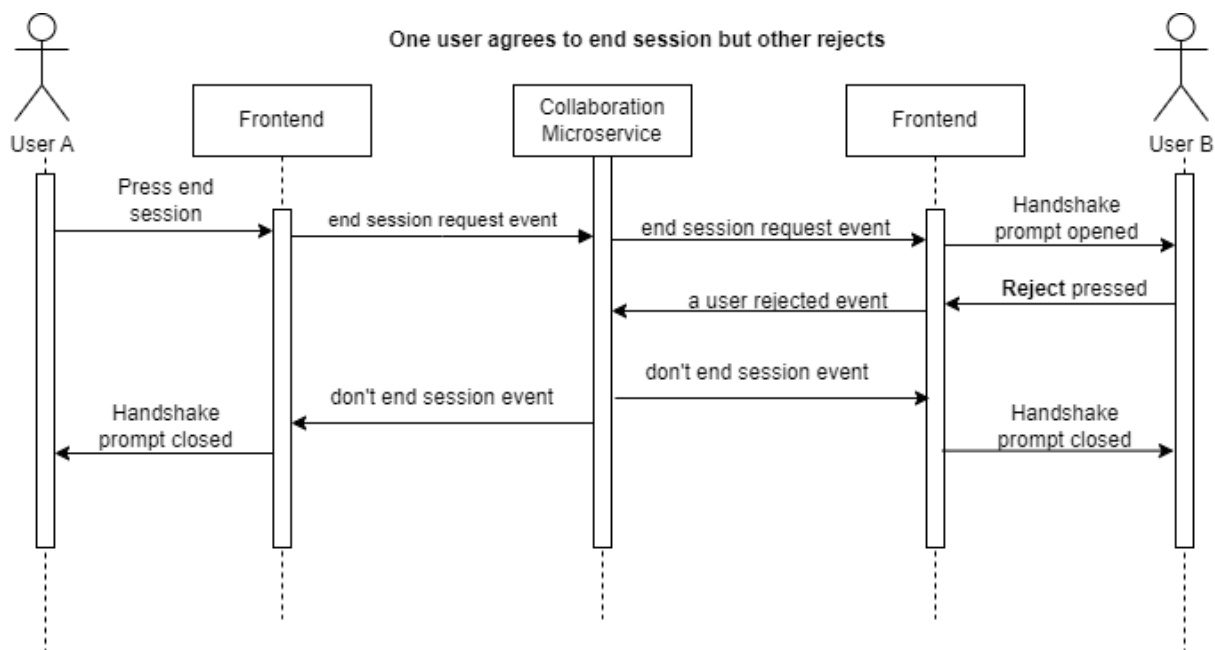
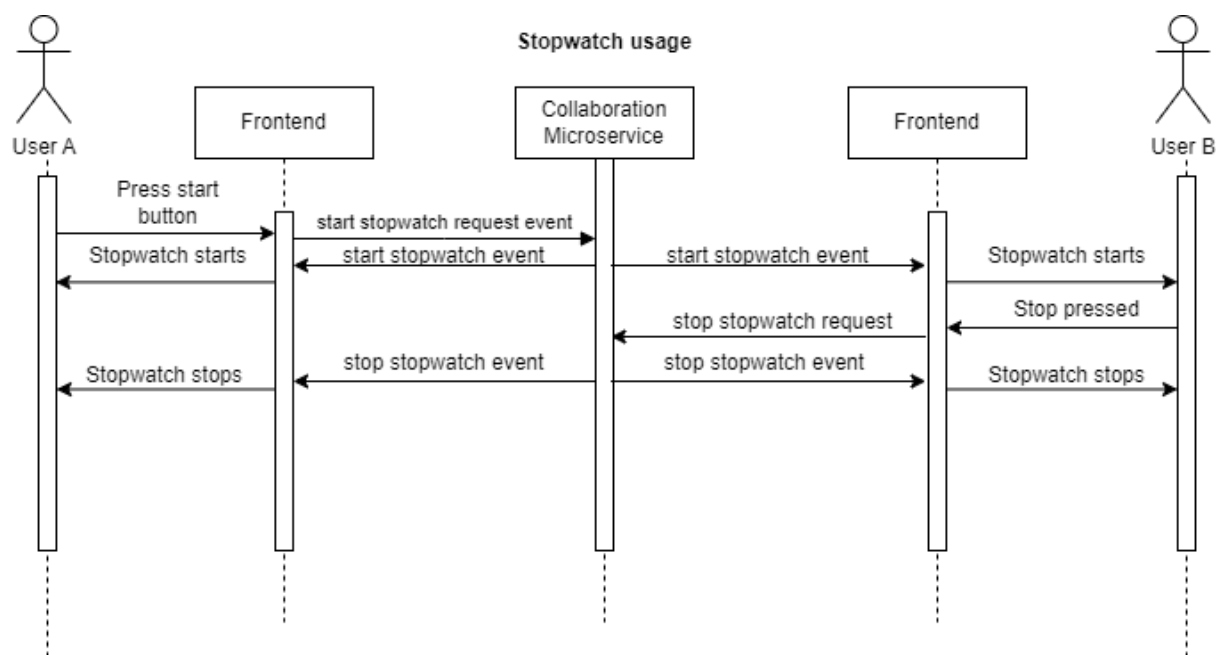


Figure 4.4.4.4b: Sequence diagram for ending session



*Figure 4.4.4.4c: Sequence diagram for stopwatch function*

### 4.4.5 Email service

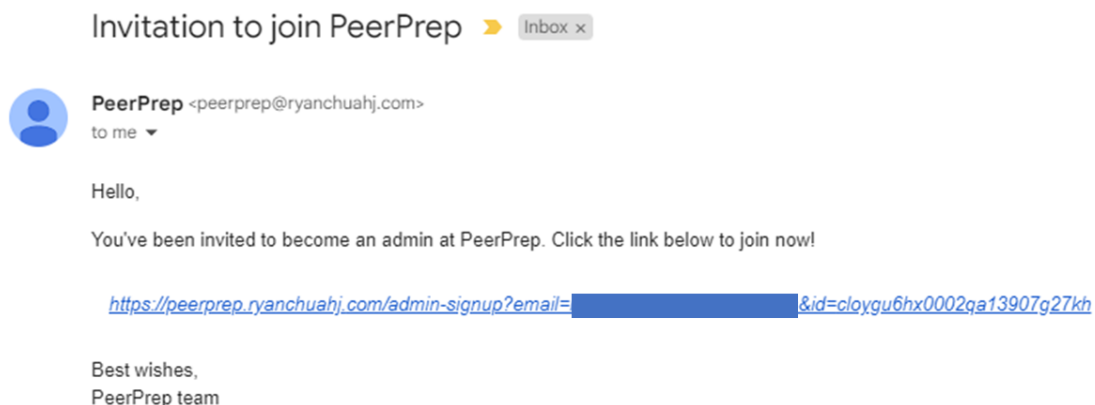
Our user service is enhanced by an email service that enables sending invitations for new admins. An email service was chosen as a feasible idea to enable direct outreach to new admins, and secure the signing up process away from regular users.

#### 4.4.5.1 Technologies used

The Email service uses Nodemailer in conjunction with Forward Email to enable the creation and sending of emails from our custom domain. Our DNS MX record is hosted on Amazon Route 53, using our custom domain, [peerprep.ryanchuahj.com](https://peerprep.ryanchuahj.com). A brief description of these technologies have been provided in section [4.3.1 Third-Party Libraries](#).

#### 4.4.5.2 How it works

The Email service has a single POST endpoint to receive email requests. Currently, only the User service calls the Email service to send an email when a new invitation is created (See section [4.4.1 User service](#)). This endpoint creates a nodemailer transport, which forwards the email to “smtp.forwardemail.net” on port 465 using our credentials. A custom invite link is generated using the invitee’s email and the id of the created invitation. The invitation email uses an existing template and simply substitutes the value of the invite link.



*Figure 4.4.5.2a: Example of invitation to join PeerPrep as admin*

The nodemailer then forwards the email to Forward Email, which has been configured for our custom domain and outbound SMTP. This has been achieved through adding MX, DKIM, DMARC and Return Path DNS records in Amazon Route 53. Forward Email is then able to send the email to the desired recipient.

#### 4.4.5.3 Security Measures

Nodemailer transport ensures that the communication between the Email Service and the service provider (smtp.forwardemail.net) is encrypted. Additionally, sensitive information like credentials are stored securely using environment variables. The encryption and security issues of sending a link within an email is addressed in section [4.4.1.4 Admin Invitations](#).



#### 4.4.6 Compiler service

Our collaboration platform supports code compilation and execution for multiple programming languages, including Python (3.8), C (GCC 9.2.0), C++ (GCC 9.2.0), Java (OpenJDK 13.0.1), and JavaScript (Node.js 12.14.0). To facilitate this, we've integrated Judge0, a powerful, open-source online code execution system. Judge0 is an optimal choice for several reasons:

**Sandboxed Execution:** It ensures the safe and isolated execution of code, which is essential for maintaining the integrity and security of our platform.

**Support for Multiple Languages:** Judge0 can handle over 60 programming languages, making it a versatile tool that aligns with our diverse coding requirements.

**Detailed Execution Results:** Judge0 offers extensive feedback on code execution, which is integral to the PeerPrep experience. This comprehensive feedback includes crucial error identification and performance insights, such as execution time and resource usage.

**Deployability:** Judge0's architecture is designed for ease of deployment. It utilizes Docker containers, which are lightweight, portable, and consistent across different environments. This means that Judge0 can be set up quickly and efficiently on various systems without compatibility issues.

**Scalability:** The system is built to scale with demand. Using Docker, Judge0 can manage increased load by adding more containers, allowing for flexible resource management. This scalability ensures that our platform can handle a high volume of concurrent code executions without performance degradation.

By using Judge0, we ensure a reliable and efficient coding environment for our users, accommodating a wide range of programming needs.

Moreover, we enhanced the utility of the compiler service by allowing the addition of custom test cases to questions. This feature is designed to maximize the effectiveness of code execution and assessment. For each programming question, there is an option to include a specific driver code tailored to each language. This driver code, which is manually curated by the administrators, enables the platform to display failed test cases immediately after the code execution, providing direct and relevant feedback to the users.

While the compiler service is equipped with the capability to automatically generate basic test cases—derived from a predefined set of function calls and expected outputs—this functionality was not integrated into PeerPrep. The decision against including automatic test case generation was based on the diverse and complex nature of real-world coding questions. Particularly in more intricate programming languages like C, the creation of test cases involves multifaceted elements such as complex array declarations, and specification of argument types and return values for each function call. The requirement for such detailed input from question setters was deemed overly complicated for the scope of PeerPrep.

This careful consideration ensures that PeerPrep remains user-friendly while still offering robust and relevant tools for coding interview practice, tailored to the diverse needs and skill levels of its users.

```

    it('should compile C code involving 1d arrays successfully', async () => {
        const language = LANGUAGE.C;
        const source_code = `
#include <stddef.h>

int* foo(int* x, size_t size) {
    for (size_t i = 0; i < size; i++) {
        x[i] += 1;
    }
    return x;
}
`;

        const calls = [
            {
                functionName: "foo",
                arguments: ["[1,2,3]", "3"],
                argumentsTypes: [
                    {
                        python: "List[int]",
                        c: "int*",
                        cpp: "std::vector<int>",
                        java: "int[]",
                        javascript: "number[]"
                    },
                    {
                        python: "int",
                        c: "size_t",
                        cpp: "size_t",
                        java: "int",
                        javascript: "number"
                    }
                ],
                expectedOutput: "[2,3,4]",
                lengthOfArray: [3]
            }
        ];

        const functions = [
            {
                name: "foo",
                returnType: {
                    python: "List[int]",
                    c: "int*",
                    cpp: "std::vector<int>",
                    java: "int[]",
                    javascript: "number[]"
                }
            }
        ];

        const result = await compileCode(language, source_code, calls, functions, null);
    });

```

Figure 4.4.6a: Screenshot of Arguments provided to automatically generate test cases in C lang involving a 1-dimensional array

```

#include <stdlib.h>
#include <stdbool.h>

// Compares two 1D arrays
bool compare1DArrays(int *arr1, int *arr2, int size) {
    for (int i = 0; i < size; i++) {
        if (arr1[i] != arr2[i]) return false;
    }
    return true;
}

// Compares two 2D arrays
bool compare2DArrays(int **arr1, int **arr2, int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        if (!compare1DArrays(arr1[i], arr2[i], cols)) return false;
    }
    return true;
}

// --- USER CODE START ---

#include <stddef.h>

int* foo(int* x, size_t size) {
    for (size_t i = 0; i < size; i++) {
        x[i] += 1;
    }
    return x;
}

// --- USER CODE END ---

void run_tests() {
    // --- TESTS START ---
    int input_1_1[3] = {1,2,3};
    int* result1 = foo(input_1_1, 3);
    int expected1[3] = {2,3,4};
    if (!compare1DArrays(result1, expected1, 3)) {
        fprintf(stderr, "AssertionError: Test 1: Expected {2,3,4}, but got a different result");
        exit(1); // Exit on failure
    }

    // --- TESTS END ---
}

int main() {
    run_tests();
    return 0;
}

{
  data: {
    stdout: null,
    status_id: 4,
    time: '0.001',
    memory: 1064,
    stderr: null,
    compile_output: null
  },
  error: false,
  message: 'Judge0 responded',
  statusCode: 200,
  firstFailedTestCaseNumber: null
}

✓ should compile C code involving 1d arrays successfully (3047ms)

```

Figure 4.4.6b: Screenshot of Automatic generation of driver code for test cases in C lang involving a 1-dimensional array

The provided diagram illustrates the integration of Judge0 into the PeerPrep platform, detailing the process of code execution requests.

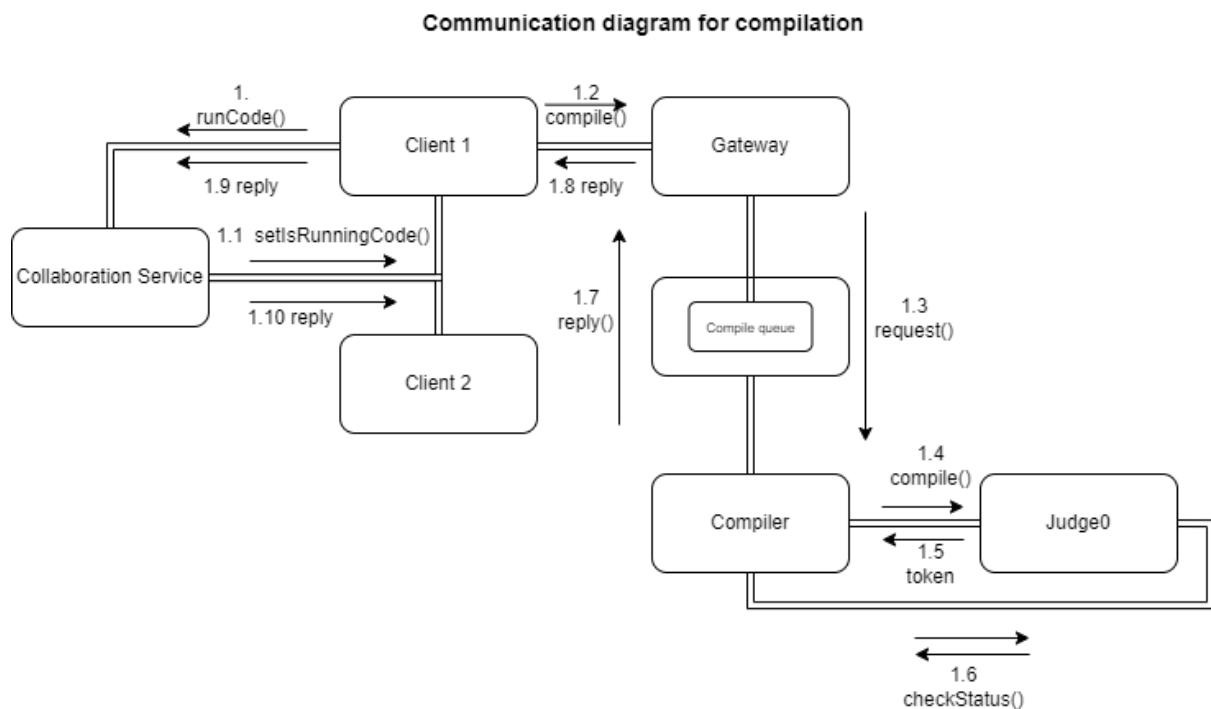


Figure 4.4.6c: Communication diagram for compilation

Here's an enhanced description of the workflow:

When a user in the collaboration room initiates a code execution request, our collaboration service immediately synchronizes this action and notifies the other user in the session. Both users then await the execution results. The request includes the contents of the code editor, the driver code for the current question, and the programming language being used in the session. This information is sent to our gateway.

Upon receiving the request, our gateway places it into the RabbitMQ compilation queue. This queuing system plays a critical role in managing the flow of requests. Our compilation service, constantly monitoring this queue, picks up the request and forwards it to our Judge0 instance. Judge0 then processes the request and provides our compilation service with a unique token associated with that specific request.

The compilation service uses this token to periodically check the status of the request with Judge0. This approach ensures that the process is not held up while waiting for a response. Once Judge0 has completed the compilation and execution of the request, it sends the results back to the compilation service. These results are then relayed through a pseudo queue in RabbitMQ to our gateway, which in turn delivers them to the clients.

This intermediate token workflow, periodic status checks, and usage of RabbitMQ queues render the compilation and execution procedure largely asynchronous. It allows our compilation service to efficiently handle multiple requests, processing new ones while

awaiting results from Judge0 for others. This system significantly enhances the responsiveness and efficiency of the code execution feature on PeerPrep, providing a seamless experience for users practicing coding interviews. Moreover, as mentioned in section 4.2.1 Request/ reply, the usage of RabbitMQ provides benefits like load-balancing, which is critical in a high-traffic service like compilation.

#### 4.4.7 AI Chatbot service

Our AI Chatbot service, leveraging the capabilities of OpenAI's GPT-3.5-turbo model, employs a similar architectural pattern to our compilation service, utilizing a request/reply mechanism with RabbitMQ acting as the message broker.

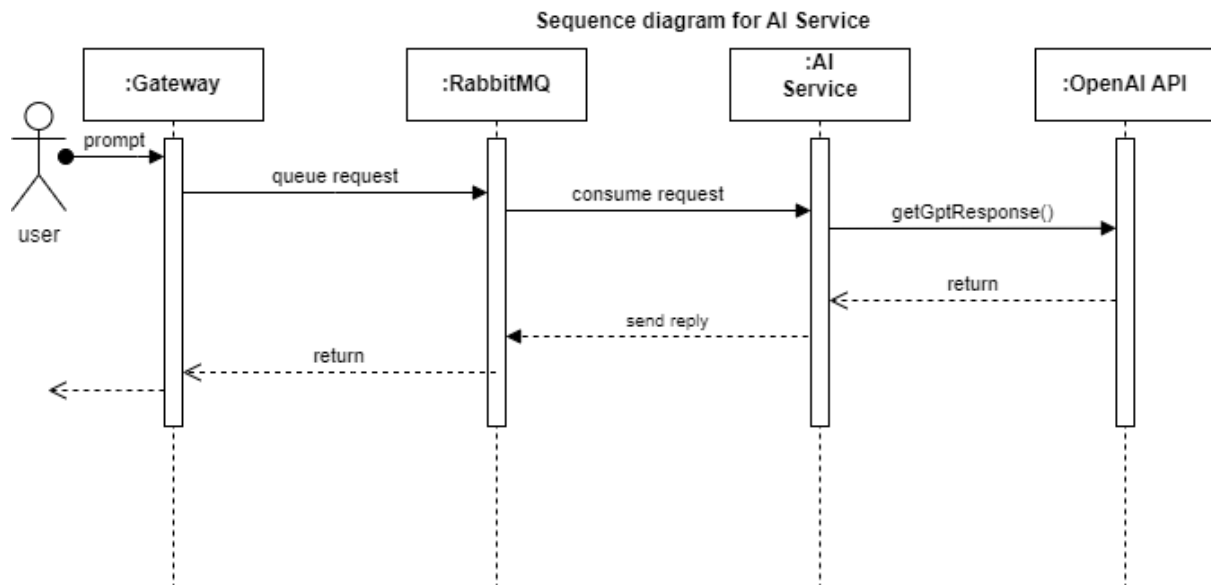


Figure 4.4.7a: Sequence diagram for querying AI ChatBot

The process begins when a user submits a prompt through our platform. This prompt first reaches our gateway, which then forwards it to the AI Service queue within RabbitMQ. The AI Service, constantly monitoring this queue, consumes the prompts and processes them using the GPT-3.5-turbo model. Once the response is generated, it is sent back to the gateway via RabbitMQ, which then delivers it to the user.

As elaborated in section 4.2.1 'Request/Reply', the implementation of RabbitMQ brings significant advantages, particularly in terms of load balancing. This aspect is essential for efficiently managing the high volume of prompts our AI Service receives. RabbitMQ's ability to evenly distribute the workload across various AI Service nodes is a key factor in preventing any single node from becoming overburdened. This distribution ensures that our system maintains optimal performance, even during periods of intense traffic, thus guaranteeing a consistent and reliable user experience.

## 4.5 Frontend

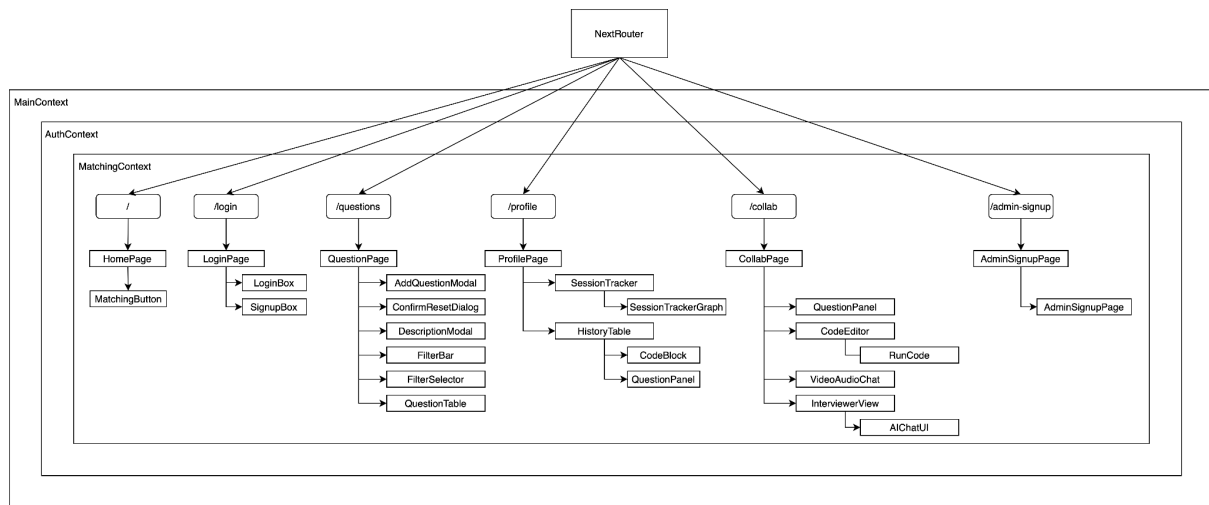


Figure 4.5: Frontend React component tree

The high level structure of our frontend, built with React and NextJS, is seen above.

### 4.5.1 React's Context-Provider

React's Context API provides a way to share values like themes, authentication status, or any global state information between components without explicitly passing them through each level of the component tree. The Context-Provider component is particularly useful when implementing prop-drilling becomes impractical or cumbersome, especially in the context of PeerPrep when the sheer number of states increases exponentially with additional features added.

Our implementation of PeerPrep utilizes three levels of Contexts, namely:

1. MainContext
2. AuthContext
3. MatchingContext

MainContext is an overall “container” context that wraps around AuthContext and MatchingContext.

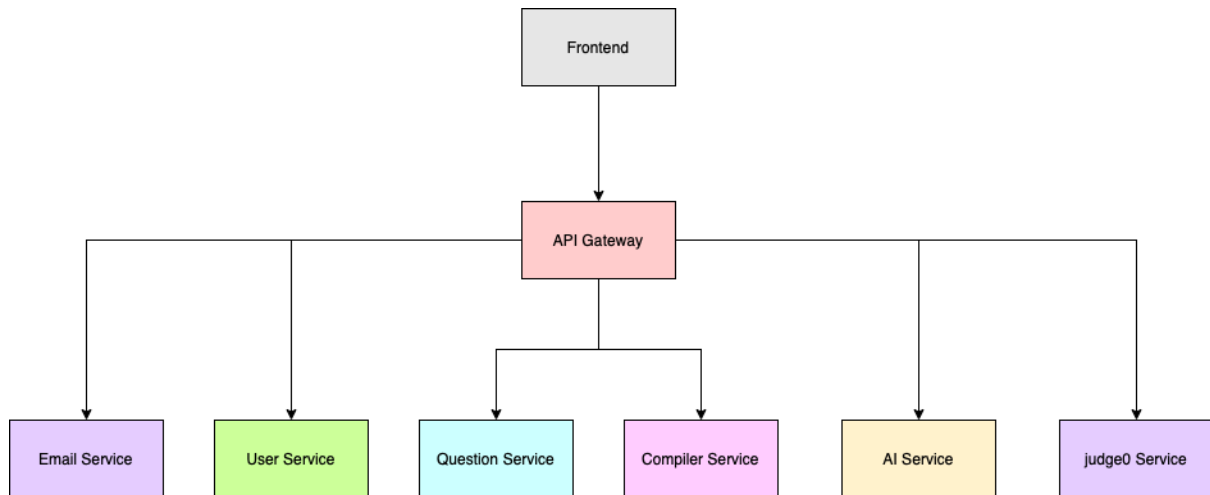
AuthContext is a context provider which provides authentication, authorization and user-service related information regarding the user to the rest of the web application, such as the user's username, email and admin status.

MatchingContext is a context provider which provides matching-service related information to the rest of the web application, such as the matched user's socketId, userId and roomId.

## 4.6 Gateway

To route traffic between the services, we implemented a simple api gateway using express-gateway.

We utilized the Facade pattern to implement the gateway, with the gateway acting as the single point of entry for all the backend services except for the matching service and collaboration service. The Facade pattern is particularly useful as it decouples the frontend from the backend services.



*Figure 4.6: Overview of API Gateway*

With this gateway, we can make isolated changes to the backend services code base without any changes to the frontend code. The gateway intercepts http requests from the frontend and forwards them to the respective services with a modified URL. The services then craft a response and replies to the frontend through the API Gateway.



## 4.7 Deployment

Peerprep's frontend is deployed on Vercel and microservices are deployed on Amazon Web Services (AWS).

### 4.7.1 Frontend Deployment

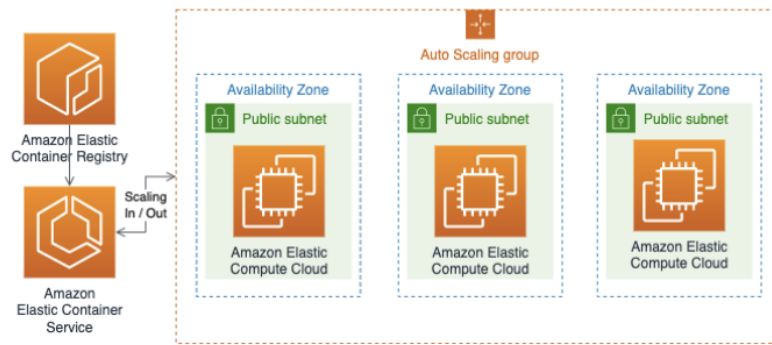
[Vercel](#) is a Frontend Cloud provider that provides the developer experience and infrastructure to build, scale, and secure a faster, more personalized Web. We chose Vercel for our frontend hosting as it was simple and easy to host our frontend repository. Vercel provides Continuous Deployment for our frontend as it automatically builds and deploys straight from our repository. It is able to provide generated domain names and urls for projects, which we utilized in the early stages of development. As we moved to HTTPS and bought a domain, we were able to host our frontend at our custom domain on Vercel. Also, we configured the necessary environmental variables for the frontend, such as the urls of our application load balancer, which we will cover in the next section.

### 4.7.2 Microservices Deployment

For our microservices, we chose [AWS](#) as AWS has the largest market share of all cloud computing services, making it more reliable and more valuable for our learning. Here is a list of all the services used:

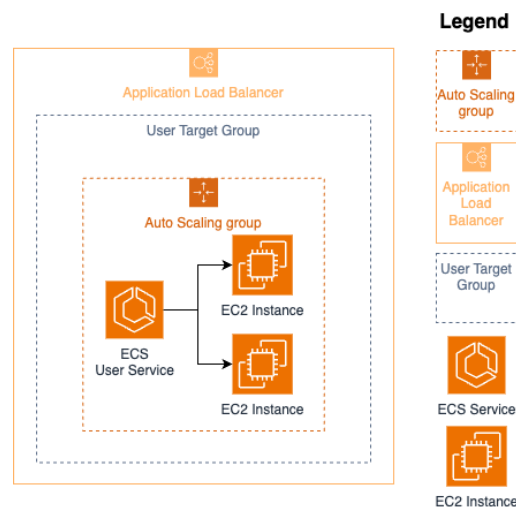
- Elastic Compute Cloud (EC2)
- Elastic Container Registry (ECR)
- Elastic Container Service (ECS)
- Message Queue (MQ)
- Route 53
- Certificate Manager

While there are many deployment strategies on AWS, we chose to deploy on ECS with EC2 instances as we felt like it gave us the desired amount of control over the services. We did not opt for Elastic Kubernetes Service as we did not require such fine control over container orchestration. Amazon Route 53 and Certificate Manager have been used in conjunction to allow for HTTPS for our custom domain.



*Figure 4.7.2a: Overview of deployment strategy*

Shown in the diagram above, referenced from [AWS blogs](#), is a brief overview of our deployment strategy. We were able to effectively automate and load balance our deployment through ECS services and tasks, EC2 Auto Scaling Groups (ASG) and EC2 Application Load Balancers (ALB). We store our container images in ECR, which ECS then retrieves through each individual service within our cluster. Each ECS service is configured to represent each microservice, to allow for individual scaling and management. ECS services are then able to scale in and out EC2 instances using EC2 ASGs set as ECS cluster infrastructure.



*Figure 4.7.2b: Example of User Service in relation to ALB*

EC2 ALBs are used with target groups to EC2 ASGs in order to positively identify each new EC2 instance created. When an ASG starts a new instance, we can deploy the selected instance to a target group, which then identifies each service. For example, The user service in ECS will trigger a new deployment of a user service task. The user service ASG will then spin up a new EC2 instance, which will be added to the user-service target group. The ALB which this target group is attached to is then capable of routing requests to the EC2 instance via port filtering. This diagram illustrates one service out of all the services attached to the ALB.

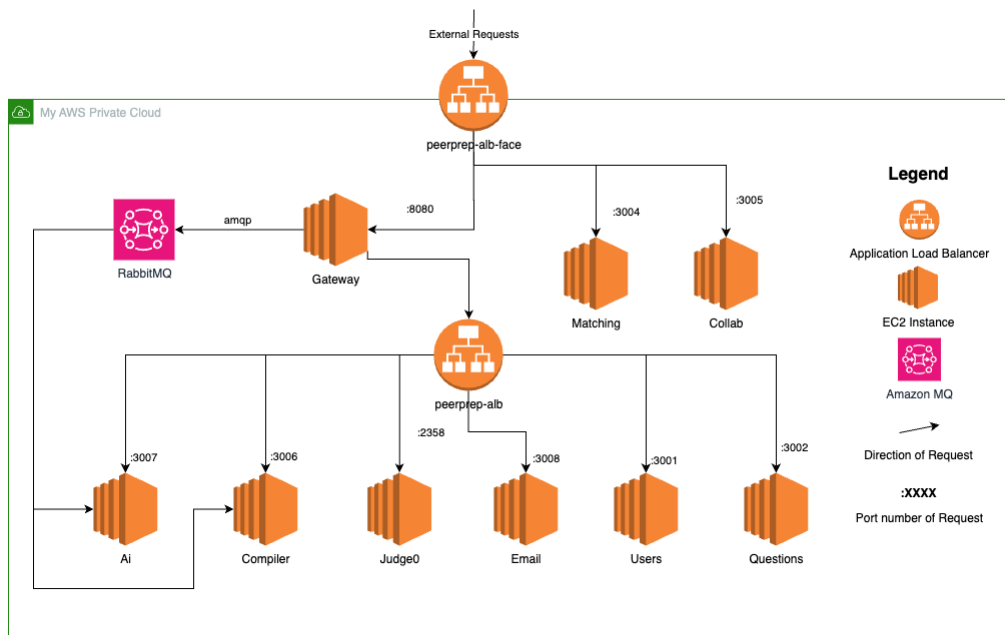


Figure 4.7.2c: Actual overview of deployment

Above is our deployment configuration for AWS. There are 5 main points of interest, which are the internet facing ALB, gateway, Amazon MQ instance, internal ALB, and the EC2 instances.

The internet facing ALB, peerprep-alb-face at the top of the diagram, is exposed to external HTTPS requests. We were able to achieve HTTPS support using a TLS certificate from Amazon Certificate Manager. This allows for HTTPS requests to be decrypted at this ALB and routed to each individual service.

The gateway is an EC2 instance which routes requests based on the request path to the other services. This gateway is a custom express gateway, which is covered in section [4.6 Gateway](#). This was our incomplete attempt at a custom API gateway, as seen from the matching and collaboration services not being behind the gateway. The gateway utilizes middleware to route requests to RabbitMQ.

The Amazon MQ instance uses RabbitMQ, to pass the request to the AI and Compiler service.

The internal ALB, peerprep-alb, receives requests from the gateway and routes the request to the corresponding EC2 instances in the target groups, as mentioned previously.

Each of our services receive requests at a specific port number. 3001 for user service, 3002 for question service etc. We are able to utilize the ALB configuration above to route HTTP requests to each service based on port number. There is one EC2 instance each for each microservice, with the addition of the Judge0 EC2 instance to support the compiler. Note that while the requests in the diagram are displaying the flow towards the instance, each EC2 instance is able to make HTTP requests to the ALB to reach different services.

## 4.8 CI/CD

We chose to implement our Continuous Integration (CI) Continuous Deployment (CD) using Github Actions, due to simplicity and convenience of having direct access to our repository.

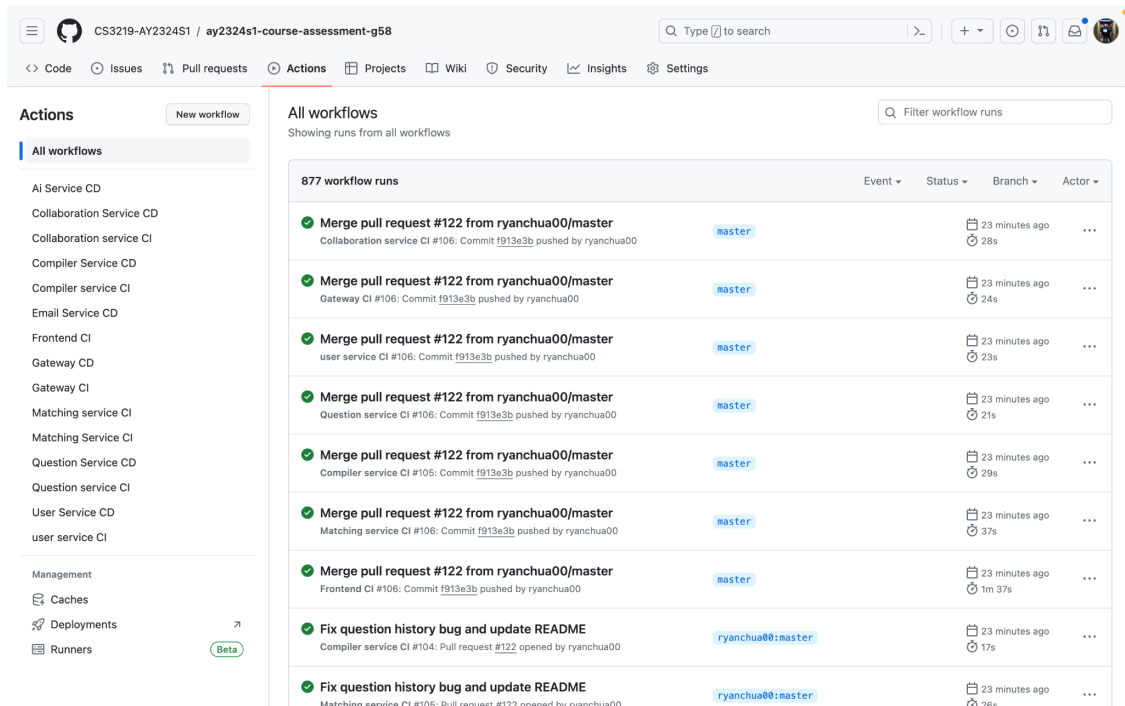


Figure 4.8a: Screenshot of Github Actions for team repository

We have a simple CI for our services, frontend and gateway on Github, that installs dependencies, creates a new build and runs tests. The CI is triggered upon pull request and push of the master and staging branch. This is purposefully done so that the code that is in the pull request is run against the tests first, so that if there are any issues, it can be rectified before the pull request is accepted.

For CD, we have a direct deployment of our services to AWS upon every successful push to master or staging branch. We trigger a docker compose of the services, deploy all newly composed services to ECR, and trigger a fresh deployment of the ECS service. The ECS service deployment waits for service stability before swapping out the previous EC2 instance, ensuring uptime on our applications. This CD is triggered on push for master and staging branches. We do not trigger the deployment when a new pull request is made to avoid disruptions to our current deployment while the pull request is not reviewed.

Note that one limitation is that since the dockerfiles are made for docker-compose, all the dockerfiles in each folder have a file directory from the root. As such, building images individually would not work due to a directory mismatch. e.g in the user service dockerfile, "COPY ./backend/user-service ." is run instead of "COPY . .". One potential workaround is having a separate dockerfile for deployments and builds. This limitation necessitates a fresh build of all services multiple times in each deployment.

## 5. Reflections

### 5.1 Further Enhancements

#### 5.1.1 Integrating third-party authentication services

Currently, our implementation of PeerPrep only supports users logging in with their email and password after they have created an account on our own password-based user service. While our solution enables basic security in order to protect our application data and services, it would be good to consider and implement alternative third-party authentication services, such as Firebase Authentication or GitHub OAuth, both of which provide industry-level authentication solutions to web applications. In addition, logging in via social media accounts can also be considered to provide a wider choice of authentication methods for the users to choose from.

#### 5.1.2 CI with Comprehensive Tests

Currently our CI pipeline is currently only being used to enforce node modules and package-lock are updated with package.json in our codebase and correctness of code. What we wish we could have done was to take a more test-driven approach where possible. This would ensure that changes to existing code such as optimisation would still give the same results.

#### 5.1.3 Enhanced Collaboration features

To better assist matched users in communicating and visualizing their ideas, having a drawing/whiteboard feature with pre-built elements would be especially helpful in tackling different questions for example those which involve graphs (DFS, BFS, Dijkstra). In some cases, words are not enough to communicate ideas. Thus, having a drawing feature or visual feature would definitely be beneficial. This could be implemented by integrating third party libraries into the collaboration service or adding a completely new drawing microservice.

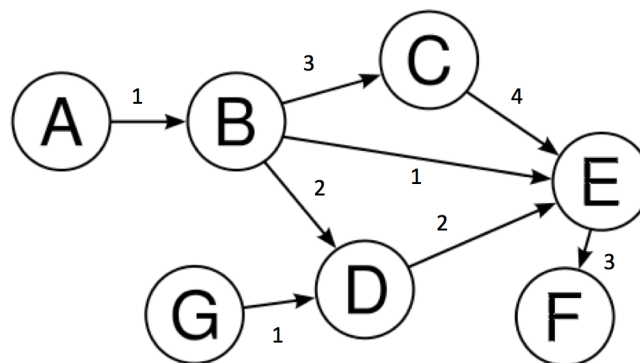


Figure 5.1.3.1a: Graph drawn using tools for finding Shortest Path in graph

## 5.2 Reflections and Learning Points

Overall this project has given us the opportunity to better ourselves as software engineers and improve both our technical as well as non-technical skills.

To start off, it exposed us to the entire software engineering cycle and web development workflow, from planning to coding for the frontend and backend frameworks and finally deploying our code. As we have the flexibility to decide on the tech stack and software design patterns we choose to implement, we have to weigh the pros and cons of each of them and choose the one that fits us the most.

### 5.2.1 Pros and cons of Microservice Architecture

The greatest benefit to having a microservice architecture is that the development process can become highly parallelizable. As different functionalities can be split into the different microservices and be worked on separately concurrently without worrying about the dependencies it might have with the other services. This is due to the microservice architecture following the Single Responsibility Principle inherently, which greatly reduces the coupling between different modules and services. Moreover, this benefit also allows easier detection of bugs as they are usually contained within the specific microservice and are easy to fix after.

However, despite its advantages, we acknowledge the disadvantages of a microservice architecture, especially when compared to other architectures like monolithic architecture. The complexity in implementing the microservice architecture proves to be much higher as coordinating the different services to run seamlessly through the implementation of communication services is an extra step to communicate between services and attain data consistency among them all. While the benefits are substantial, the challenges emphasize the importance of careful planning and consideration when opting for a microservice architecture.

### 5.2.2 Importance of planning and well-defined requirements document

Having all our backlogs at the start, and having a sprint schedule for each part really helped us to pace ourselves appropriately throughout the project. Also, producing our scope and specification requirements early in the project development cycle served as an important reference 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. This allowed us to focus on the development of the product without much backtracking as we simply worked to attain what our initial goals were although some changes were made to further optimize or improve the user friendliness of our web app.