# CS3219 Project Report

## AY22/23 Semester 1

## Group 27

# PeerPrep

**GitHub Repo:** https://github.com/CS3219-AY2223S1/cs3219-project-ay2223s1-g27
**Deployment:** https://peerprep.one/

| Team Member | Student No. | Email |
|---|---|---|
| Amanda Ang Yee Min | A0205620W | e0425543@u.nus.edu |
| Ng Chi Sern | A0219866M | e0550524@u.nus.edu |
| Ng Yong Xiang | A0196443X | e0388929@u.nus.edu |
| Ng Zhen Teng | A0199633N | e0406614@u.nus.edu |

# Table of Contents

## 1. Background & Purpose

Increasingly, students face challenging technical interviews when applying for jobs which many have difficulty dealing with. Issues range from a lack of communication skills to articulate their thought process out loud to an outright inability to understand and solve the given problem. Moreover, grinding practice questions can be tedious and monotonous. To solve this issue, our team came up with an interview preparation platform that comes with a peer matching system.

This is a web application called PeerPrep, where students can find peers to practice whiteboard-style interview questions together, hence better prepare themselves for technical interviews during their job hunt. While this can be achieved in many ways, the main objective of this application is to leverage a collaborative learning system where students can revise core algorithmic concepts with other students, hence learning from each other and breaking the monotony of revising alone.

## 2. Individual Contribution

| **Entity/Developer** | **Ng Yong Xiang** | **Ng Chi Sern** | **Amanda Ang Yee Min** | **Ng Zhen Teng** |
|---|---|---|---|---|
| **Deployment, CI/CD Pipeline** | Developer | Developer | | |
| **Frontend** | Developed cookie retrieval and refresh feature | Contributed to UI Improvements | Developer | Contributed to UI Improvements |
| **User Service** | Developer | | | Developer |
| **Matching Service** | | Developer | | |
| **Collaboration Service** | Developer | Developer | | |
| **Question Service** | Developer | Developer | | |
| **Communication Service** | | | | Developer |

# 3. Product Requirements

## 3.1 Functional Requirements

| User Context (FR1) | | | |
|---|---|---|---|
| **S/N** | **Functional Requirement** | **Priority** | **Must-have** |
| FR1.1 | The system should allow users to create an account with username and password. | High | ✔ |
| FR1.2 | The system should ensure that every account created has a unique username. | High | ✔ |
| FR1.3 | The system should allow users to log into their accounts by entering their username and password. | High | ✔ |
| FR1.4 | The system should maintain a logged in session until the user logs out. | High | ✔ |
| FR1.5 | The system should allow users to log out of their account. | High | ✔ |
| FR1.6 | The system should allow users to delete their account. | Medium | |
| FR1.7 | The system should allow users to change their password. | Medium | ✔ |
| FR1.8 | The system should allow users to reset their password using their email address. | Medium | |
| FR1.9 | The system should issue access and refresh JWT tokens when a user logs in successfully. | High | |
| FR1.10 | The system should blacklist refresh JWT tokens when a user logs out. | High | |

| Match Context (FR2) | | | |
|---|---|---|---|
| FR2.1 | The system should allow users to select the difficulty level of the questions they wish to attempt. | High | ✔ |
| FR2.2 | The system should allow users to change the chosen | Medium | |

| | | | |
|---|---|---|---|
| | difficulty level if a match is not yet found. | | |
| FR2.3 | The system should be able to match two waiting users with identical difficulty levels and put them in the same room. | High | ✔ |
| FR2.4 | If there is a valid match, the system should match the users within 30s. | High | ✔ |
| FR2.5 | The system should inform the users that no match is available if a match cannot be found within 30 seconds. | High | ✔ |
| FR2.6 | The system should allow users to leave the matchmaking queue. | High | |

| **Interview Question Context (FR3)** | | | |
|---|---|---|---|
| FR3.1 | The system should allow the Interviewer to send qualitative questions. | Medium | |
| FR3.2 | The system should provide questions of the right difficulty from a database of relevant coding questions. | High | ✔ |
| FR3.3 | Questions should include an “Examples” section to help students better understand questions. | Low | |
| FR3.4 | The system should provide a link for users to attempt questions from past attempts on an external coding practice website. | Low | |

| **Real-time Editor Context (FR4)** | | | |
|---|---|---|---|
| FR4.1 | The system should allow users to see other users’ edits in real-time. | High | ✔ |
| FR4.2 | The system should allow users to submit their code, compile and execute | Medium | |
| FR4.3 | The system should display the compilation output of  their code. | Medium | |
| FR4.4 | The system should notify either user if the other collaborator | Medium | |

| | | | |
|---|---|---|---|
| | in the session has disconnected and provide a link for the remaining user to continue on an external coding practice website. | | |

| **Communication Context (FR5)** | | | |
|---|---|---|---|
| FR5.1 | The system should assign users the role of Interviewer or Interviewee once they enter the coding room. | Medium | |
| FR5.2 | The system should allow the Interviewee to swap roles after he/she has completed his question. | Medium | |
| FR5.3 | The system should allow users to communicate via text and voice call or video call to train their communication skills. | Medium | |
| FR5.4 | The system should provide a list of interviewer questions for the Interviewer. | Medium | |

| **History Context (FR6)** | | | |
|---|---|---|---|
| FR5.1 | The system persistently stores the attempt upon browser tab close or clicking of finish button. | Low | |
| FR5.2 | The system should maintain a record of the user’s past attempted questions, including a record of the text editor contents and chat history. | Low | |

| **User Interface Context (FR6)** | | | |
|---|---|---|---|
| FR6.1 | The system should allow users to log in by interacting with the UI. | High | ✔ |
| FR6.2 | The system should allow users to log out by interacting with the UI. | High | ✔ |
| FR6.3 | The system should have a homepage for the user once logged in. | High | ✔ |
| FR6.4 | The system should allow users to leave the coding room | High | ✔ |

|  |  |  |  |
|---|---|---|---|
|  | during the interview |  |  |
| FR6.5 | The system should allow users to find a match by interacting with the UI. | High | ✔ |
| FR6.6 | The system should allow users to delete their accounts by interacting with the UI. | High |  |
| FR6.7 | The system should have a button for the users to leave the matching queue before the timer ends. | Low |  |
| FR6.8 | The system should provide a reset password page for users who have forgotten their passwords and wish to retrieve their accounts. | Medium |  |
| FR6.9 | The system should have a pop-up to notify the user if the other collaborator has disconnected from the session. | Medium |  |
| FR6.10 | The system should provide a collaborative text editor for users to edit their code. | High | ✔ |
| FR6.11 | The system should display the programming problem to the user. | High | ✔ |
| FR6.12 | The system should provide a real-time chat UI for two collaborators to communicate during the same session. | Medium |  |
| FR6.13 | The system should add role tags beside the usernames in the chat bot. | Low |  |
| FR6.14 | The system should allow the two collaborators to swap roles upon clicking a button. | Medium |  |
| FR6.15 | The system should allow users to access the history of previously attempted questions at the homepage. | Low |  |
| FR6.16 | The system should provide a dropdown of interview questions for the user with the "Interviewer" role. | Medium |  |

We will see in later sections that these contexts happen to correspond with the microservices developed by us. Each microservice corresponds to one or more contexts hence our product is able to address the above Functional Requirements.

## 3.2 Non-functional Requirements

| **S/N** | **Non-functional Requirement** | **Priority** | **Must-have** |
|---|---|---|---|
| **Security (NFR1)** | | | |
| NFR1.1 | Users' passwords should be hashed and salted before storing in the DB. | Medium | ✔ |
| NFR1.2 | JWT Access tokens issued by the system should expire reasonably quickly for system security. | Medium | |
| NFR1.3 | The service should only allow valid members of a room to enter a room. | Medium | |
| **Availability (NFR2)** | | | |
| NFR2.1 | The service should perform without failure in 95% percent of use cases in a month. | High | |
| NFR2.2 | The service must be available 98% percent of the time every month. | High | |
| **Performance (NFR3)** | | | |
| NFR3.1 | The system should respond to each request within 5 seconds. | High | ✔ |
| NFR3.2 | Any diagrams or animations attached to the question should take less than 3 seconds to load. | High | |
| NFR3.3 | A question of the chosen topic and difficulty level should be loaded within 5 seconds upon being matched. | Medium | |
| NFR3.4 | The maximum delay between a user's modification and the other user(s) seeing the modification is 0.5 seconds for the text editor. | High | ✔ |
| NFR3.5 | The rooms should be set up within 7 seconds. | Medium | |
| NFR3.6 | The text editor should be available to the user within 5 seconds. | Medium | |
| NFR3.7 | The chat feature should have a maximum end-to-end delay of 2 seconds. | High | ✔ |

|  |  |  |  |
|---|---|---|---|
| **Usability (NFR4)** |  |  |  |
| NFR4.1 | The system should support concurrent matchmaking without any error. | High | ✔ |
| NFR4.2 | The stored history data must be consistent. | High |  |
| NFR4.3 | The UI should have a consistent color scheme. | Medium |  |
| NFR4.4 | The application should be accessible via a public url. | Medium |  |
| NFR4.5 | The application should be intuitive and easy to use. | High |  |
| **Scalability (NFR5)** |  |  |  |
| NFR5.1 | The system should support up to 50 concurrent matchmaking requests. | Medium |  |
| NFR5.2 | All requests to the application should be facilitated by an API Gateway. | High |  |
| NFR5.3 | The application should be available, scalable and performant. | Medium |  |

## 3.3 Quality Attributes Identified

The above NFRs were crafted using the Quality Attributes we identified. The following is a subset of the quality attributes identified.

| S/N | Attribute | Importance & Relevance | Internal/External |
|---|---|---|---|
| 1 | Security | User authentication, encryption of user passwords and data privacy of the chat and code editor contents are likely to contain sensitive data which needs to be protected by security measures. | External |
| 2 | Availability | Our application has to be up and fully operational most of the time. | External |
| 3 | Performance | As one of the selling points of this application is the feature of a real-time collaborative mode, performance is crucial in this aspect to maintain a stable functioning application. This is specifically so for features such as the chat and collaborative code editor, which will not be usable if not performant. | External |
| 4 | Usability | Usability in this application ensures the ease of use and intuitiveness for users. This means that users can easily navigate its interface, and can easily understand and determine the purpose of each feature and what it can do. Making the application easily usable and intuitive is crucial in helping to appeal to new users and retain old users, since this application needs to have more active users before it can even proceed to matching users. | External |
| 5 | Scalability | Support for a large student population is crucial to the platform's effectiveness and usability. | Internal |

## 3.4 Reasoning for NFRs

| S/N | Reasoning |
|---|---|
| **Security (NFR1)** | |
| NFR1.1 | User passwords should be stored securely to prevent hackers from stealing user accounts. |
| NFR1.2 | Access tokens grant users access to privileged API endpoints. In cases where they are stolen, an expiration date helps provide added security. |
| NFR1.3 | Allowing users to enter rooms they don't belong to will mean confidential chat and editor information seen by unintended audiences. |
| **Availability (NFR2)** | |
| NFR2.1 & NFR2.2 | Down time of our application will make it unusable. |
| **Performance (NFR3)** | |
| NFR3.1 to NFR 3.7 | These metrics cover core functionalities of our application. Their running with minimal lag and latency is crucial to allowing users to focus on programming problems and the interview process. |
| **Usability (NFR4)** | |
| NFR4.1 | For the application to be usable by the student population, it has to support multiple concurrent room sessions. |
| NFR4.2 | The history of attempts must reflect the reality of past attempts for users to learn effectively. |
| NFR4.3 | User interface is crucial to its usability |
| NFR4.4 | The app should be easily recognisable and accessible via a public domain name. |
| **Scalability (NFR5)** | |
| NFR5.1 | Beyond usability, we need to ensure the matching system is designed to support a large student population. |
| NFR5.2 | An API Gateway will be crucial to supporting scalability of various functions |

## 4. Quality Attributes Prioritization Matrix

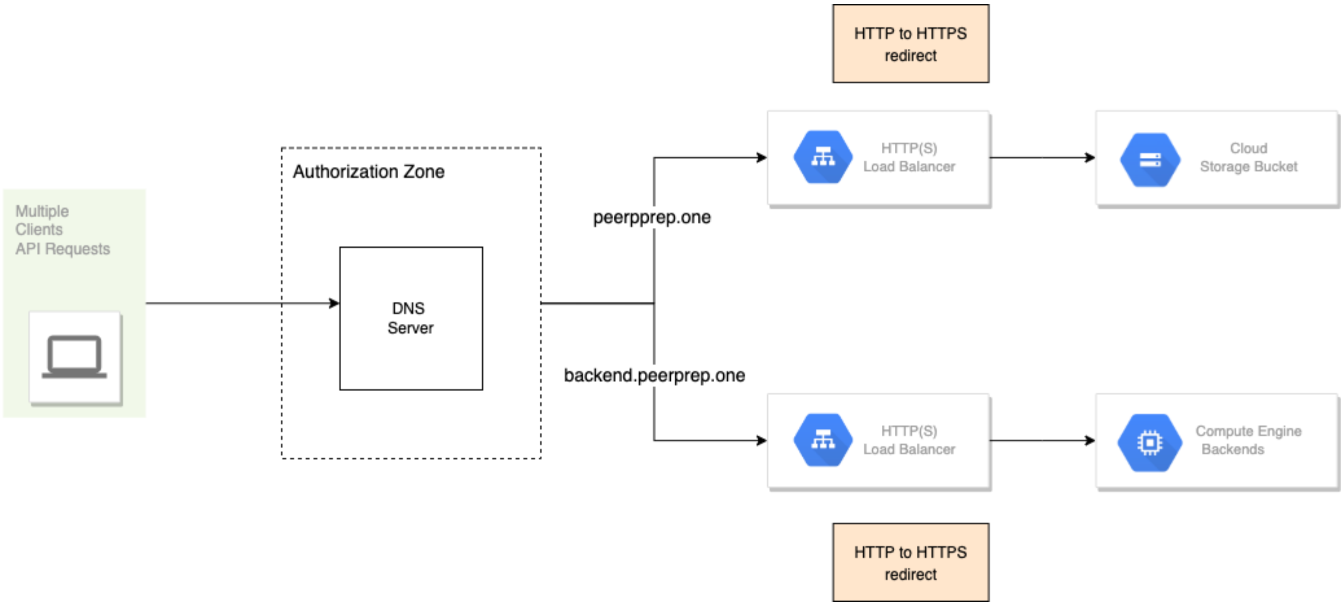
| Attribute | Score | Availability | Integrity | Performance | Reliability | Robustness | Security | Usability | Verifiability |
|---|---|---|---|---|---|---|---|---|---|
| Availability | 6 | | < | < | < | ^ | < | ^ | < |
| Integrity | 3 | | | ^ | ^ | ^ | < | ^ | < |
| Performance | 5 | | | | < | ^ | < | ^ | < |
| Reliability | 4 | | | | | ^ | < | ^ | < |
| Robustness | 7 | | | | | | < | ^ | < |
| Security | 2 | | | | | | | < | < |
| Usability | 8 | | | | | | | | < |
| Verifiability | 1 | | | | | | | | |

We prioritized usability the most in our application because our application works best with a large number of user base. We need more users, so that our users can find a match quickly, and usability is a key factor behind it.

To improve usability, we exposed our application on the domain name “peerprep.one” for users to easily remember and access our application anywhere in the world.

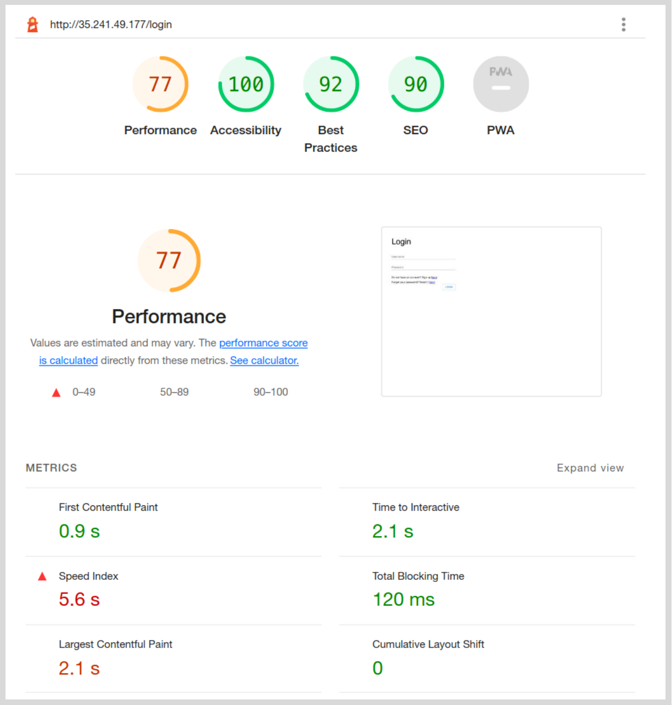
To provide a good user experience, we need the system to be robust, highly available, performant and reliable. Our use of a microservices architecture and deployment stack enables reliability, performance and scalability as explained in the Deployment section.

Certain design choices such as the employment of Pub-Sub pattern in the use of Redis Adapters for Socket.io allows for scalability, so we can handle higher user loads, and the user is able to access our system whenever they want, and have our system working as they would expect it to be.



*High Level Overview of Architecture Diagram*

Since we are not storing sensitive information, security is not of a high priority. However we understand that it is imperative for any website to be trustable, hence we implemented security measures such as support for HTTPS, HTTP to HTTPS redirect and best practices when storing user data like salting and hashing the passwords.



http://35.241.49.177/login

77 Performance
100 Accessibility
92 Best Practices
90 SEO
PWA

77

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 50–89 90–100

METRICS

Expand view

First Contentful Paint
0.9 s

Time to Interactive
2.1 s

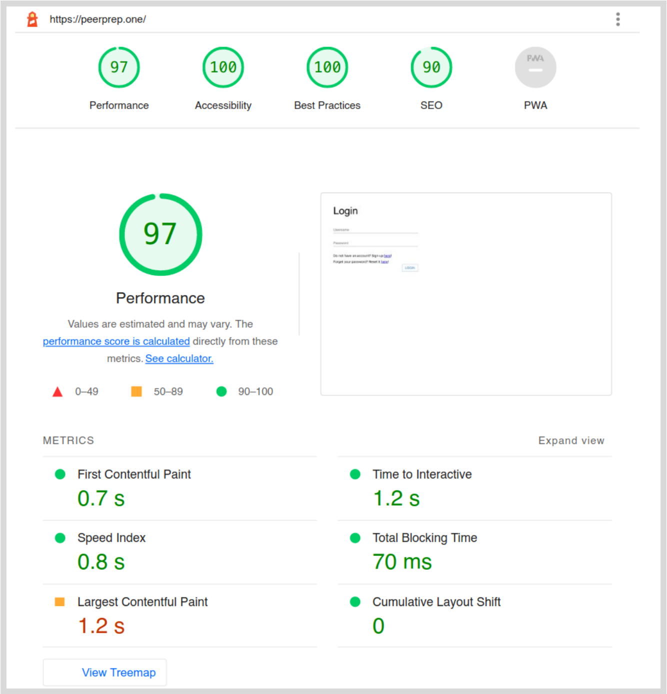
▲ Speed Index
5.6 s

Total Blocking Time
120 ms

Largest Contentful Paint
2.1 s

Cumulative Layout Shift
0

*Before*



https://peerprep.one/

97 Performance

100 Accessibility

100 Best Practices

90 SEO

PWA

97

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

Login

METRICS

Expand view

● First Contentful Paint
0.7 s

● Time to Interactive
1.2 s

● Speed Index
0.8 s

● Total Blocking Time
70 ms

■ Largest Contentful Paint
1.2 s

● Cumulative Layout Shift
0

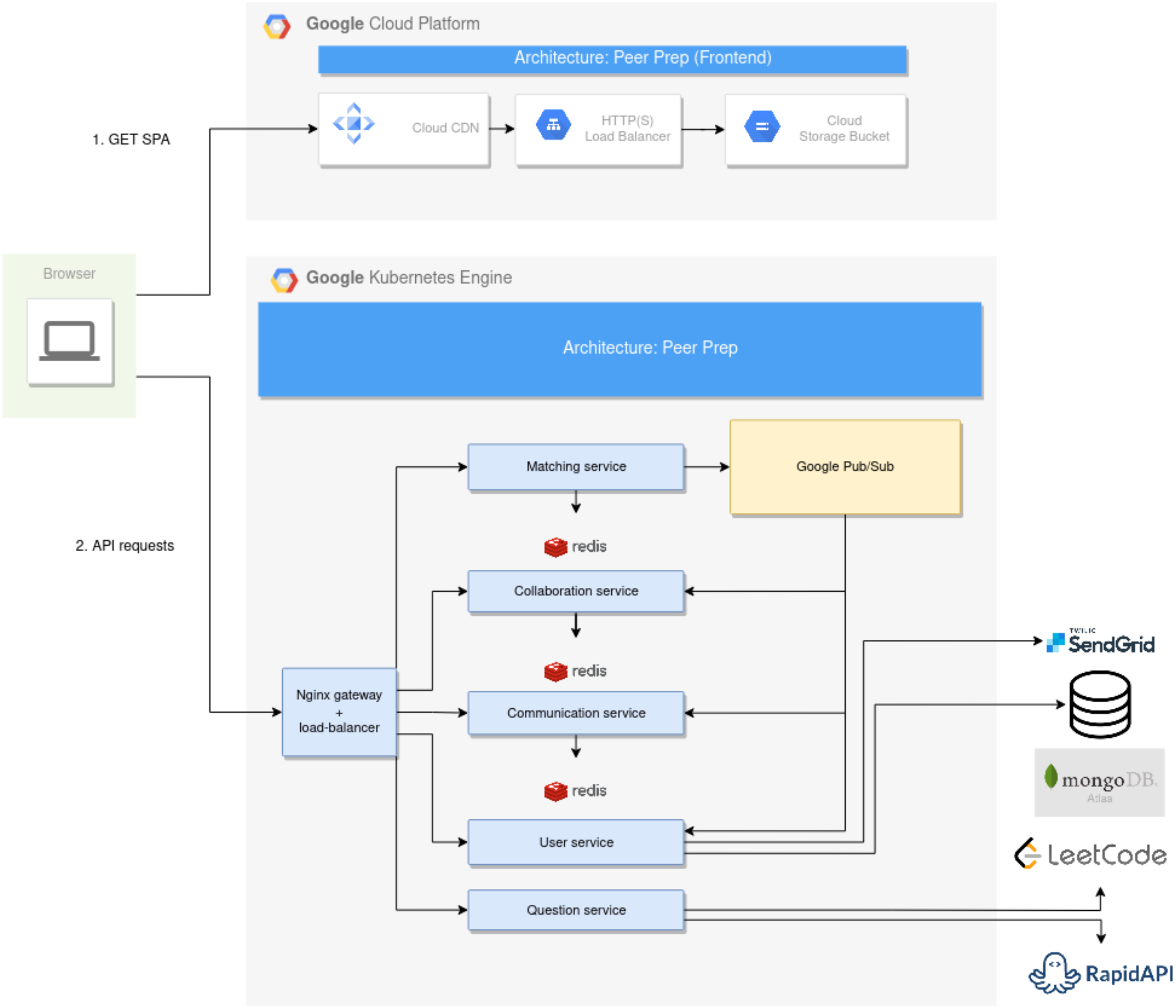
View Treemap

*After*

To increase the usability of our product, we optimized our page by implementing https and hosting the frontend in a CDN. We were able to quantify an improvement in our frontend with the use of lighthouse as we can see from the pictures above.

# 5. Developer Documentation & Artifacts

## 5.1 Architecture Design



In our architecture design, we used an API gateway to provide a single point of entry for all the backend services, as well as load balancing. This increases usability from the perspective of the frontend. The API gateway is responsible for authentication by checking the JWT token for the incoming requests. By offloading the authentication responsibility to the API gateway, we achieve better separation of concerns. It also allows for reusability and modularity as our individual services will not have to reimplement the same authentication functionality, thereby increasing the maintainability of our software.

For communication methods, we opted for a mix of synchronous HTTP(S) requests from the frontend to the backend, and also asynchronous message passing between our backend services. We used synchronous HTTP(S) requests because it is a common protocol to serve as an interface between the frontend and backend services. This provides the frontend engineers with familiarity, and boosts development speed. Asynchronous message passing was used for the communication between backend services. Specifically, we were using pub sub, implemented by Google Pub/Sub. The asynchronous nature of the communication decouples the communicators, and the message queue inverts the dependencies such that the communicators are now dependent on the message queue instead of each other, providing us with lower coupling. This increases the overall testability and improves the quality of our software. Lastly, we chose to use a pub sub model, since we have to communicate from one to many, because a match event is produced by the matching service and consumed by collaboration and communication service for socket authentication, as well as user service to save the past attempts of the questions.

While the collaboration and communication services do have very similar code structure, we decided to separate them into 2 independent services. This is because they have different responsibilities and we are able to apply both separation of concerns and single responsibility principle by separating the 2 services. It also enables us to scale the services independently, allocate more resources to the services independently based on their usage patterns. Moreover, the collaboration service is pivotal to our application, and we would not want a bug or a performance issue from the communication service to impact our collaboration service. Separation of the microservices enables better isolation.

## 5.2 Architectural Decisions

Architecture Choice: **Microservice Architecture**

| Options | For | Against |
|---|---|---|
| Microservice architecture | Enables the continuous delivery and deployment of large, complex applications<br>Fault Isolation<br>Eliminates any long-term commitment to a technology stack | Additional inter-service communication mechanism has to be implemented<br>More difficult to test the interaction between different components |
| Monolithic architecture | Traditional architecture and hence more familiarity with it<br>Easy deployment as we only need to deploy one executable file or directory<br>High performance as it is not subjected to network connectivity | Large application makes the development more complex and slower<br>Individual components cannot be scaled independently based on the usage<br>Small change in the code requires a redeployment of the entire application |

We chose microservice architecture because it offers a flexible scaling mechanism based on the load capacity of the microservices. Each microservice can be scaled horizontally, allowing efficient allocation of resources when the load is uneven. Besides, it can also help to simplify and speed up our development process as each member can work on each microservice independently after deciding on the application interface protocol.

Container Orchestration Framework: **Kubernetes for Cloud Deployment**

| Options | For | Against |
|---|---|---|
| Docker Swarm | Easy to install and configure. It does not require configuration changes if your system is already running | Limited functionality<br>Automation capability is not as robust compared to |

| | | |
|---|---|---|
| | inside Docker.<br>Offers automatic load balancing tools<br>Supports monitoring through third part application<br>Offers auto-scaling of instances quickly and on demand | Kubernetes |
| Kubernetes | Very customisable<br>Has built-in monitoring along with third party monitoring tools integration support<br>Provides scaling based on traffic. Horizontal scaling is built in. | Complex installation process and a steep learning curve. Can be overly complicated and result in a loss of productivity. |

We chose Kubernetes as our container orchestration framework because it provides high availability, and supports monitoring and more transport layer protocol. The cons of the Kubernetes also do not apply to us that much, as our team members are quite familiar with the framework. More importantly, it is more configurable, meaning that we will have more flexibility on the deployment.

Frontend-to-Backend Communication: **API Gateway**

| Options | For | Against |
|---|---|---|
| API Gateway | Reduce the coupling between the client and microservices. The client does not need to know the IP address of every microservice<br>Reduce the risk of security attack by narrowing down the IP addresses exposed to the internet | Single point of failure<br>Increased response time due to additional network call<br>Requires additional development |

| | | |
|---|---|---|
| Direct client-to-microservice communication | No need to setup and host an additional deployment or service | Security concerns. If each microservice is accessed via a public endpoint, there’s a higher risk of attack.<br>Tight coupling and increased complexity in the frontend. The client needs to know the implementation or ip address of every microservice to work properly. |

We decided to go with the API gateway implementation because we value the security of our services and we do not want our frontend to be tightly coupled with the microservices. In addition, we were also able to reduce the code duplication for request authentication by handling it in the gateway instead of in all microservices.

Communication Between Microservices: **Using a message broker to manage dependencies between Matching Service, Collaboration and Communication Service.**

| Options | For | Against |
|---|---|---|
| Matching Service to Post Match details directly to Collaboration and Communication Services. | Synchronization is simpler.<br>- Matching service can ensure that match details are successfully posted to Collaboration and Communication Services before returning a match to frontend. | Tighter coupling between Matching service and Communication/Collaboration service.<br>- When a new service X which depends on match details needs to be added to PeerPrep, both Matching and X will require modifications. |
| Matching Service to post match details to a message broker<br>Collaboration and Communication Services will receive match details via message broker | Resemblance with Observer Pattern<br>- Services requiring match information subscribe to topics which notify them of updates on matches created.<br>- Open/Closed Principle: Ease of adding new Services which depend on match details. No change required on Publisher. | Challenges in Synchronization<br>- Matching service does not know if Collaboration and Communication Service has received the message.<br>- Matching service returns match details to Frontend, Frontend may potentially attempt to connect to Collaboration and |

| | | |
|---|---|---|
| | Looser coupling between Matching service and the other services as they all communicate through an Interface, the format of messages passed. | Communication Services before they know of the match. Resulting in Frontend being denied connection. |

Beyond these considerations, we made the decision to use Google Cloud Pub/Sub instead of a self managed messaging service such as a Redis Deployment in our cluster. An analysis of this decision is mentioned in the Message Queues(Pub/Sub) section below.

### 5.2.1 Microservices Discovery Process

First, we identified the subdomains of PeerPrep:

- User
- Question
- Text Editor
- Interviewer
- Interviewee
- Match

Secondly, we mapped them to the contexts they pertain to.

| **Bounded Contexts** | **Subdomain** |
|---|---|
| User Context | User Creation<br>- username<br>- password<br>- email |
| | User Deletion<br>- username |
| | User Update<br>- new password |
| | User Recovery<br>- email |
| | User attempt history<br>- Editor contents<br>- Communication contents |
| Interview Question | Question<br>- Difficulty<br>- Details<br>- Answer |
| | Verification<br>- Code compilation & Output |
| Real-time editor | Real-time Text Editor<br>- Language |

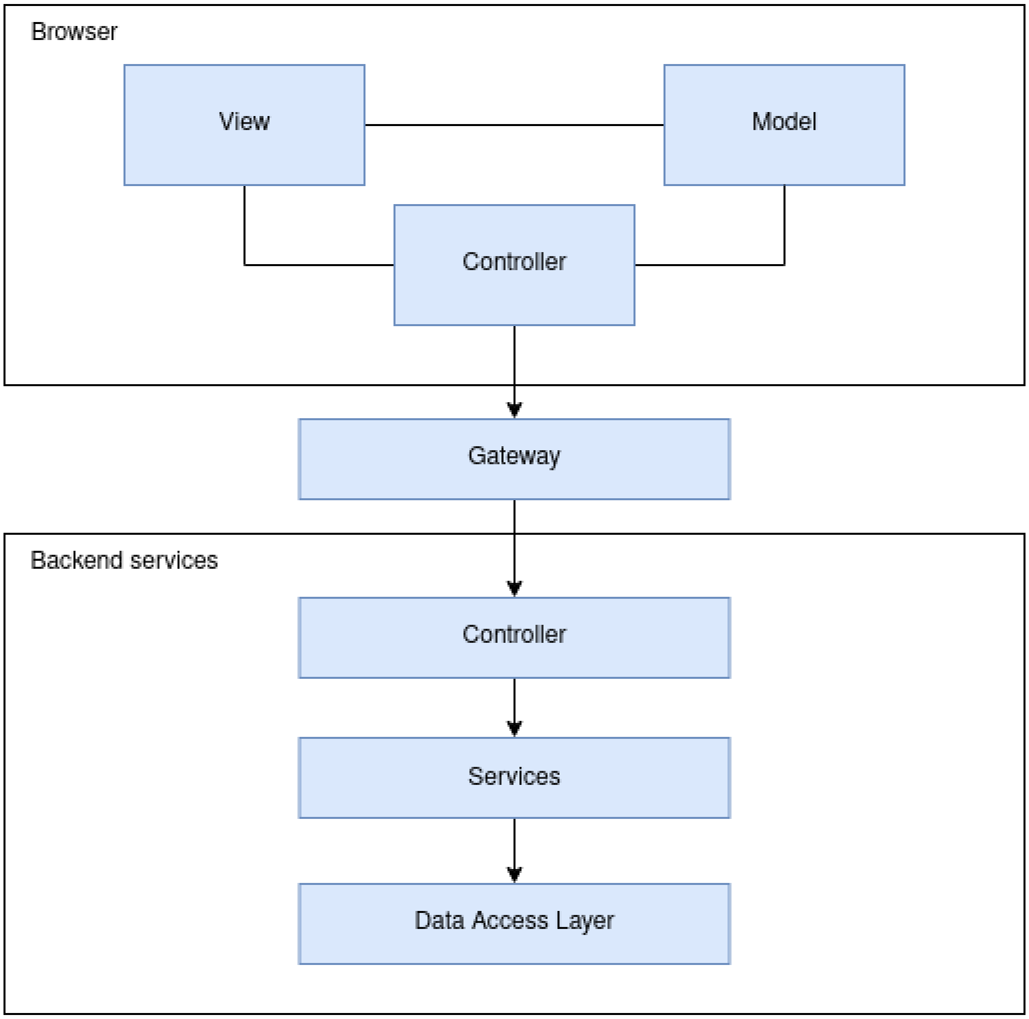
| | |
|---|---|
| | - Theme |
| | Verification<br>- Code compilation & Output |
| Communication | Interviewer role<br>- Asking Interviewer Questions(Time complexity, Space Complexity)<br>- Ability to select questions for interviewee<br>- Visibility on interviewee’s attempt |
| | Interviewee<br>- Access to real-time editor<br>- Access to question details<br>- Access to requested difficulty |
| Match | Interview Rooms<br>- Difficulty<br>- Participants |
| History | Attempt History<br>- Code editor contents<br>- Chat contents<br>- Question details |

Through this process of identifying how subsets of our problem space (subdomains) map to subsets of the solution space (bounded contexts), we managed to identify each Bounded Context as a Microservice and the grouping functionalities solving each subdomain in microservices.

For instance, in identifying that “Interviewer” access pertains to the Communication Context, we concluded that interviewer-specific access, such as access to a list of qualitative Interview questions, should belong to the communication service. When implementing Communication Service, this design made it easy for the chat feature to render UI based on Interviewer or Interviewee roles.

## 5.3 Microservice Design Details

### 5.3.1 General Component Design

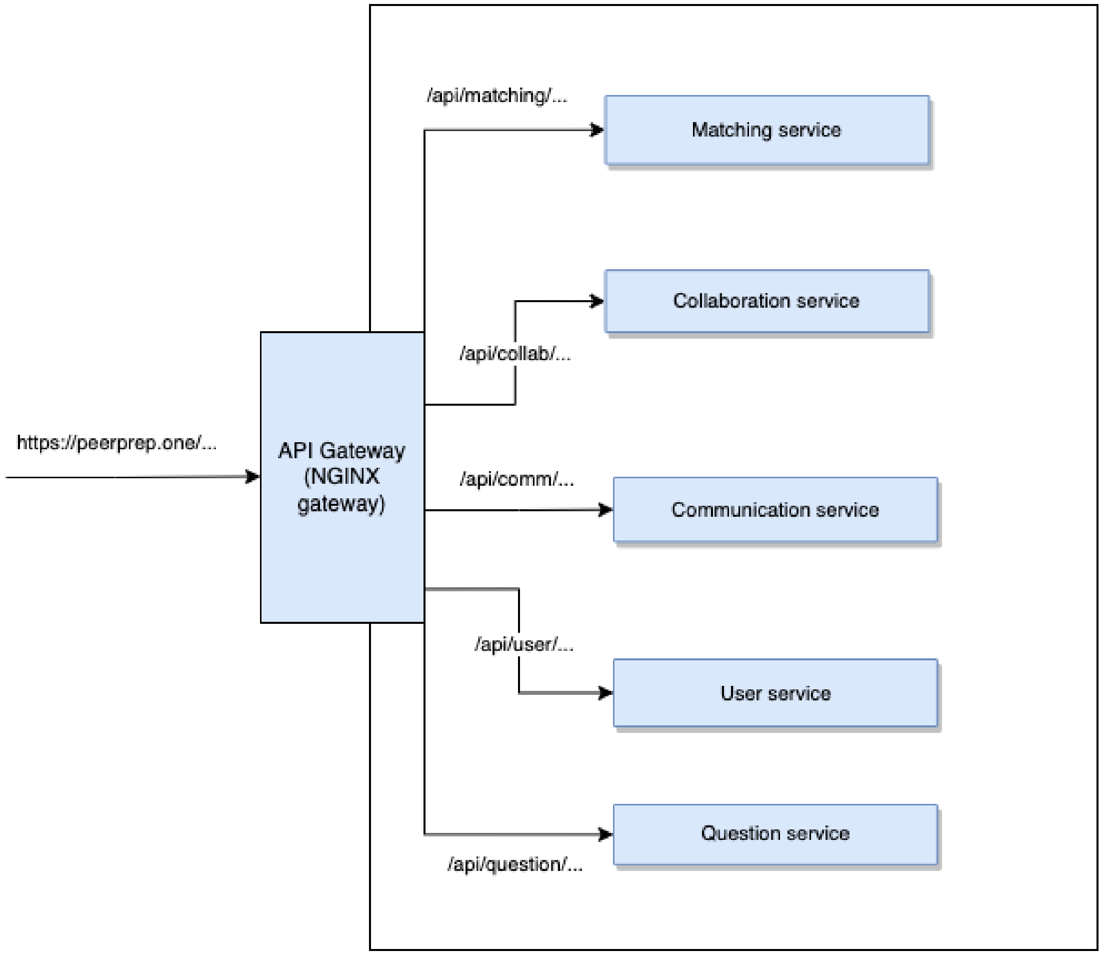


In general, we adopted a **Web MVC - SPA architecture**. Our frontend controller would then call the backend API controllers to receive data when needed. The backend services are all behind an API gateway, encapsulating the internal architecture of our system. Each backend service follows a layered architecture. While we have shown a diagram with 3 layers consisting of the controller, services and data access layers, not all services have all the layers, because some services are stateless and do not use a persistent data store.

### 5.3.2 API Gateway

We have discussed numerous drawbacks with clients accessing the microservices directly in the frontend. The lack of encapsulation makes it difficult to change the microservice implementation or even the deployed ip addresses and ports.

Hence, it is more desirable to have a simple and unified interface like API gateway to encapsulate the internal system and provide a set of interfaces to its clients. This pattern is similar to the **facade pattern** from object-oriented design.



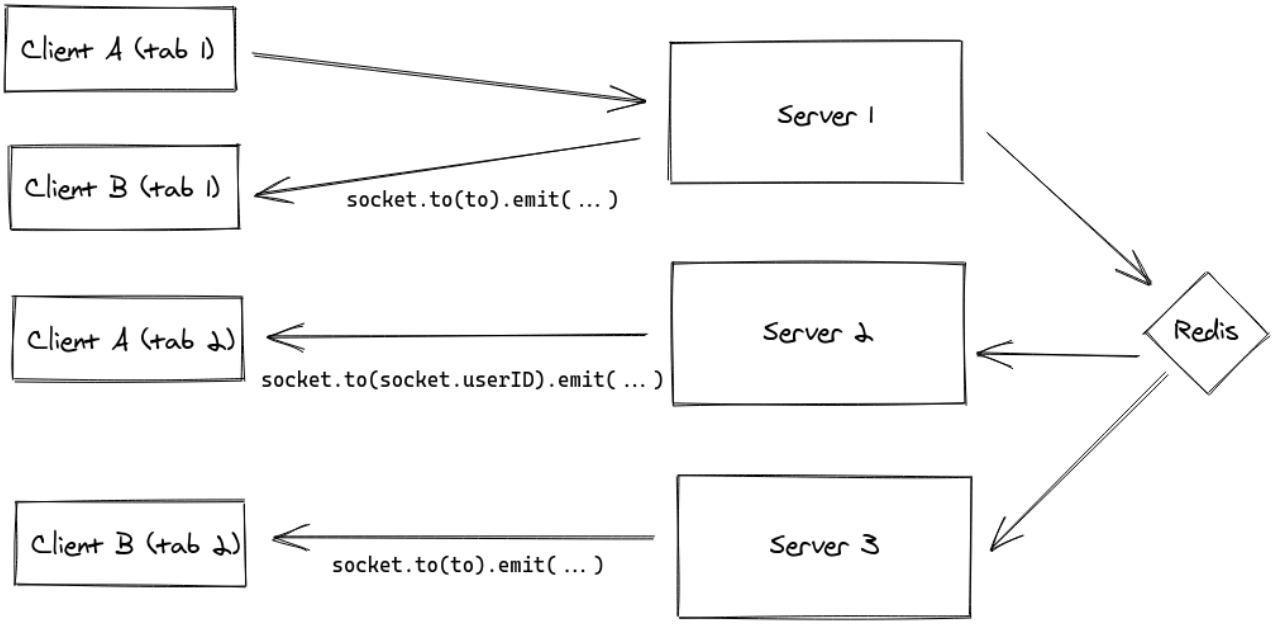
This pattern provides a reverse proxy to redirect or route requests to your internal microservices endpoints. An API gateway provides a single endpoint or URL for the client applications, and it internally maps the requests to internal microservices. A layer of abstraction is provided by hiding certain implementation details.

| **Pro** | **Con** |
|---|---|

| | |
|---|---|
| Open-closed Principle: Crucially this means additional functionality such as response and request transformations, endpoint access authentication and authorisation, and request monitoring can be added to the backend service, without having to make any changes to frontend. | Single point of failure on the API Gateway. However this is circumvented with deployment techniques such as push notifications of pod failures on GCP Kubernetes will also automatically and immediately bring in a new pod to replace our API gateway when needed. |

### 5.3.3 Redis Adapter for scaling to multiple Socket.IO servers

A Socket.IO adapter is a component which is responsible for broadcasting events to all or a subset of clients. As we scale the number of microservice instances, client sockets may be connected to different microservice instances and hence become unaware of other clients. Therefore, we employ adapters to achieve the synchronization between all these microservice instances using the Pub/Sub mechanism of Redis.



[1] *Illustration of servers using Redis adapter*

The diagram above shows the Redis adapter at work, which follows the paradigm of a **publisher/subscriber pattern**. Client A (tab 1) is able to communicate with Client B (tab 1),

[1] Adapter | Socket.IO

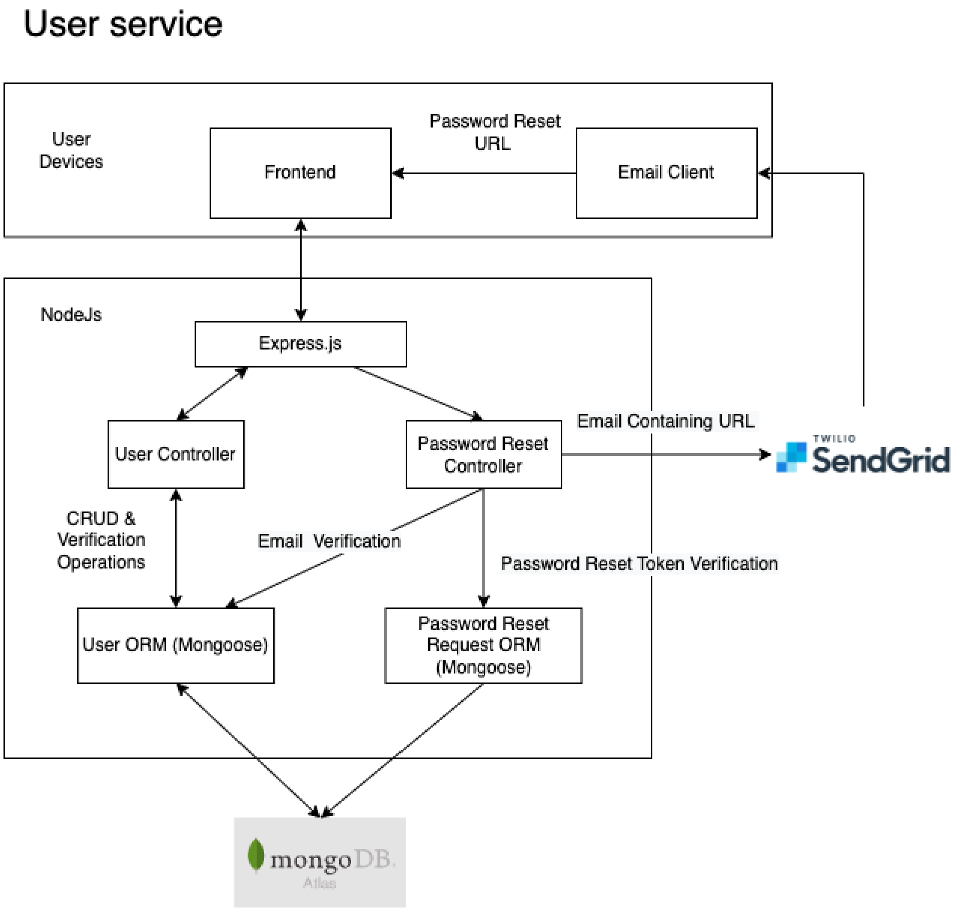
Client A (tab 2) and Client B (tab 2) regardless of which microservice instance they are connected to.

The Redis adapter is used in Matching, Collaboration, and Communication microservice.

### 5.3.4 User Service

User service is responsible for the following functionalities:

- User Management & Authentication
- User Account Password Reset
- Maintaining User Attempt History



#### 5.3.4.1 Tech Stack

- NodeJs (Express)
- MongoDB + Mongoose
- Redis

- Twilio SendGrid[2]

### 5.3.4.2 User Management & Authentication

CRUD operations on user accounts & Issuing Access and Refresh tokens for User Authentication

#### 5.3.4.2.1 API Endpoints

Create and Login requests do not require authentication.

1. Create User

```
POST /api/user/createuser
```

2. Login

```
POST /api/user/login
```

Renew access and refresh tokens & logout user requests in user service require the refresh token in the Authorization header:

```
Authorization: Bearer <Refresh Token>
```

3. Renew Access and Refresh Tokens

```
POST /api/user/renewtokens
```

4. Logout User

```
POST /api/user/logout
```

All other requests in user service require a valid Authorization header.

```
Authorization: Bearer <Access Token>
```

5. Delete User

```
DELETE /api/user/renewtokens
```

6. Update User Password

```
PUT /DELETE /api/user/renewtokens
```

[2] SendGrid Email API

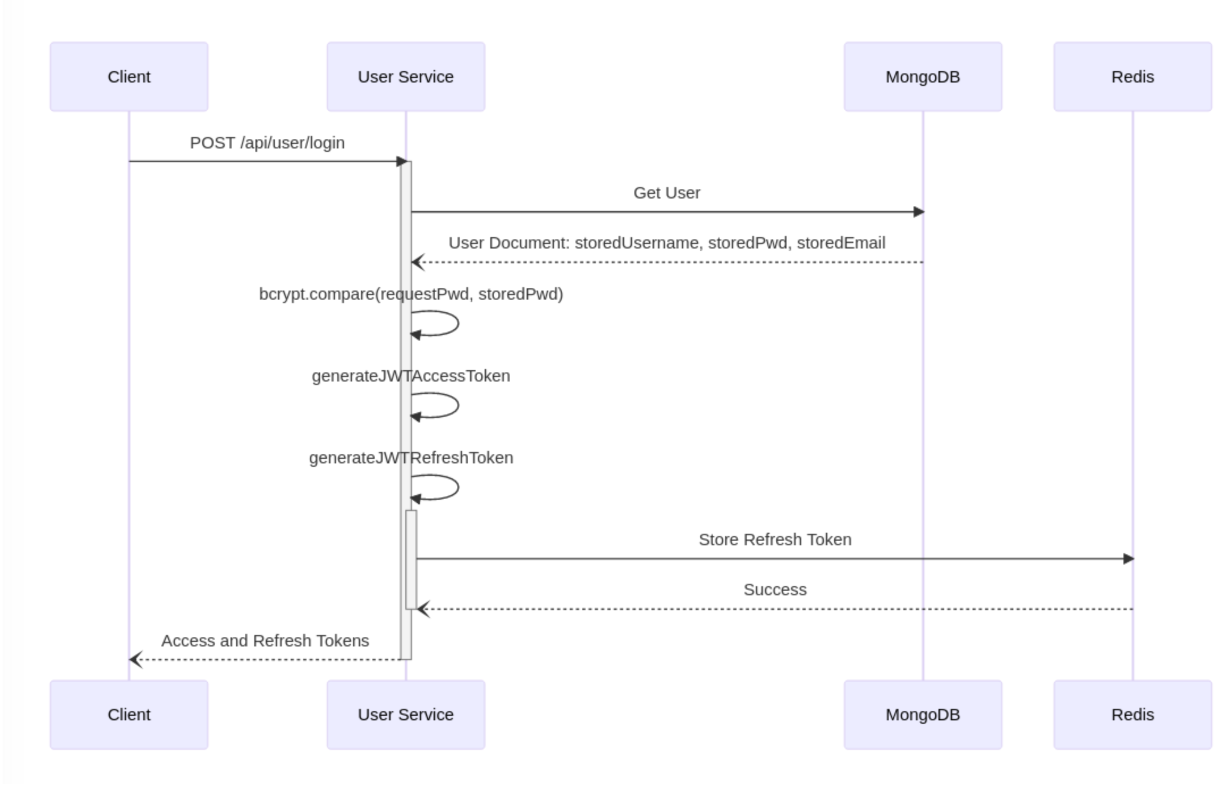
#### 5.3.4.2.2 Mongoose Schema

User service stores User details in MongoDB. We utilize Mongoose to interact with MongoDB via a Schema.

| **UserModelSchema** | | |
|---|---|---|
| **Field** | **Type** | **Required** |
| username | String(Unique) | True |
| password | String | True |
| email | String | True |

Each mongoose schema corresponds to a Collection in MongoDB, hence each object representing a User is stored as a Document in the “UserModelSchema” Collection. Username serves as the Primary Key of the UserModel Collection hence most APIs query the User collection using username.

### 5.3.4.2.3 Token Issuing Process (Sequence Diagram)



### 5.3.4.2.4 Design Decisions

**Decision: Storing Refresh Tokens in Redis**

| | **For** | **Against** |
|---|---|---|
| **Storing refresh tokens in RAM** | Less resource intensive deployment/architecture | Not scalable beyond 1 instance of User Service<br>All refresh tokens deleted if User Service shuts down |
| **Storing refresh tokens in Redis** | Allows more than 1 instance of User Service as all instances can access a single source of refresh token storage | More Deployments in our Kubernetes Cluster to be provisioned and managed |

| | | |
|---|---|---|
| | Tokens do not get deleted when User Service Fails - Reliability.<br>Newly scaled User Service Pods can serve APIs involving refresh token as if no outage happened.<br>Layered architecture: Separating data from logic for better extensibility. | |

We would also like to note that Passwords are encrypted using the bcrypt package before being stored in MongoDB. Hence it will be very difficult for hackers to login using the stored password information even if they hack into our MongoDB Collection since it requires decrypting the password.

#### 5.3.4.3 User Account Password Reset

##### 5.3.4.3.1 API Endpoints

1. Request Password Reset via Email

```
POST /api/user/sendresetlink
```

2. Reset Password via Reset Link

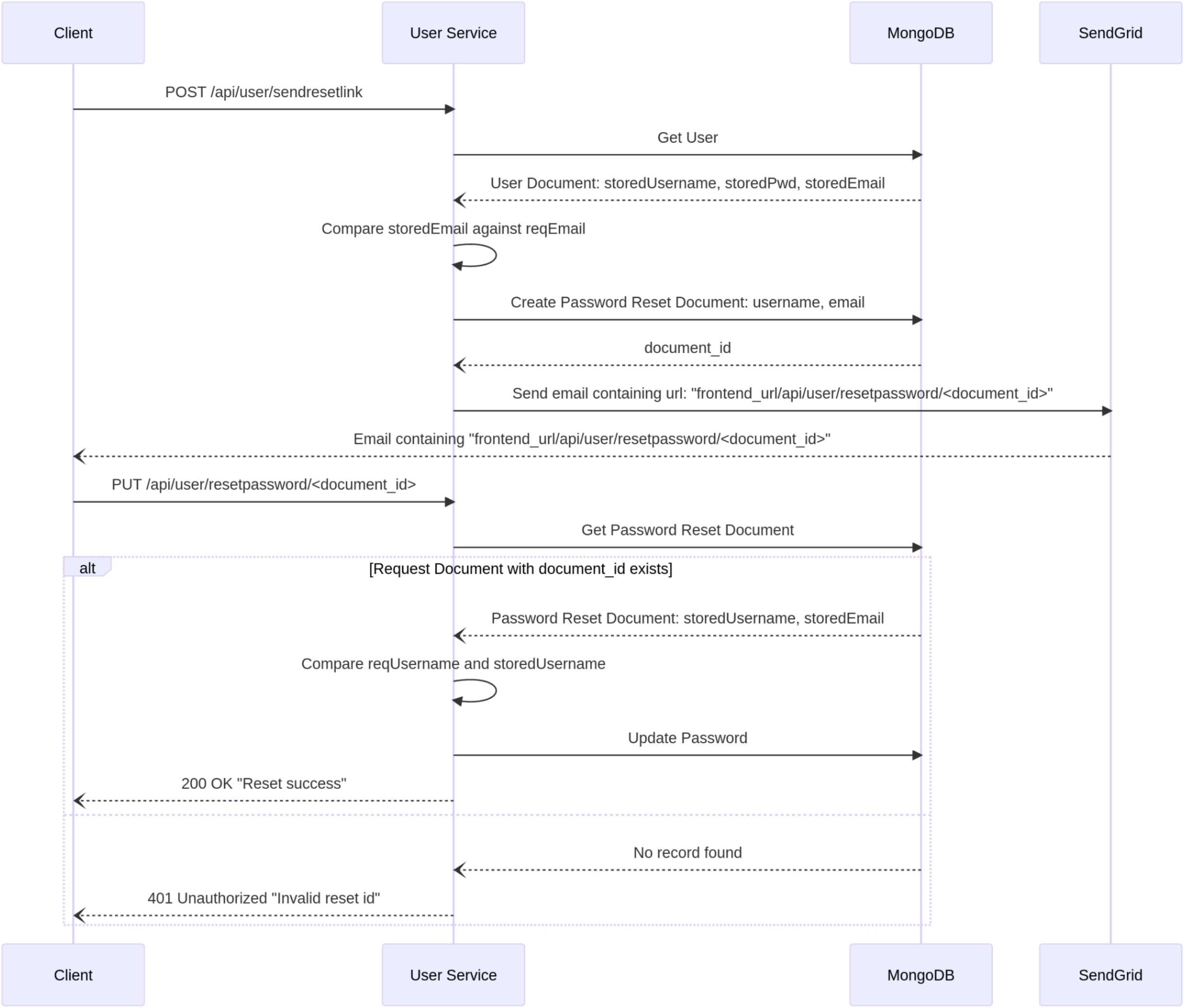
```
PUT /api/user/resetpassword
```

##### 5.3.4.3.2 Mongoose Schema

User service stores various entities in MongoDB. We utilize Mongoose to interact with MongoDB via a Schema.

| **PasswordResetSchema** | | |
|---|---|---|
| **Field** | **Type** | **Required** |
| username | String | True |
| email | String | True |

### 5.3.4.3.3 Password Reset Process


Client
User Service
MongoDB
SendGrid
POST /api/user/sendresetlink
Get User
User Document: storedUsername, storedPwd, storedEmail
Compare storedEmail against reqEmail
Create Password Reset Document: username, email
document_id
Send email containing url: "frontend_url/api/user/resetpassword/<document_id>"
Email containing "frontend_url/api/user/resetpassword/<document_id>"
PUT /api/user/resetpassword/<document_id>
Get Password Reset Document
alt
[Request Document with document_id exists]
Password Reset Document: storedUsername, storedEmail
Compare reqUsername and storedUsername
Update Password
200 OK "Reset success"
No record found
401 Unauthorized "Invalid reset id"
Client
User Service
MongoDB
SendGrid

When the Client clicks on the URL attached in the reset email sent to their user account's email address, they access the reset page in the frontend which will submit the PUT resetpassword request containing the <document_id> as appended at the end of the URL. As shown in the sequence diagram above, this document ID is crucial in retrieving the reset request and validating that the resetpassword request originates from the initial sendresetlink request. This ensures that only the owner of the user's email account can reach the reset page on the frontend. The username field in the PasswordResetSchema is used to validate the resetpassword request as well.

### 5.3.4.4 Maintaining User Attempt History

Stores and Retrieves the following content

- Code Editor Contents & Programming Language settings
- Chat history
- Match details: room_id, difficulty level, matched users.

#### 5.3.4.4.1 API Endpoints

1. Retrieve question history

```
GET /api/user/questionhistory?uid={uid}&limit={limit}&offset={offset}
```

2. Save Question

```
POST /api/user/resetquestion
```

3. Get / Save Message

```
GET /api/user/message
```

```
POST /api/user/message
```

#### 5.3.4.4.2 Mongoose Schema

User service stores various entities in MongoDB. We utilize Mongoose to interact with MongoDB via a Schema.

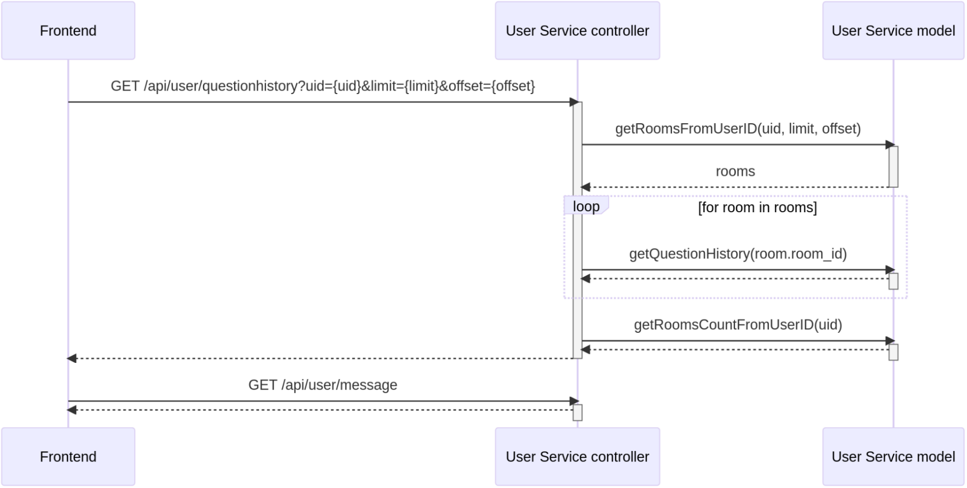
| **MatchHistorySchema** | | |
|---|---|---|
| **Field** | **Type** | **Required** |
| room_id | String | True |
| difficulty_level | String | True |
| users | Array of Strings | True |

| | | |
|---|---|---|
| usernames | Array of Strings | False |

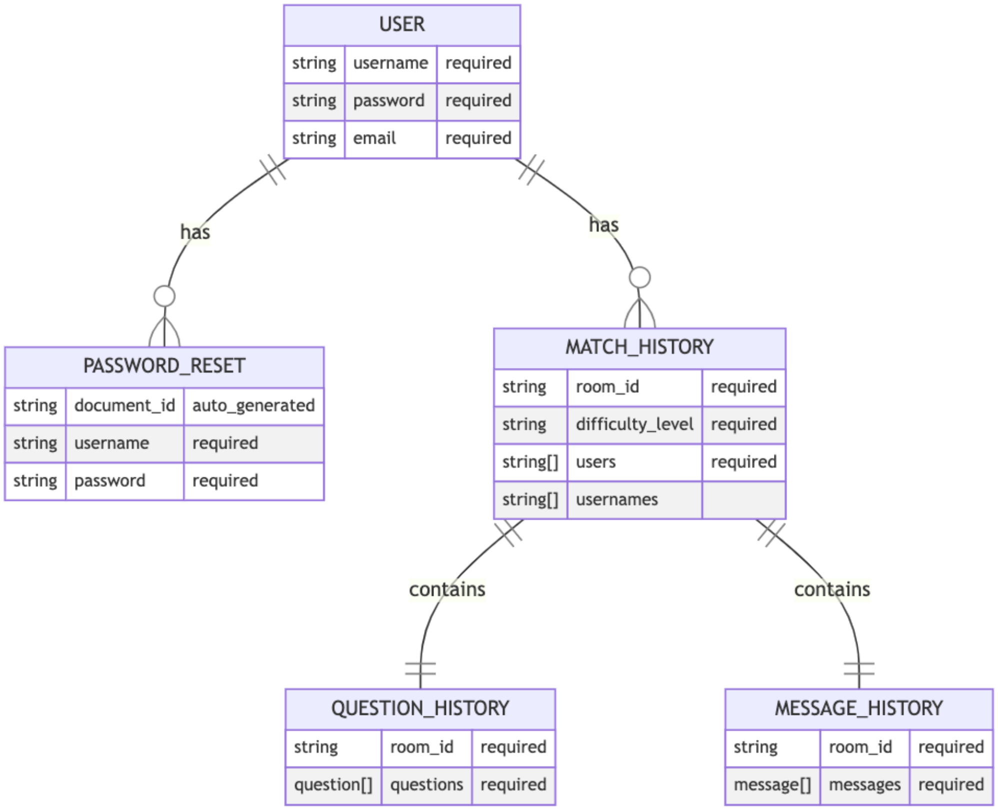
| **QuestionHistorySchema** | | | | |
|---|---|---|---|---|
| **Field** | **Type** | | | **Required** |
| room_id | String | | | True |
| questions | Array of Objects | **Object Fields** | **Type** | - |
| | | titleSlug | String | False |
| | | codeSegment | String | False |
| | | language | Number | False |

| **MessageHistorySchema** | | | | |
|---|---|---|---|---|
| **Field** | **Type** | | | **Required** |
| room_id | String | | | True |
| messages | Array of Objects | **Object Fields** | **Type** | - |
| | | text | String | False |
| | | username | String | False |
| | | socketID | String | False |
| | | id | String | False |

### 5.3.4.4.4 Retrieving Attempt History Process


Frontend
User Service controller
User Service model
GET /api/user/questionhistory?uid={uid}&limit={limit}&offset={offset}
getRoomsFromUserID(uid, limit, offset)
rooms
loop
[for room in rooms]
getQuestionHistory(room.room_id)
getRoomsCountFromUserID(uid)
GET /api/user/message
Frontend
User Service controller
User Service model

### 5.3.4.5 User Service Database



USER

| | | |
|---|---|---|
| string | username | required |
| string | password | required |
| string | email | required |

PASSWORD_RESET

| | | |
|---|---|---|
| string | document_id | auto_generated |
| string | username | required |
| string | password | required |

MATCH_HISTORY

| | | |
|---|---|---|
| string | room_id | required |
| string | difficulty_level | required |
| string[] | users | required |
| string[] | usernames | |

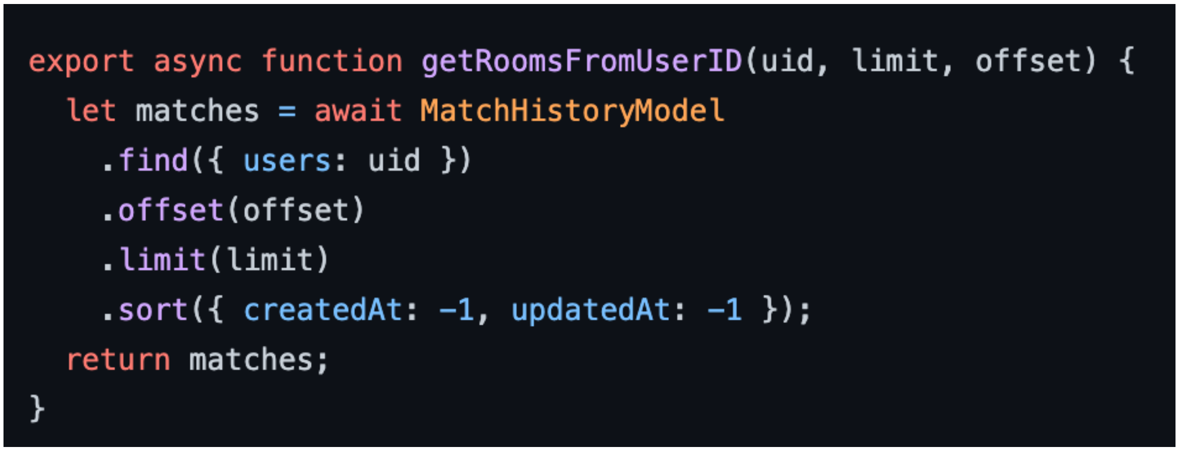
QUESTION_HISTORY

| | | |
|---|---|---|
| string | room_id | required |
| question[] | questions | required |

MESSAGE_HISTORY

| | | |
|---|---|---|
| string | room_id | required |
| message[] | messages | required |

User service communicates with MongoDB Atlas Database via an object data modeling library called Mongoose. The library is implemented with the **Builder pattern** and omits the query parameter during object construction or initialisation.

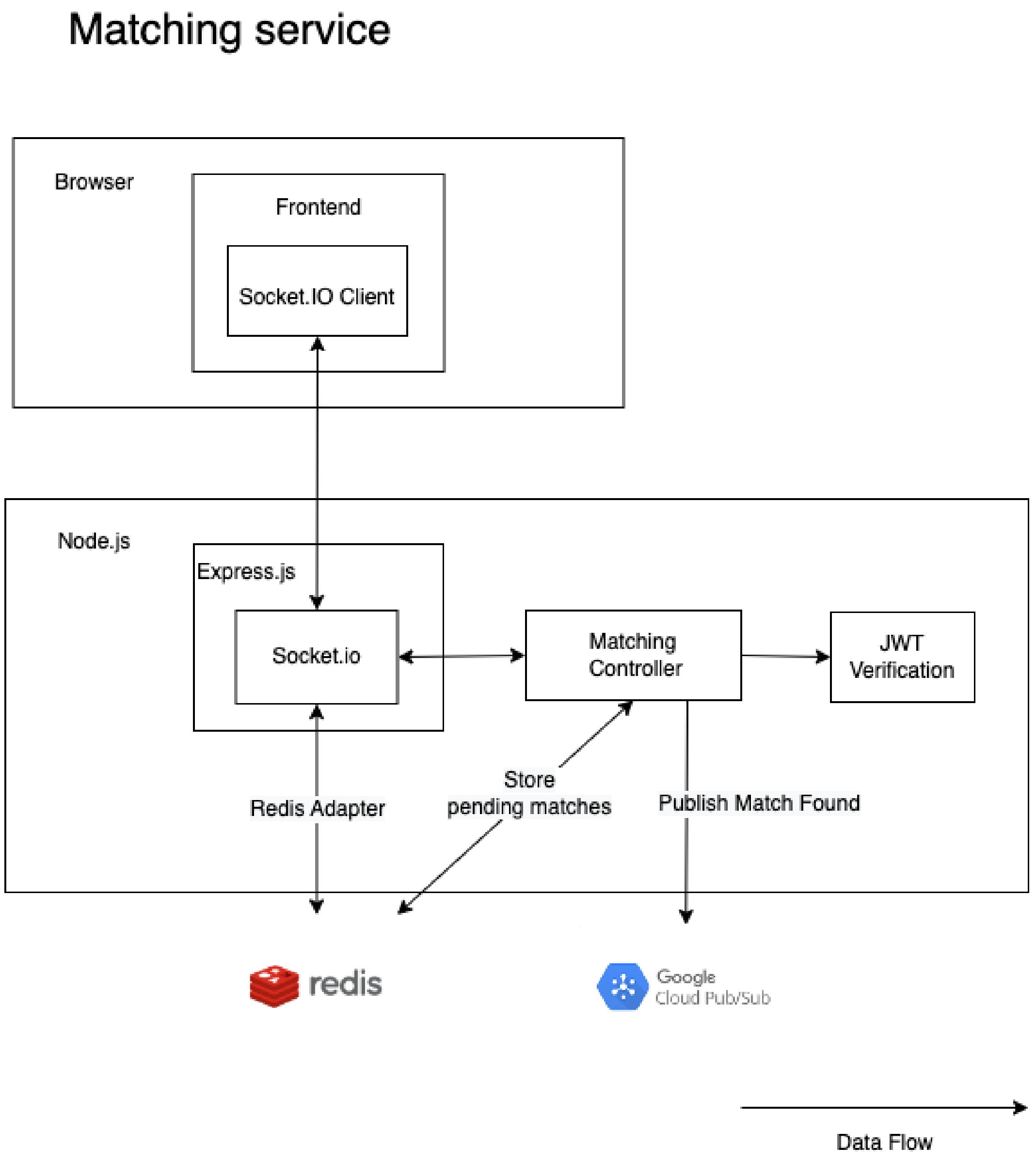


```
export async function getRoomsFromUserID(uid, limit, offset) {
  let matches = await MatchHistoryModel
    .find({ users: uid })
    .offset(offset)
    .limit(limit)
    .sort({ createdAt: -1, updatedAt: -1 });
  return matches;
}
```

As we can see from the code, the Builder pattern allows us to build the query object step by step, using only the required parameters. With this pattern, we no longer need to include a long link of parameters into the constructors, making the code more readable and extensible.

### 5.3.5 Matching Service

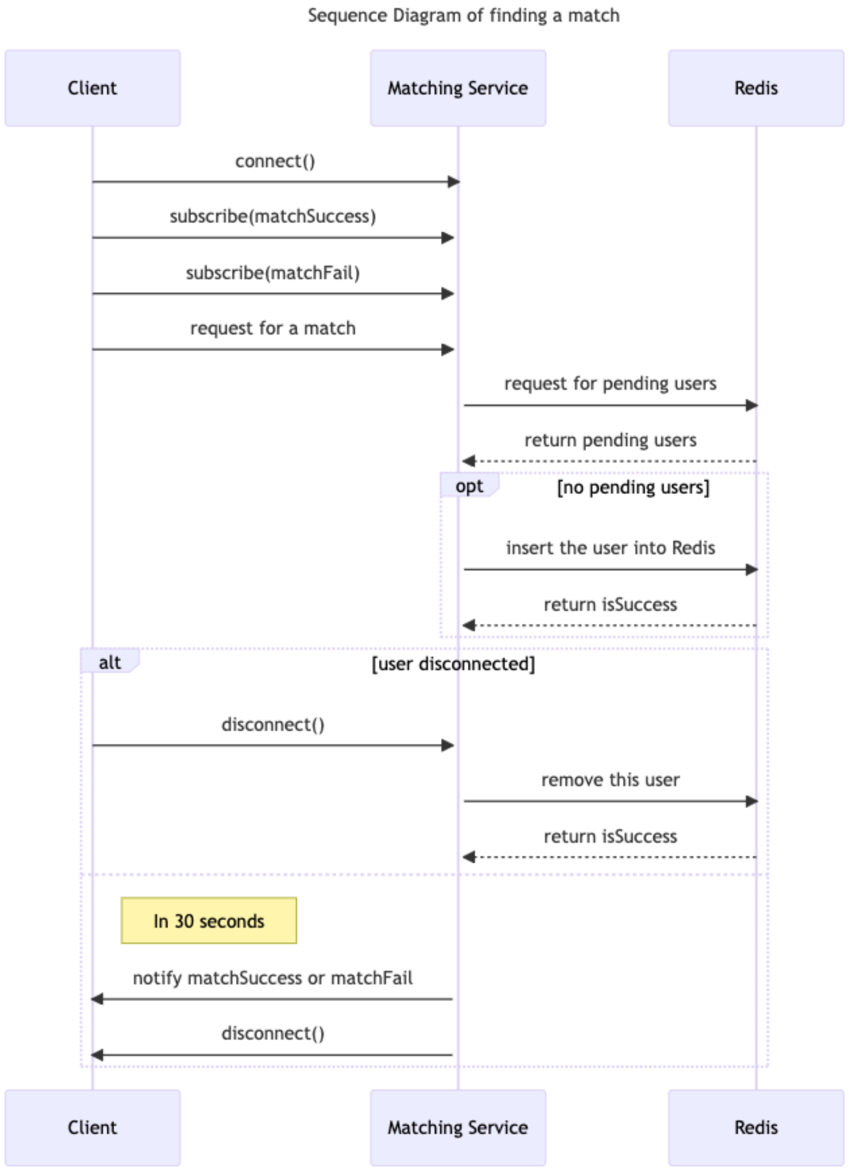
The matching service is set up to be a high performance matchmaker that matches two users based on their selected difficulty.



#### 5.3.5.1 Tech Stack

- NodeJs (Express)
- Socket.IO
- Redis

### 5.3.5.2 Matching Process



Request Flow:

1. User requests for a match in the frontend
2. Frontend establishes a socket connection with the matching service.
3. Once the connection is established, the frontend requests for a match by emitting user information to *match* event. It subscribes to both the *matchSuccess* and *matchFail* events in the meantime.
4. Matching service attempts to find another user with the selected difficulty.

a. If the match is successful, a unique room id is generated and sent to the two users via *matchSuccess* event.
b. If a match cannot be found after 30 seconds, the matching service emits a *matchFail* event to the frontend socket, indicating a failure to find a match.

#### 5.3.5.3 Publishing Match Events on a Google Cloud Pub/Sub Topic

As shared in the Architecture Design, Matching Service publishes “match” events for consumption by Collaboration, Communication and User Service. This is crucial in ensuring users are authorized to be in a room.

The usage of Google CLoud Pub/Sub is another instance of employing Publisher-Subscriber patterns

**Schema of the “match” message to be published**

```
{
 room_id: 'la7xyag17be4x8lg24d',
 difficulty_level: 'beginner',
 username1: 'hello2',
 username2: 'hello',
 user_id1: '633d318e3dfd543b7fce077f',
 user_id2: '633d303c0b8db1ad1de8283a'
}
```

#### 5.3.5.4 Design Decision

##### 5.3.5.4.1 Find Match Communication Protocol: **WebSocket**

| | For | Against |
|---|---|---|
| HTTP | Ease of implementation and every member is comfortable | Concurrent requests may increase the server load, |

|  |  |  |
|---|---|---|
|  | with this protocol<br>Reliable and stable connection protocol | which can lead to server timeout |
| WebSocket | Event-driven protocol, meaning that it is good for real time communication<br>Reduce latency issues by keeping a single and persistent connection open | Need additional logic to handle web socket reconnection<br>More difficult for testing and authentication. |

Decision: We decided to make use of Socket.IO (WebSocket) as our communication protocol to find a match and check if the match is found. This is because it would provide a more real time experience to the user and reduce continuous HTTP(S) requests from the clients.

#### 5.3.5.4.2 Pending Matches Storage: **Redis Cache**

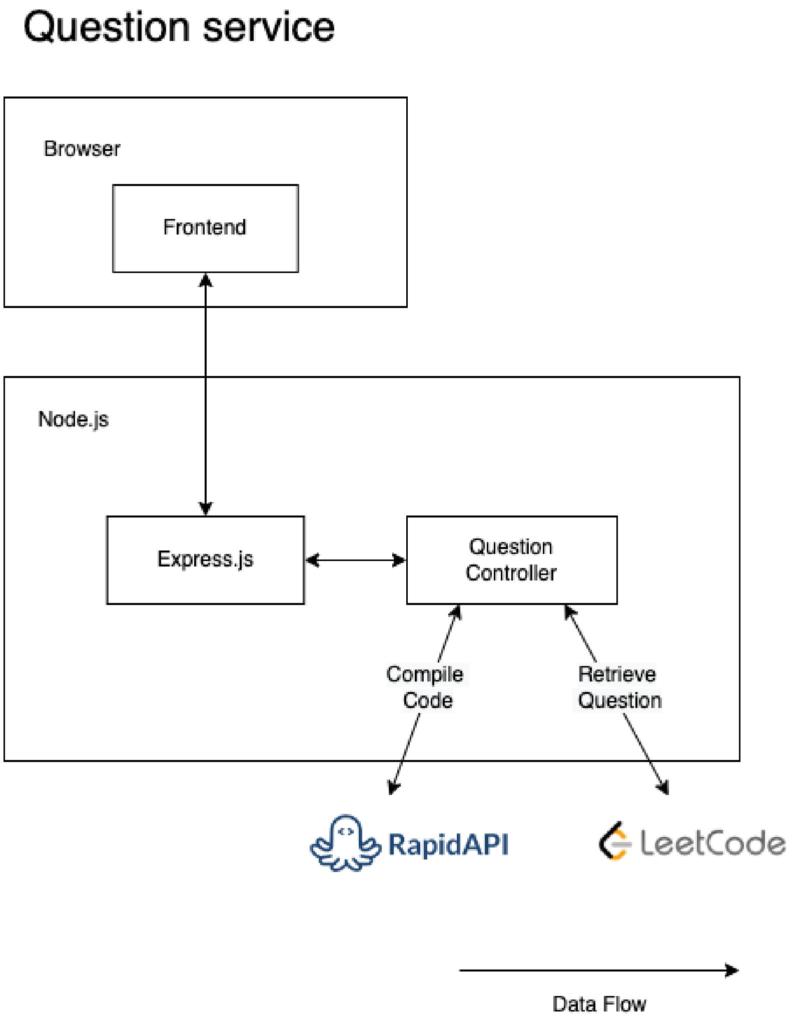
|  | For | Against |
|---|---|---|
| Storing pending matches in SQLite, which is suggested in the development guide | Easier to setup as it does not require any additional software installation or servers because it’s self contained.<br>Lightweight and efficient as it is an embedded data store.<br>Persistent Data | No easy way to scale beyond 1 instance of Matching Service as SQLite is not a server and the performance will not be great even if it works over a network file system.<br>Sqlite does not support concurrency, meaning that only one user can write to the database at a time. |
| Storing pending matches in In-Memory Databases (like Redis Cache) | Allows more than 1 instance of Matching Service as all instances can access a single source of pending matches storage.<br>Fast as memory access is | No data persistence.<br>May not be as fast as SQLite if there is only one server instance. |

| | | |
|---|---|---|
| | several orders of magnitude faster than random disk I/O. | |

Decision: The final decision is to use Redis Cache because we value the scalability of our product very highly and have two Matching Service instances by default in the deployment cluster. Redis is the clear winner here as it allows multiple servers to communicate with each other through Redis over the network.

## 5.3.6 Question Service

The question microservice is responsible for providing coding questions of different difficulties and executing the code submitted by the user.



### 5.3.6.1 Tech Stack

- NodeJs (Express)
- Rapid API

### 5.3.6.2 API Endpoints

1. Retrieve a list of questions with selected difficulty

```
GET /api/question/questions?difficulty={difficulty}&page={page}&pageSize={pageSize}
```

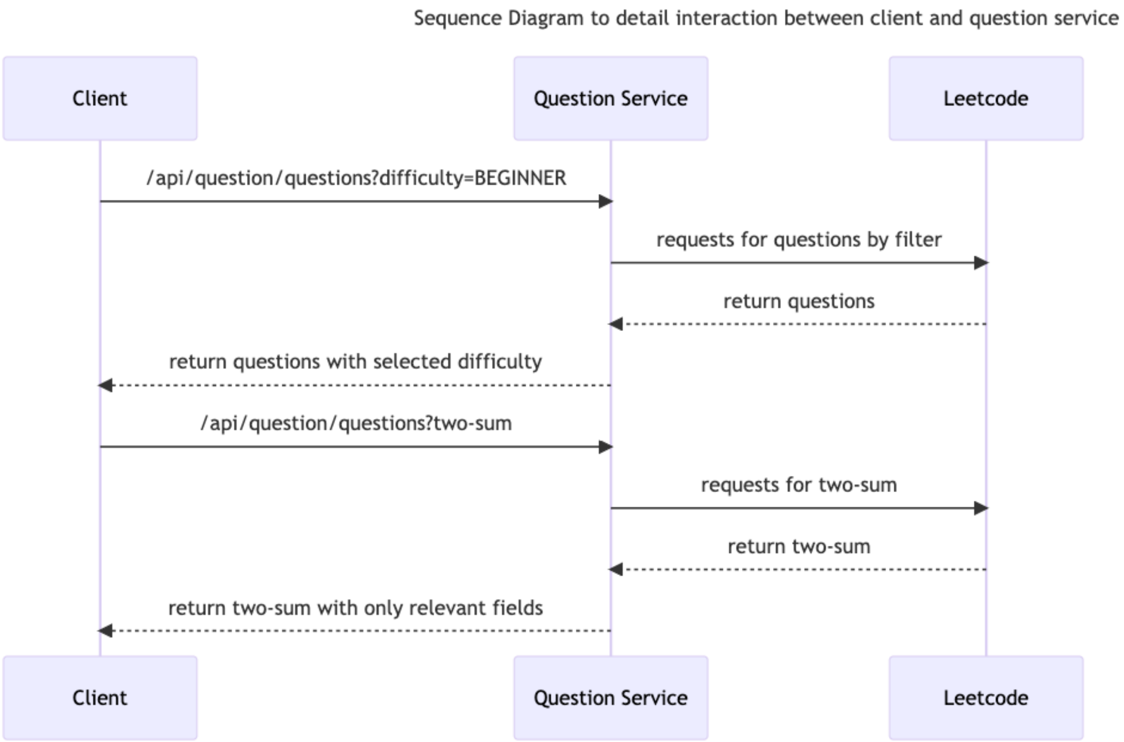
2. Retrieve a question with detailed information

```
GET /api/question/questions?titleSlug={titleSlug}
```

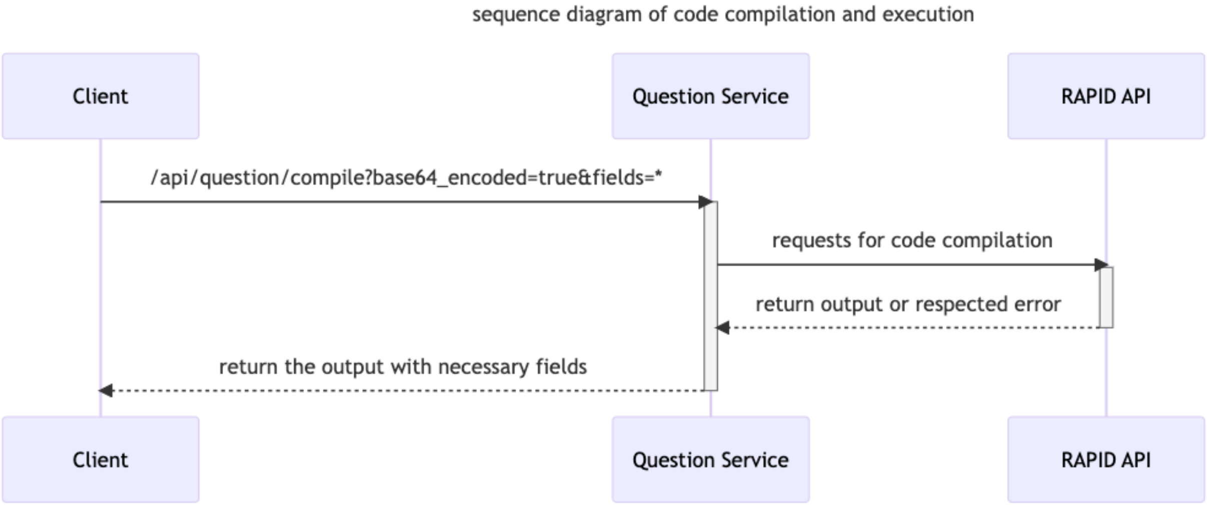
3. Compile and execute the code

```
POST /api/question/compile?base64_encoded=true&fields=*
```

### 5.3.6.3 Retrieve Question Process


Sequence Diagram to detail interaction between client and question service
Client
Question Service
Leetcode
/api/question/questions?difficulty=BEGINNER
requests for questions by filter
return questions
return questions with selected difficulty
/api/question/questions?two-sum
requests for two-sum
return two-sum
return two-sum with only relevant fields
Client
Question Service
Leetcode

### 5.3.6.4 Code Compilation and Execution Process


sequence diagram of code compilation and execution
Client
Question Service
RAPID API
/api/question/compile?base64_encoded=true&fields=*
requests for code compilation
return output or respected error
return the output with necessary fields
Client
Question Service
RAPID API

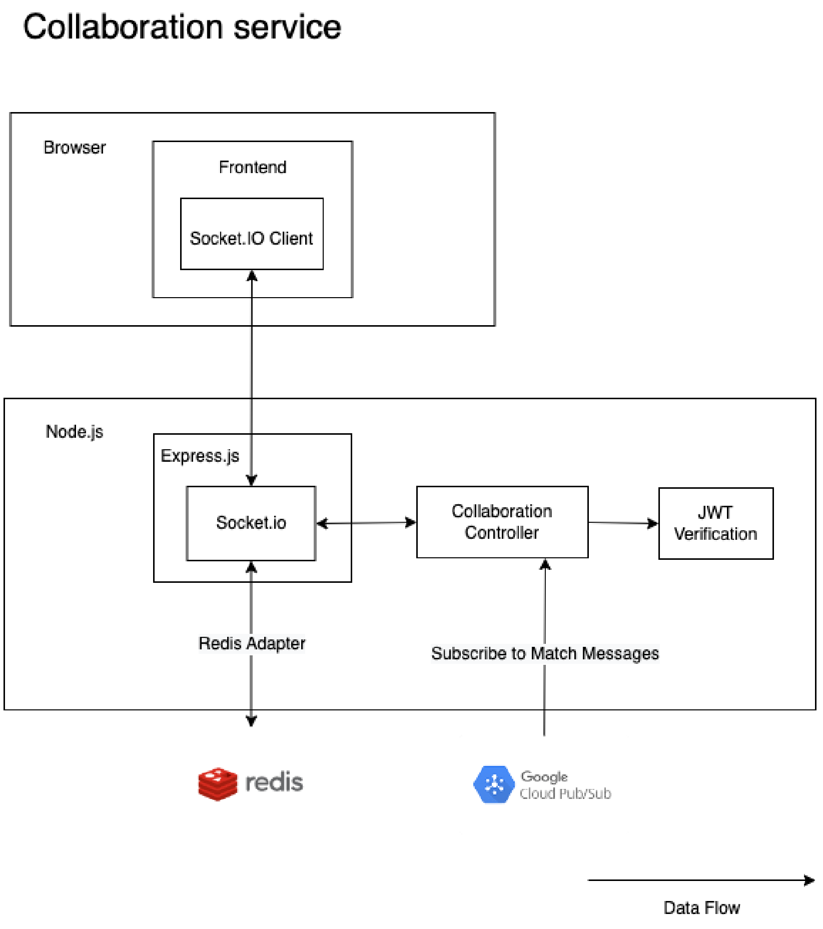
## 5.3.7 Collaboration Service

The collaboration service is responsible for enabling code collaboration by syncing both client’s views of the code editor.

### 5.3.7.1 Tech Stack

- NodeJs (Express)
- Socket.IO
- Redis

### 5.3.7.2 Component Design

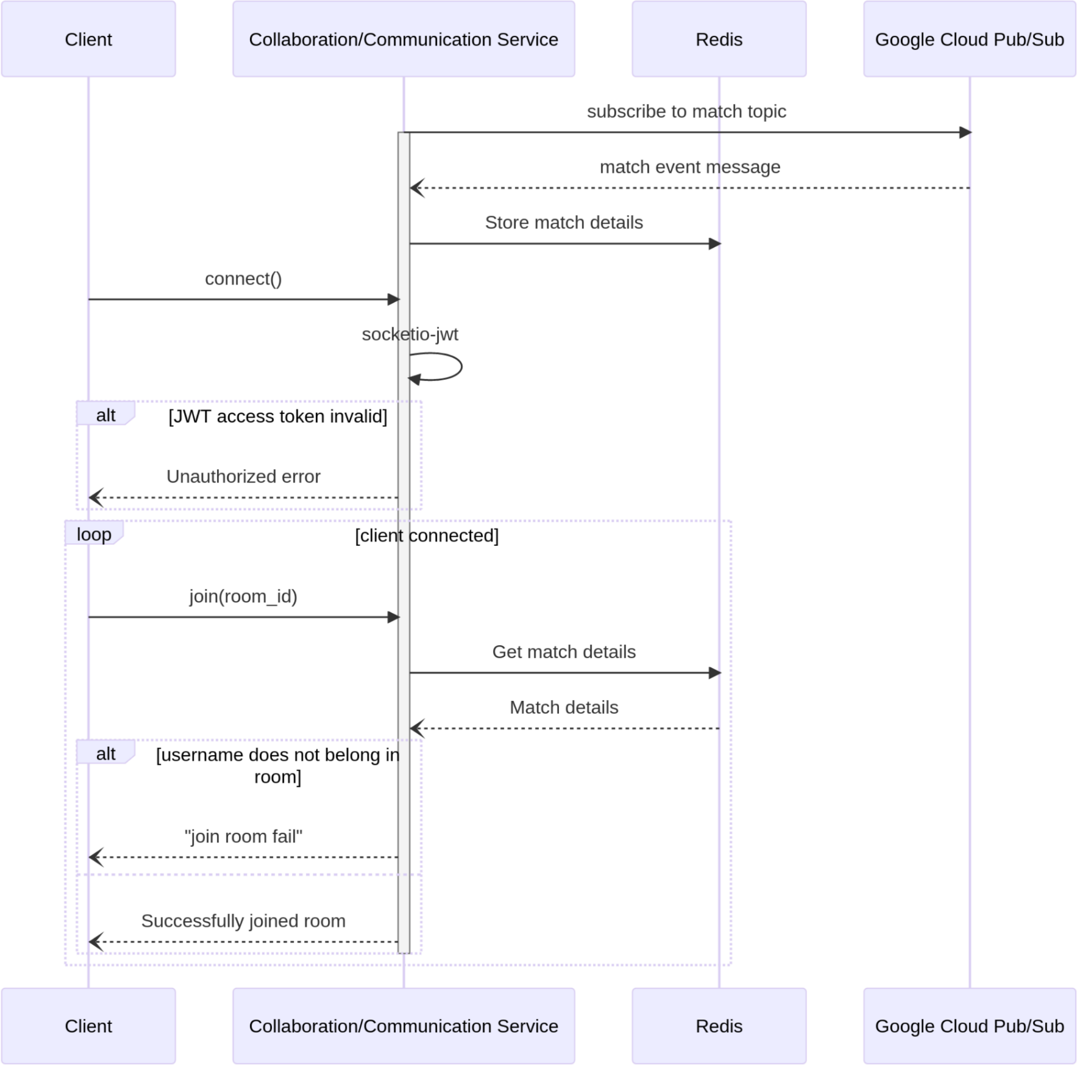


### 5.3.7.3 Security of Communication Channel

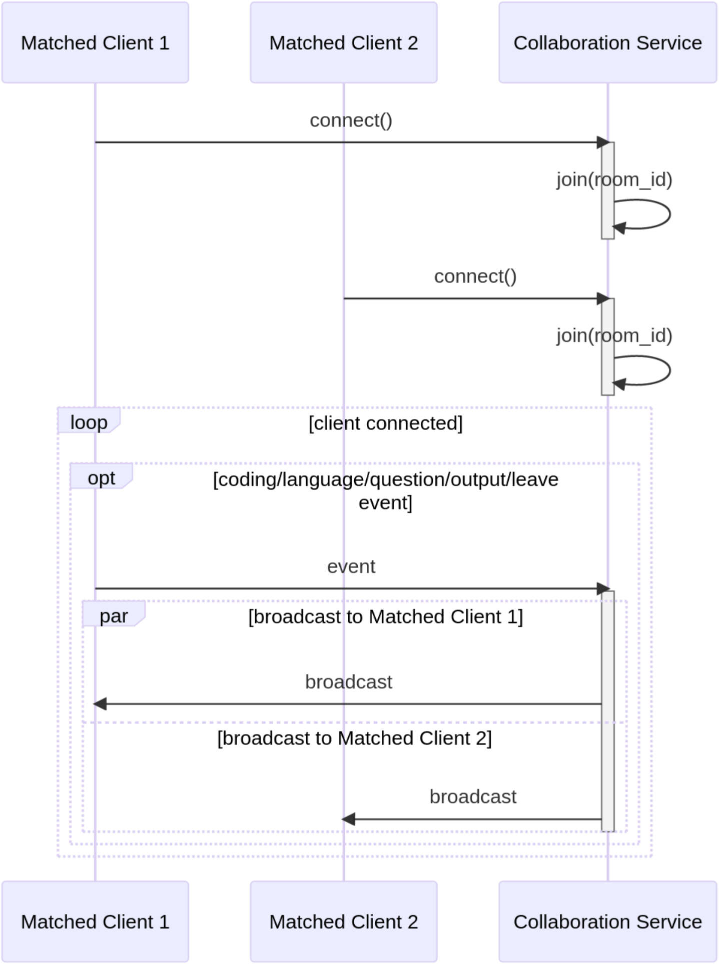
Both Collaboration and Communication Service are designed with the same 2 step authentication for socket connections.

- Utilize the socketio-jwt library as a middleware to decrypt and verify Socket.io connections. Client sockets include the jwt access-token issued by User Service when establishing a connection with the server sockets in Collaboration/Communication Service.
- Listen to Match messages published by Matching Service on Google Cloud Pub/Sub topics and store match details in corresponding Redis. When client sockets fire the “join room” event, Collaboration/Communication Service queries Redis checking if the username belongs to room_id and denies room join if false.

### 5.3.7.4 2-Step Authentication of Socket Connections



### 5.3.7.5 Code Collaboration Process



## 5.3.8 Communication Service

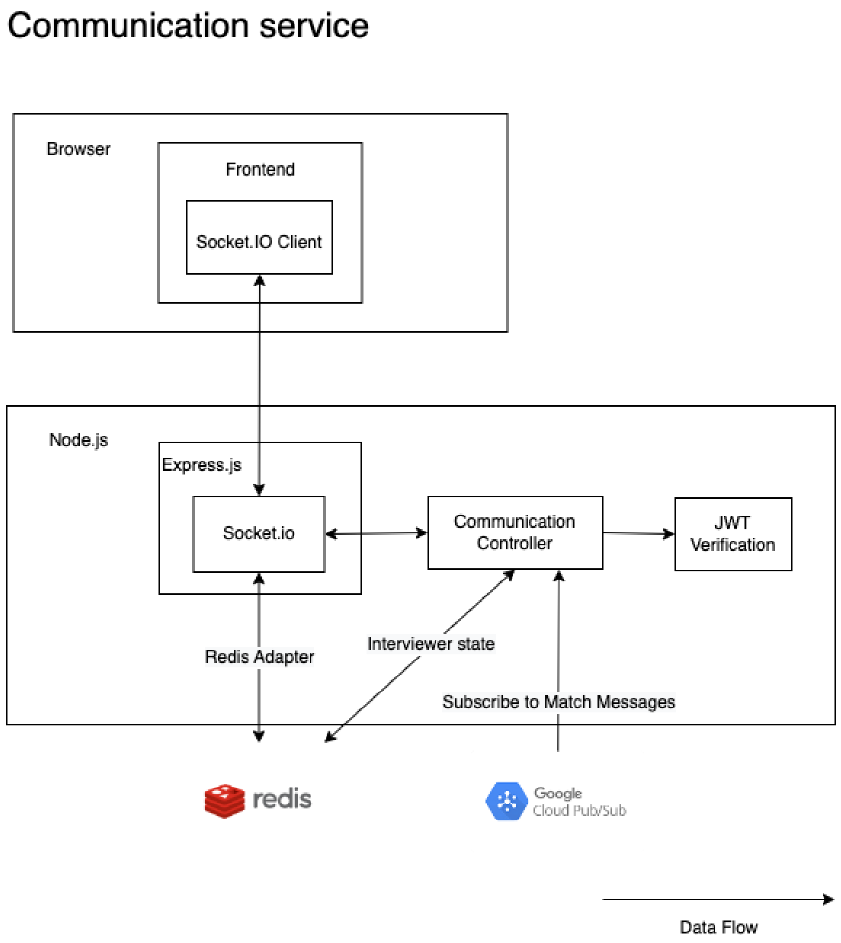
Communication service is responsible for Live-Chat functionality between 2 users in the same room. The following is an exhaustive list of the features it provides:

- Real-time Chat(Instant Messaging)
- Allowing users to take on Interviewer/Interviewee roles

### 5.3.8.1 Tech Stack

- NodeJs (Express)
- Socket.io
- Redis
- Google Cloud Pub/Sub

### 5.3.8.2 Architecture Design



#### 5.3.8.4 Security of Communication Channel

As explained in Collaboration Service, Communication Service utilizes the same 2 step authentication for socket connections.

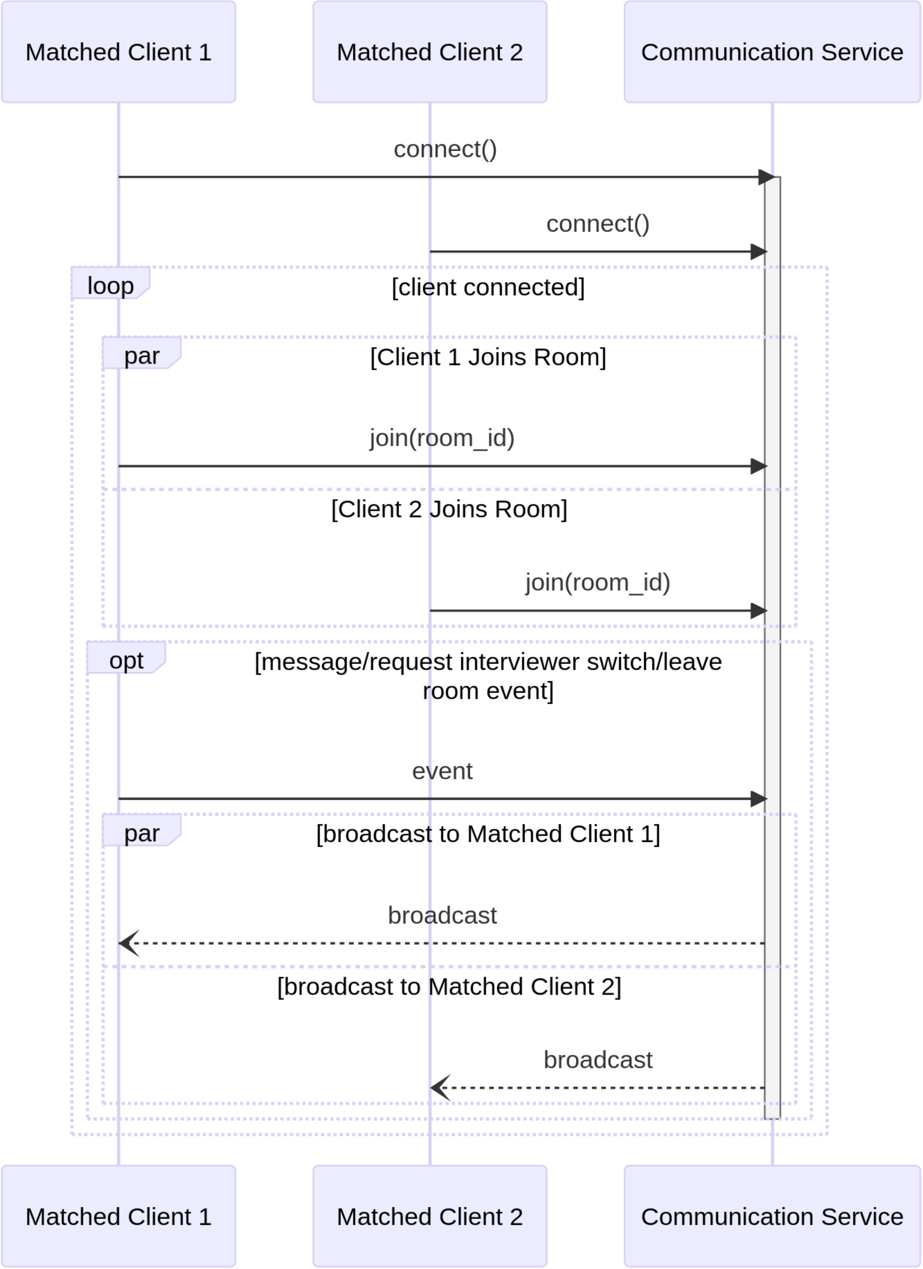
### 5.3.8.5 Real-time Chat (Instant Messaging)

#### 5.3.8.5.1 Socket Events

Communication Service uses Socket.io Rooms as channels for clients to communicate on. Rooms are identified using room_id returned by Matching Service.

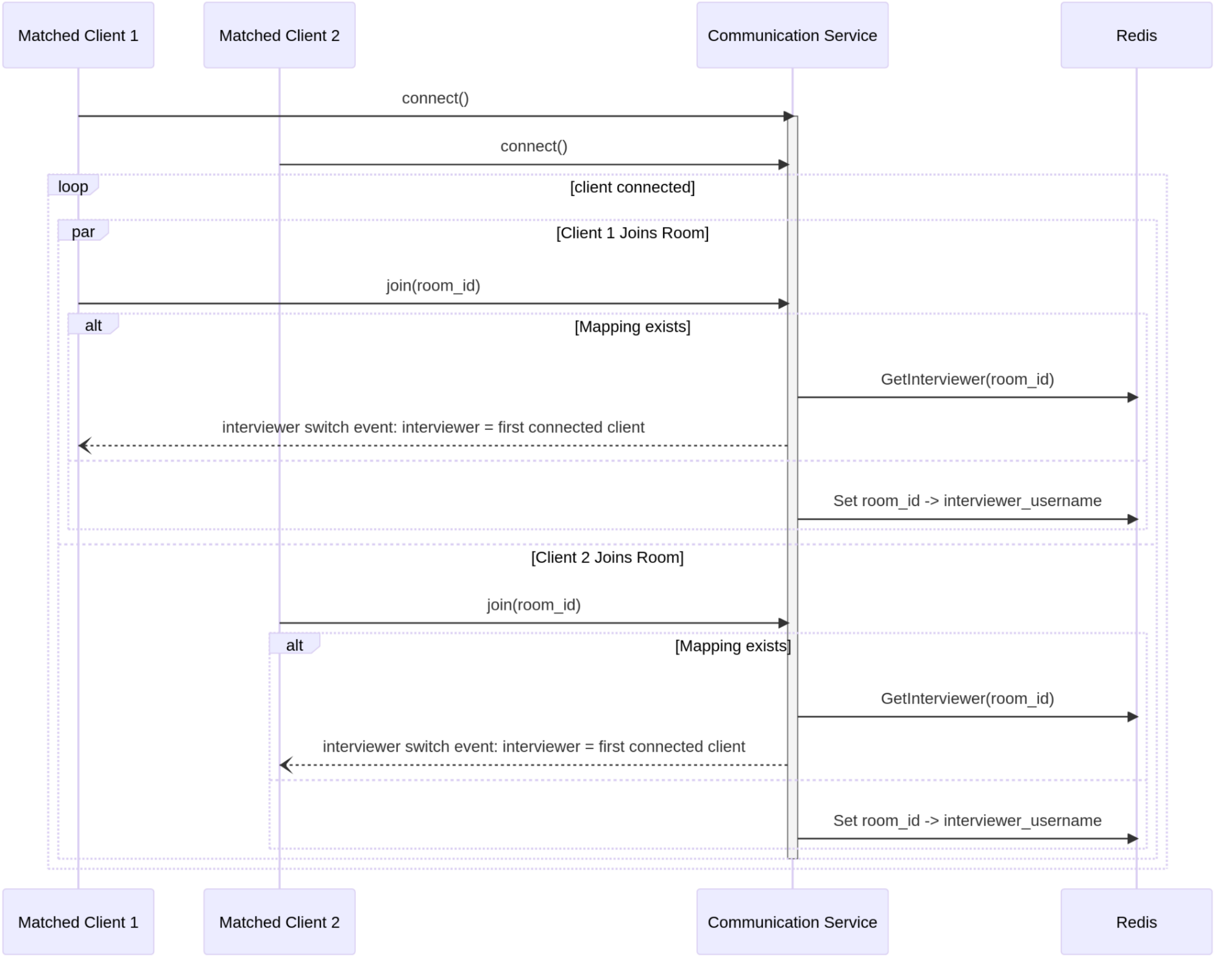
| **Socket Event** | **Purpose** |
|---|---|
| “join room” | Emitted by client sockets to join rooms identified by room_id(From matching service) |
| “message” | Emitted by client sockets to send a message to other client sockets in the same room |
| “request interviewer switch” | Emitted by client sockets who request to become the interviewer |
| “interviewer switch event” | Emitted by server sockets to relay requests to become the interviewer.<br>Emitted by server sockets to assign interviewer roles to client sockets upon socket connection. |
| “leave room” | Emitted by client sockets before they leave the room. |

5.3.8.5.2 Live Chat Process



### 5.3.8.6 Allowing Users to Take On Interviewer/Interviewee Roles

#### 5.3.8.6.1 Interviewer Role Assignment on Initial Connection



Each room has two matched clients and one of them assumes the role of interviewer, while the other assumes the role of interviewee.

As seen in the sequence diagram above, upon establishment of connection, the client socket fires join(room_id). The First client to fire join(room_id) will be assigned the interviewer role. This mapping of room_id -> interviewer_username is stored in Redis. As the two join(room_id) events are executed in parallel, we ensure to fire a broadcast of the interviewer switch event only when the mapping already exists in Redis.

## 5.3.9 Frontend

The frontend focuses on enhancing the user interactions and user experience (UI/UX)

### 5.3.9.1 Tech Stack

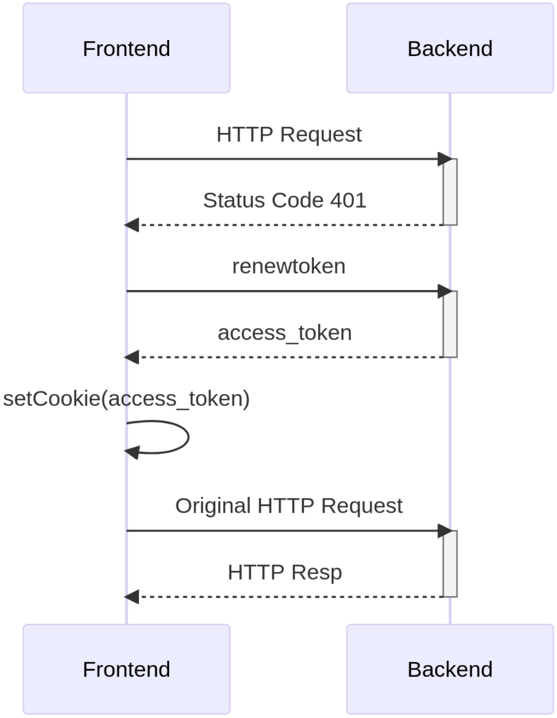
- ReactJS
- HTML & CSS
- Material UI

### 5.3.9.2 Design Patterns

While not explicitly expressed, the frontend, written in React does offer some elements of the Observer patterns. With the use of state (observable), and useEffect (observers), we are able to re-render parts of the view when there is a change in state.

As a Single Page Application (SPA), it offers some elements of MVC as well. The pages that serve data work as the view, while buttons and inputs serve as controllers. Lastly, the states that the SPA keeps track of are similar to the model in MVC.

### 5.3.9.3 Authenticated HTTP Request Process



When the frontend sends a HTTP request to a backend endpoint that is authenticated, and receives a 401 status code, we use axios interceptors to handle this case. The frontend would request for a new access token from user service, then set the access token to the cookie, and retry the HTTP request, which would work successfully now.

#### 5.3.9.4 Code Editor Choice

We chose Code Mirror as the code editor of our application because it is suitable for collaborative editing. Unlike other editors like Monaco Editor, Code Mirror provides useful information about what triggers the changes in the editor content. With this information, the frontend only needs to post the changes to the collaboration room when these changes are caused by user events.

Besides, Code Mirror also supports many useful extensions such as syntax highlighting, editor theming and code autocompletion based on the selected programming language. We believe that these features will help users perform better by allowing them to configure a comfortable coding environment.

#### 5.3.9.5 User Considerations

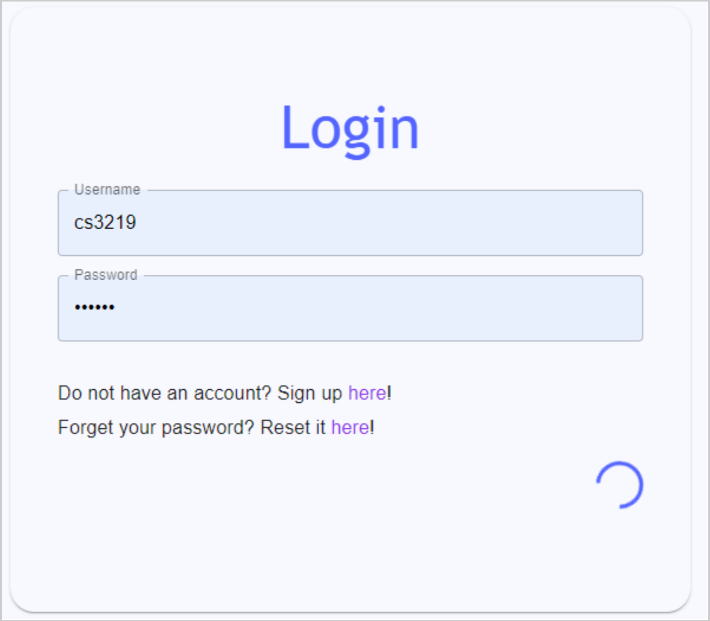
User considerations are highly relevant to the usability of the product. This application is designed with the objective to fulfill NFR4.3 and NFR4.5, so as to ensure that it is easy to use and intuitive even for new users. Below are some of the pointers that we took note of when designing the application's UI/UX.

##### 5.3.9.5.1 Addition of System State Dialogs

For PeerPrep, we offer informative feedback to the users in the form of dialogs and error messages. The state of the system should be clearly visible to the user in an unambiguous form. Following this design guideline, we made sure that the dialogs displayed are not only user-friendly, but also clearly conveys the state of the system, such as having loading states (Figure 1.1), simple and concise error messages (Figures 1.2 and 1.4) and success messages (Figure 1.3).

The clear presentation of these system states help reduce uncertainty for the user by explaining what is happening or preparing them for what is coming up next when using the application[3].

[3] https://pencilandpaper.io/articles/ux-pattern-analysis-loading-feedback/


Login

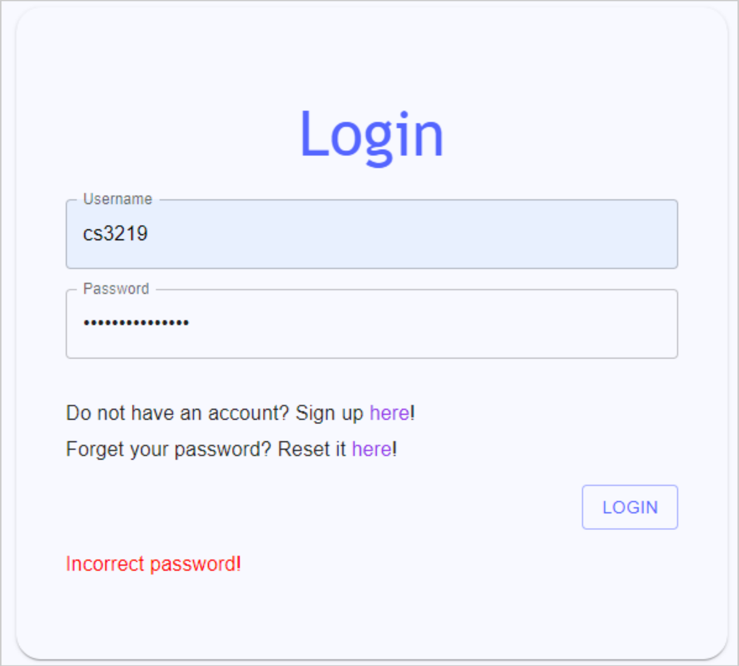
Username: cs3219

Password: ••••••

Do not have an account? Sign up here!

Forget your password? Reset it here!

Figure 1.1: “Loading” State


Login

Username: cs3219

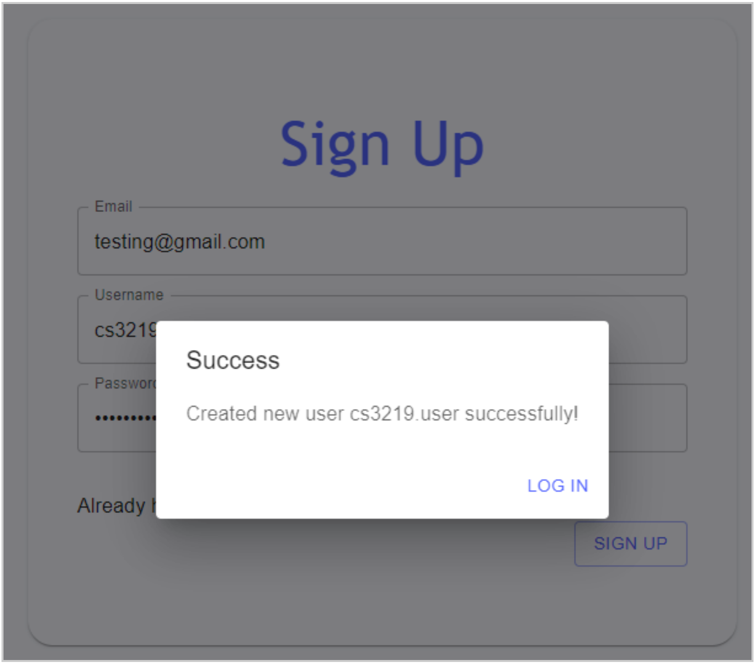
Password: •••••••••••••••

Do not have an account? Sign up here!

Forget your password? Reset it here!

Incorrect password!

Figure 1.2: “Incorrect Password” Error Message



Sign Up

Email: testing@gmail.com

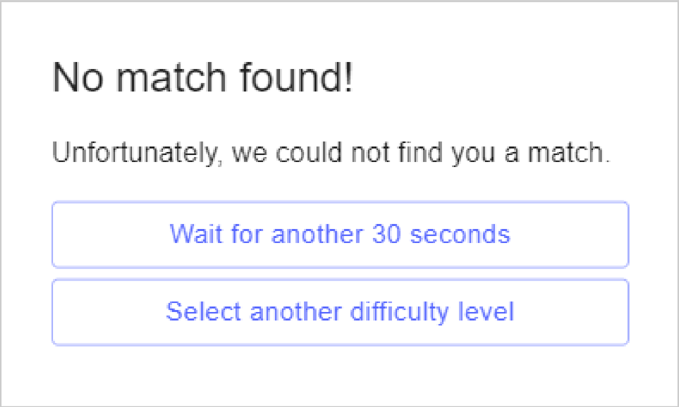
Username: cs3219

Password: ••••••••

Success

Created new user cs3219.user successfully!

Figure 1.3: “Success” Sign-up Message



No match found!

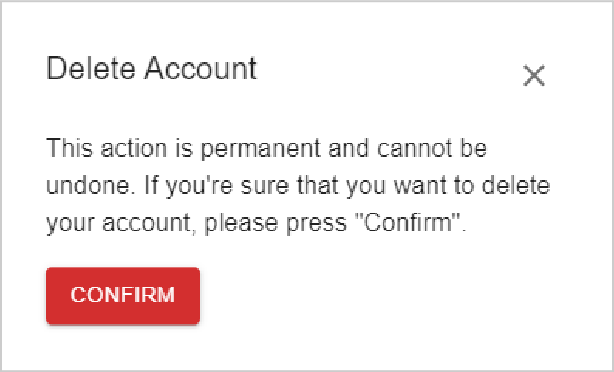
Unfortunately, we could not find you a match.

Figure 1.4: “No match found!” Dialog

### 5.3.9.5.2 Permit Easy Reversal of Actions

In PeerPrep, we added confirmation dialogues and buttons for users to revert their actions quickly and effectively, such as deletion of accounts (Figure 2.1), logging out (Figure 2.2), canceling of matching processes (Figure 2.3) and swapping of user roles in the coding room (Figure 2.4).

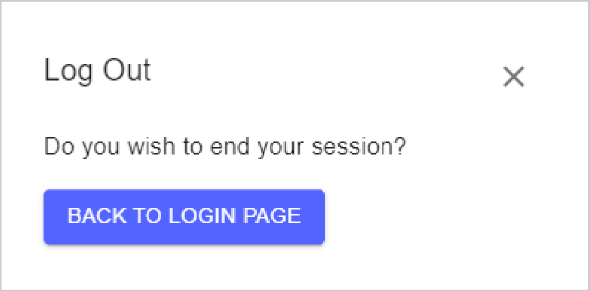
This adheres to Shneiderman's Eight Golden Rules of Interface Design[4], more specifically, the rule to permit easy reversal of actions. According to Shneiderman, this helps users to relieve anxiety. Since they know that actions can be undone, they will be more encouraged to explore unfamiliar options, hence enhancing user efficiency.



Delete Account

This action is permanent and cannot be undone. If you're sure that you want to delete your account, please press "Confirm".

Figure 2.1: “Delete Account” Confirmation Dialog

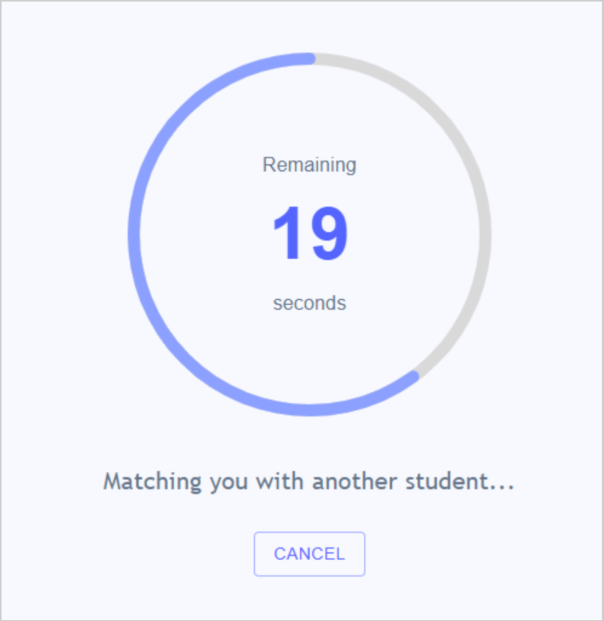


Log Out

Do you wish to end your session?

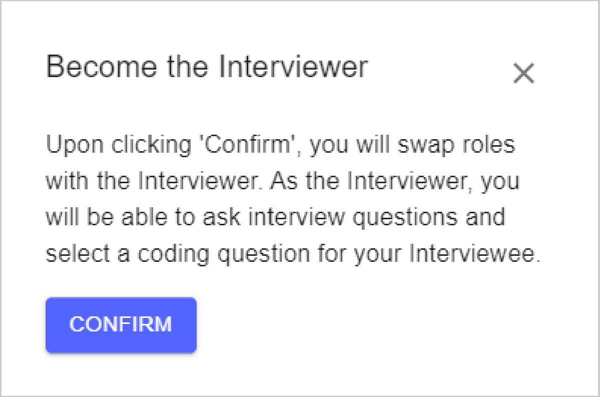
Figure 2.2:  “Log Out” Confirmation Dialog

[4] https://www.interaction-design.org/literature/article/shneiderman-s-eight-golden-rules-will-help-you-design-better-interfaces



Matching you with another student...

Figure 2.3: “Cancel” Matching Process



Become the Interviewer

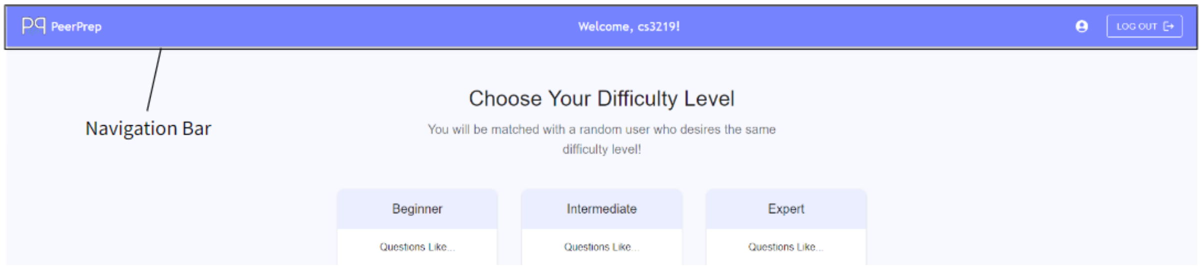
Upon clicking 'Confirm', you will swap roles with the Interviewer. As the Interviewer, you will be able to ask interview questions and select a coding question for your Interviewee.

Figure 3.4: “Role-swapping” Confirmation Dialog

#### 5.3.9.5.3 Consistent Visual Elements

Visual elements are consistent throughout the application. For PeerPrep, we ensured that every page follows the same theme, such as the font style, icon style and color scheme used in every page of the application. We also followed a similar design layout for every page, such as having the position of navigation bar fixed at the top of every page (Figures 3.1 and 3.2).

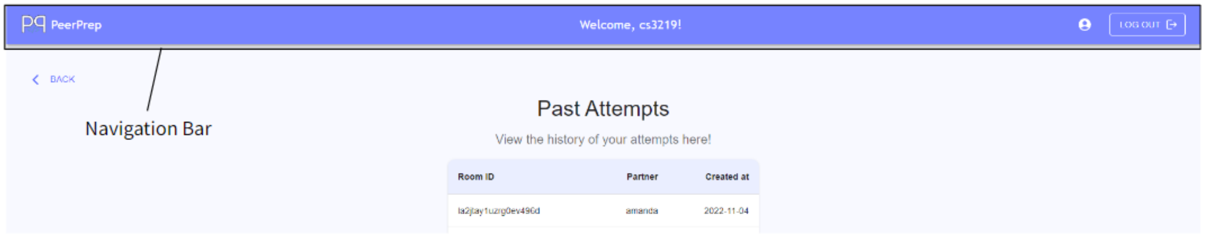
Consistency makes the entire application more cohesive and put together. This will be helpful for users as they will not have to learn new representations or toolsets whenever they navigate from page to page, and makes it easier for them to navigate around the pages[5].



Choose Your Difficulty Level

You will be matched with a random user who desires the same difficulty level!

Figure 3.1: Landing Page



Past Attempts

View the history of your attempts here!

| Room ID | Partner | Created at |
|---|---|---|
| la2jtay1uzrg0ev490d | amanda | 2022-11-04 |

Figure 3.2: Past Attempts Page

### 5.3.9.5.4 Usage of Familiar Language/Icon

In PeerPrep, we also made sure not to deviate from design standards and conventions as much as possible. Throughout the application, we made use of familiar language and globally understood icons to represent general functions.  Some examples include using a 'User' icon at the navigation bar to represent user settings (Figure 4.1), a chat icon to represent the messenger function (Figure 4.2), an exit icon to represent ending a session (Figure 4.3), and  a send icon to represent the function of sending a message.

Besides using icons to capture the user's attention, iconography also helps to remove ambiguity of the functionalities and in turn helps users locate the functionalities easily[6]. This will help eliminate any confusion when users use the application, and allow them to easily carry out general tasks such as managing their settings, log out etc.

[5] https://www.interaction-design.org/literature/article/principle-of-consistency-and-standards-in-user-interface-design

[6] https://medium.com/successivetech/significance-of-iconography-2e2dc7e5611a

Figure 4.1: Settings Icon

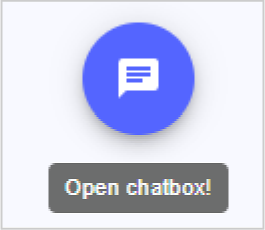


Figure 4.2: Messenger Icon

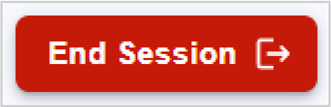


Figure 4.3: Exit Icon

Figure 4.4: Send Icon

#### 5.3.9.5.5 Enhancements

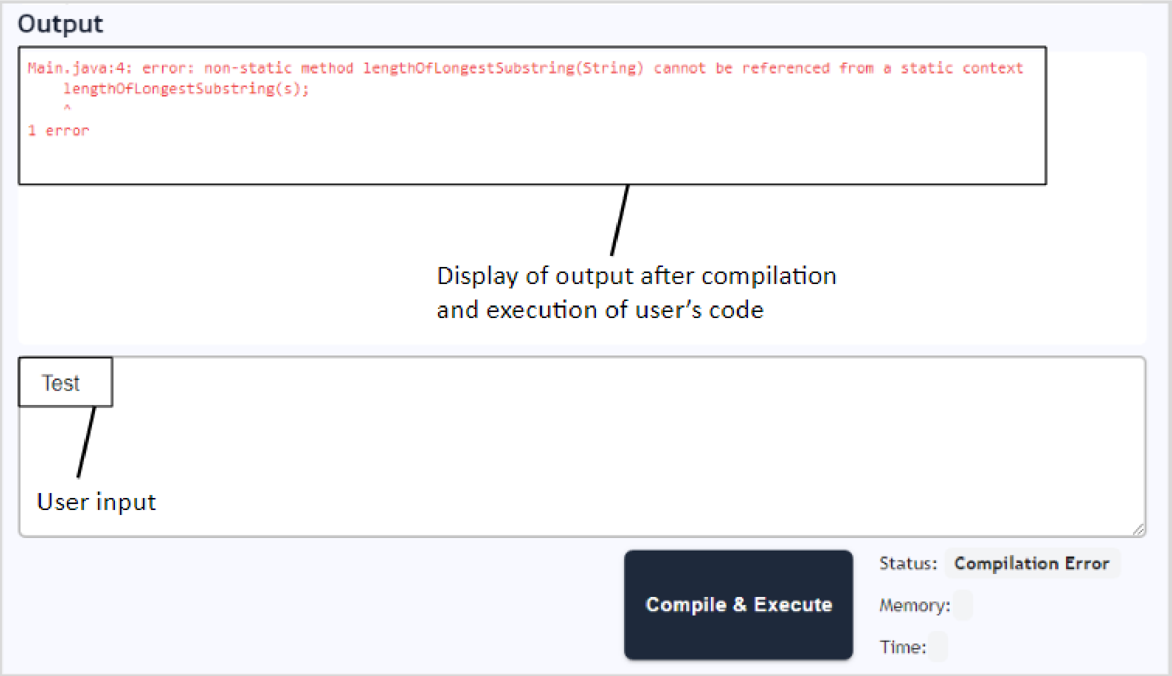
On top of the must-have functionalities, we decided to enhance PeerPrep by adding on some features, boosting user's experience when using the application. Some notable additions include:

- Instead of coding on a simple text editor, users will be able to compile and execute their code. More specifically, users will get to input test cases in the input (Figure 5.1). The compilation and execution feature accommodates an exhaustive list of commonly-used coding languages, which users can select from (Figure 5.2). This is linked to FR4.2 and FR4.3, which will enhance the user's experience since they will be able to test the correctness of their code to a certain extent.

  In addition, users will be able to select the code editor theme from a dropdown provided (Figure 5.2), which includes both light and dark themes. With the help of color schemes, users can easily highlight and distribute the context according to their likeness and preferences, or even reduce eye strains by using dark themes. This is said to increase the user's productivity[7], hence boosting user experience.

- To make the interview session more realistic, users with the "Interviewer" role will be provided with a list of common interview questions asked during technical interviews (Figure 5.3). This is linked to FR6.16. Since users may lack interview experiences, the user with the "Interviewer" role may not know the right questions to ask during the session. This list of questions can serve as a guide to the user with the "Interviewer" role. On the receiving end, the user with the "Interviewee" role will benefit, since the questions posed to him/her will be more professional, hence allowing the user to better prepare himself/herself for an actual technical interview.

- The users will be able to navigate directly to the particular question chosen on the Leetcode website if the other user has disconnected, by clicking on the link provided in the pop-up notification (Figure 5.4). This is linked to FR4.4. Considering cases where users get disconnected unexpectedly during a session before the other user completes the question, this may cause frustration to the user who has yet to complete the question, reducing the productivity of the session. To maximize the productivity of the session in such cases, the user can continue the session alone by clicking on the link and continue coding on the Leetcode website itself.



```
Main.java:4: error: non-static method lengthOfLongestSubstring(String) cannot be referenced from a static context
    lengthOfLongestSubstring(s);
    ^
1 error
```

Display of output after compilation and execution of user's code

User input

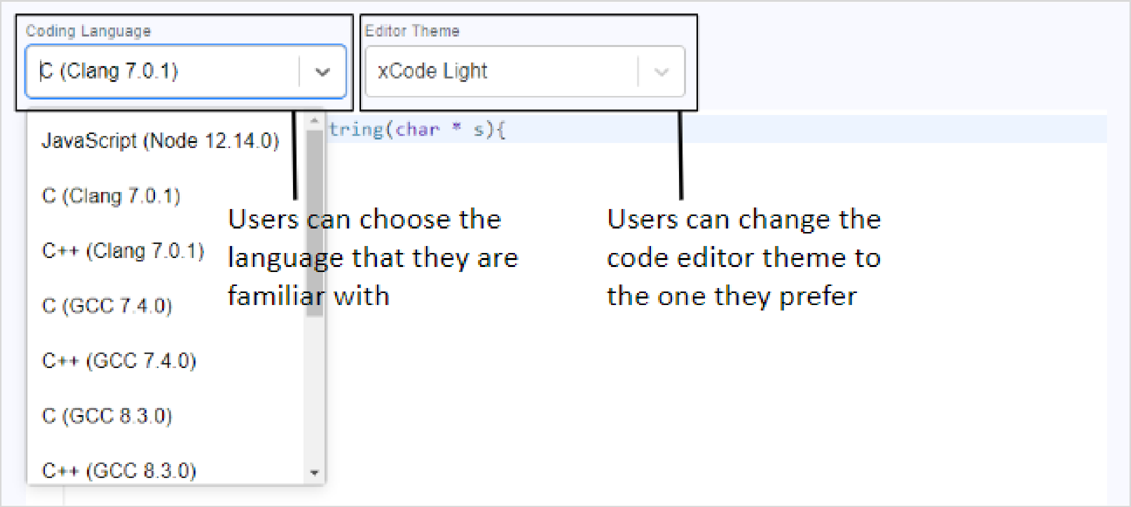
Status: Compilation Error

Memory:

Time:

[7] https://blog.ndepend.com/6-reasons-visual-studio-theme-affects-productivity/

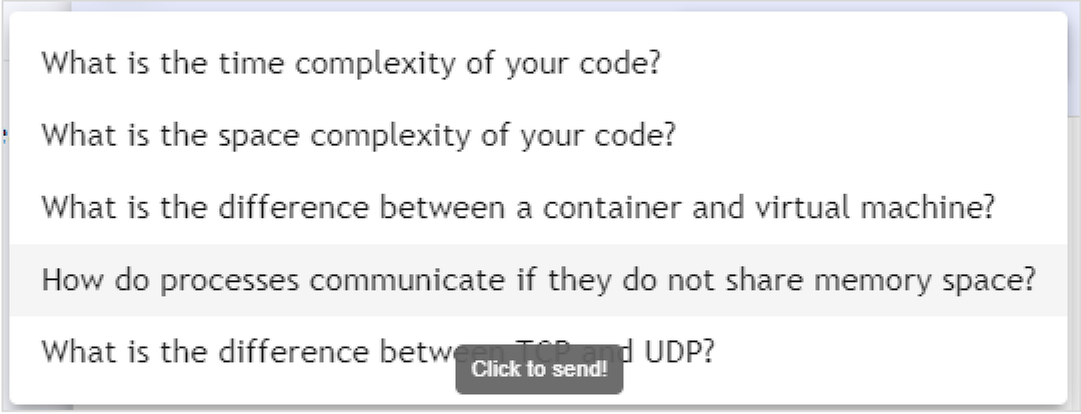
Figure 5.1: Display of Output and Input After Execution



Users can choose the language that they are familiar with

Users can change the code editor theme to the one they prefer

Figure 5.2: Dropdown of Coding Language and Editor Theme



What is the time complexity of your code?

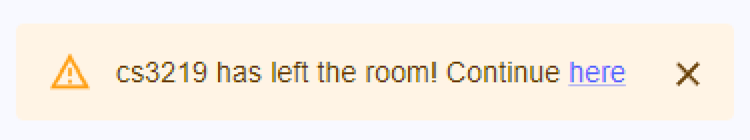
What is the space complexity of your code?

What is the difference between a container and virtual machine?

How do processes communicate if they do not share memory space?

What is the difference between TCP and UDP?

Figure 5.3: List of Common Interviewer Questions



cs3219 has left the room! Continue here

Figure 5.4: Disconnection Pop-up Notification

### 5.3.9.6 UI Design Decisions

#### 5.3.9.6.1 Modification of Coding Page

After gaining feedback from our mentor during Milestone 2's check-in, we decided to modify the layout of our room page.

As seen from the mock-up of the coding page before modification (Figure 6), the question display, code editor, output window and chatbox are placed side-by-side in a column layout. This may be frustrating to users as they may constantly find the need to scroll back and forth to refer to the question while coding, or to refer to the question/code while interacting with the other collaborator.

To resolve this issue, we decided to place the question display and code editor side-by-side in a row layout, and the code editor and output window side-by-side in a column layout (Figure 7). We also decided to modify the chat window, which was previously fixed at the bottom of the page, into a draggable pop-up chat box (expanded window shown in Figure 8), offering users the choice to show or hide the chat window, and position it in any part of the page they deem fit. With the additional flexibility and convenience/intuitiveness, it will be easier for users to attempt the coding question in a collaborative mode, enhancing the efficiency of the coding session and user experience.

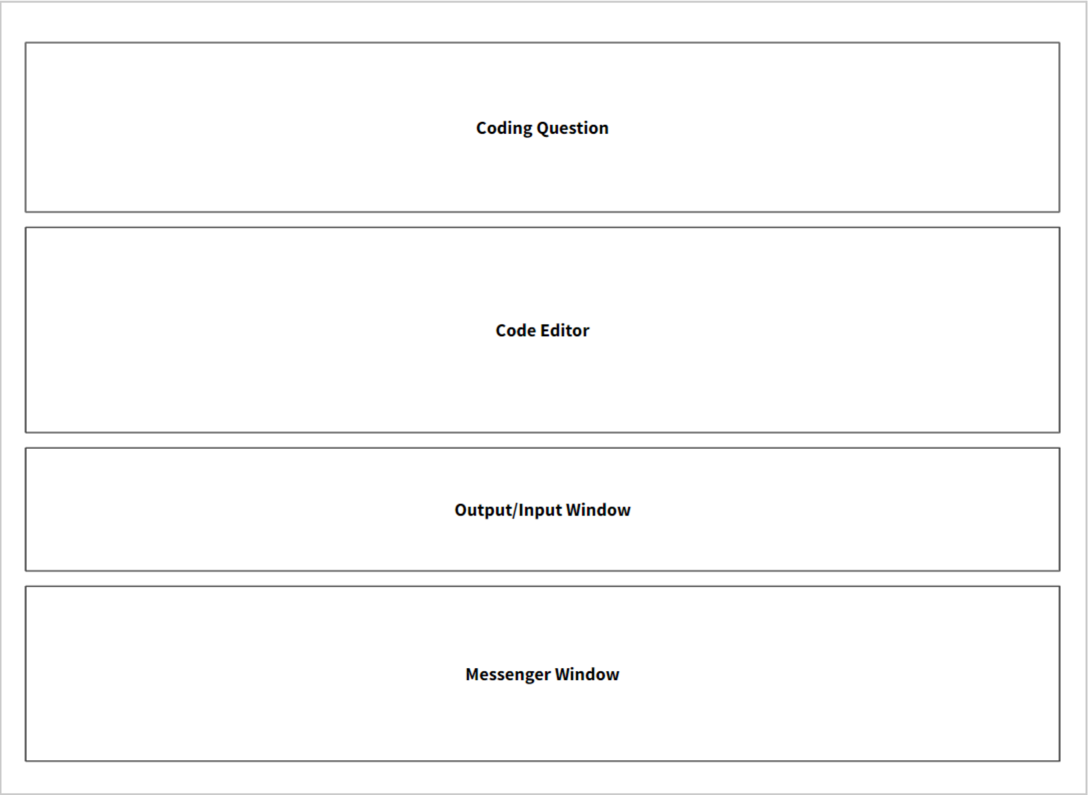


Figure 6: Mock-up of Coding Page Layout Before Modification

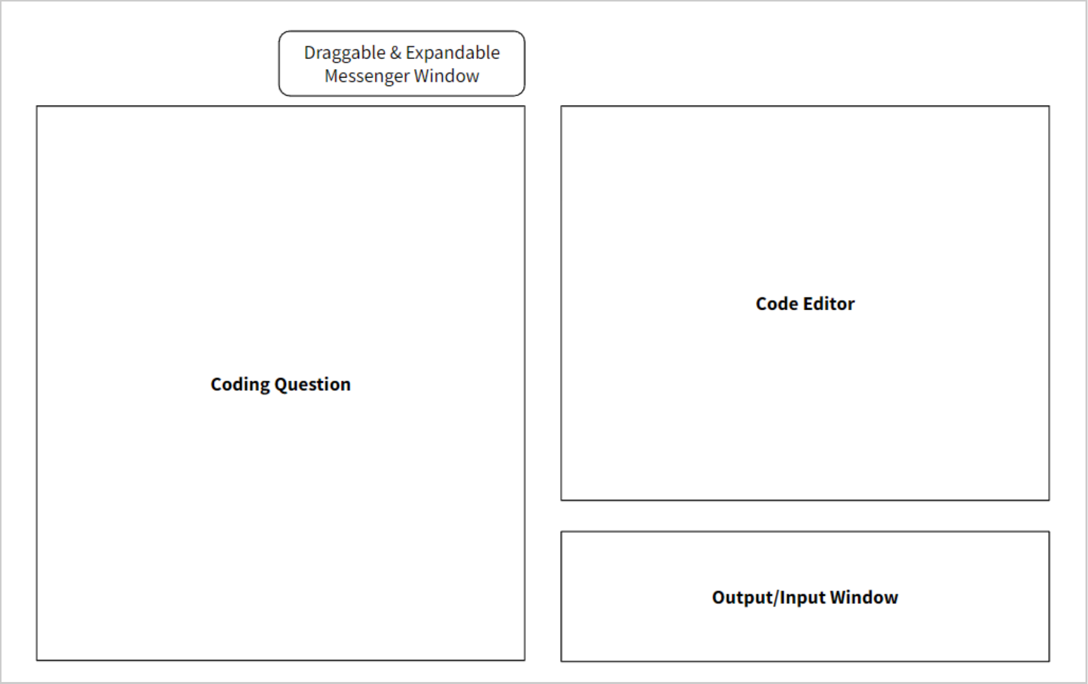
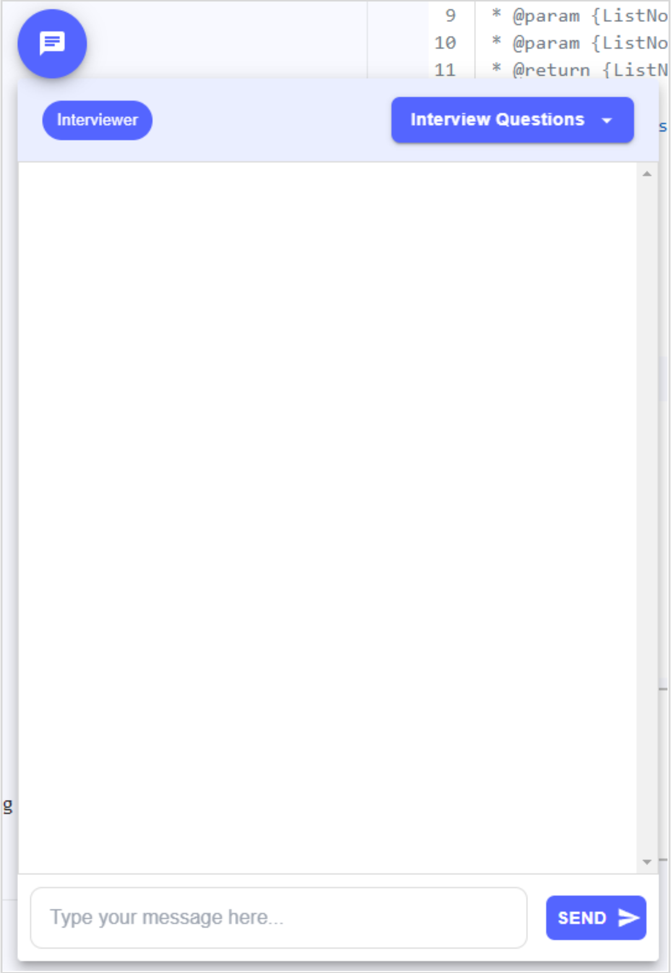


Figure 7: Mock-up of Coding Page Layout After Modification



```
 * @param {ListNo
 * @param {ListNo
 * @return {ListN
```

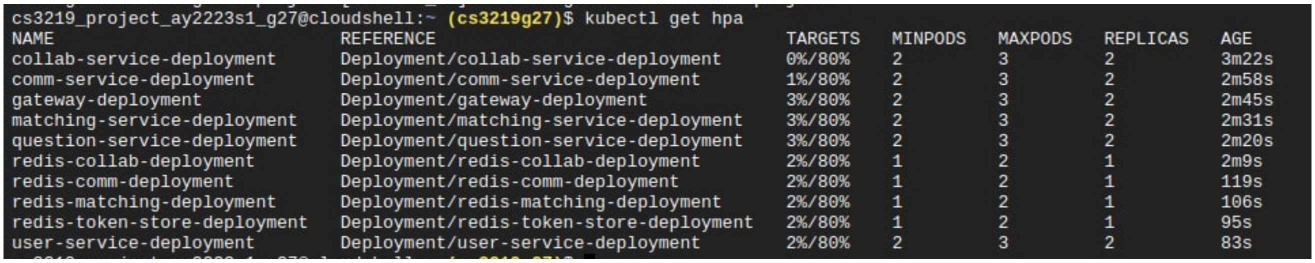
Figure 8: Expanded Chat Window

### 5.3.10 Deployment Details

The deployment for the backend services was done in Google Kubernetes Engine (GKE), while the frontend was deployed to Google Cloud CDN.

We containerized our backend services using docker and pushed them into DockerHub for storage. Dockerizing our services provides a lightweight alternative to using VMs and provides portability, isolation and allows for rapid deployment. We then used Kubernetes for container orchestration, to automate deployment, scaling, and management of our containerized applications. With the use of rolling deployment provided by kubernetes, we are able to provide greater availability with lower downtime to end users.  Kubernetes also provides resource management for pods and containers, allowing us to vertically scale our deployments based on usage patterns.

The Kubernetes horizontal pod autoscaler (HPA) also provides us the means to easily horizontally scale our deployments. We configured the HPA to scale the number of replicas for each deployment when CPU utilization exceeds 80%. For instance when there is a surge in the number of users making requests to the microservices, our gateway deployment will have to utilize more CPU resources. Hence HPA will scale the replica count if each pod is utilizing too much CPU resources. This directly addresses our desired quality attribute of scalability, which links to the availability and usability of our application.



```
cs3219_project_ay2223s1_g27@cloudshell:~ (cs3219g27)$ kubectl get hpa
NAME                          REFERENCE                                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
collab-service-deployment     Deployment/collab-service-deployment     0%/80%    2         3         2          3m22s
comm-service-deployment       Deployment/comm-service-deployment       1%/80%    2         3         2          2m58s
gateway-deployment            Deployment/gateway-deployment            3%/80%    2         3         2          2m45s
matching-service-deployment   Deployment/matching-service-deployment   3%/80%    2         3         2          2m31s
question-service-deployment   Deployment/question-service-deployment   3%/80%    2         3         2          2m20s
redis-collab-deployment       Deployment/redis-collab-deployment       2%/80%    1         2         1          2m9s
redis-comm-deployment         Deployment/redis-comm-deployment         2%/80%    1         2         1          119s
redis-matching-deployment     Deployment/redis-matching-deployment     2%/80%    1         2         1          106s
redis-token-store-deployment  Deployment/redis-token-store-deployment  2%/80%    1         2         1          95s
user-service-deployment       Deployment/user-service-deployment       2%/80%    2         3         2          83s
```

Kubernetes also provides built-in server-side service discovery, allowing our requests to reach a service instance and to be easily load balanced across multiple service instances. Kubernetes thus provides an excellent solution to scalability, and is able to manage many deployments, which complements the microservice architecture which has many components.

We deployed our frontend by saving it in the google bucket via our cd pipeline, and used live reloading capability to automatically update the CDN with the latest version. The use of a CDN brings the data closer to the end user, resulting in lower latency when loading the page and thus better performance.

## 5.3.11 Development Tech Stack

### 5.3.11.1 Programming Language

JavaScript was used as our programming language of choice, as all developers have experience working with JavaScript, thereby increasing development speed. This also allows developers to easily review each other’s code and to understand how each component of the system works. This results in greater collaboration and cross checks, increasing our software quality through code reviews.

### 5.3.11.2 JavaScript Libraries

- JWT
- Bcrypt

- Material UI
- Axios
- Socket IO
- Mongoose
- Dotenv
- React
- Nodejs, Express
- @google-cloud/pubsub
- @uiw/react-codemirror

As there were too many libraries used, the above list is definitely not exhaustive and we will not be discussing every library used above because many of them are influenced by other tech stacks mentioned in the other sections (e.g. database/message queue).

Our discussion would be around the use of JWT and Bcrypt. We used JWT because there are no sessions to be maintained and it is stateless and portable. As long as all services that authenticate requests have the same secret, we can verify the authenticity of the JWT token independently. To store the user's passwords in a salted and hashed form, we used Bcrypt because Bcrypt provides an easy way to salt and hash a password. Password hashing is essential to make sure that we are not storing the user's passwords in plaintext, which would be disastrous if our system is compromised. By salting the password, we make the hash algorithm's output unpredictable, making it difficult for attackers to reverse-engineer passwords, thereby increasing the security of our software.

### 5.3.11.3 Local Development

Postman was frequently used during development, to manually test and verify that our endpoints are working as intended.

### 5.3.11.4 Datastore

In matching service, Redis was used as the in memory data store, to facilitate match finding. Such a use case focuses more on latency performance over persistence. After all, if the matching service fails, all the pending matches will not find any match anyways, hence it is ok if our data store is volatile in this case. As Redis has low latency and is performant in memory data store, it is perfect for such a use case.

### 5.3.11.5 Persistent Storage

- MongoDB

- MongoDB Atlas
- MongoDB Compass

Since most of our services are stateless and do not require persistent storage, our discussion below only refers to user service, where we chose MongoDB as our database.

Often, the consideration for the choice of database begins with SQL versus NoSQL. Since our application does not have many relations, and does not require transactional database operations, we do have the option to go with a NoSQL database. We chose a NoSQL database because it allows for easier scalability and reliability through sharding and replication. Although we did not have to employ that, we do have that option to scale if our application requires them in the future. While it is definitely possible to do sharding and replication on a SQL database, it is often much more complex as a SQL database has to maintain the ACID guarantees and also costly if a distributed join is used. A NoSQL database also does not require the same rigid schema used by SQL databases, and thus gives us the flexibility to modify our data models, making our development more adaptable to changes.

We chose to use MongoDB as our NoSQL database because it stores data records as BSON documents which are binary representations of JSON. Since we are working with JavaScript on both the frontend and backend, this format is familiar and intuitive with us, and mongoose provides an intuitive wrapper to query the database.

### 5.3.11.6 Message Queues (Pub/Sub)

- Redis
- Google Pub/Sub

For the message queues, we used Redis for communication within the same service and Google Pub/Sub for communication across services. Redis was used in the Matching service, Communication service and Collaboration service to provide communication within each of these services, so that we can have multiple instances of each service, to provide scalability and reliability[8]. We did not mention the use of redis as pub sub in our architectural design discussion, because redis is used here to facilitate scalability, and that this level of detail should be omitted from a high level architecture diagram which should convey the high level components and interactions between them.

[8] https://socket.io/docs/v4/redis-adapter/

The benefits of using asynchronous message passing communication is already described in the architectural design section. We specifically chose Google Pub/Sub, because it supports most of the popular programming languages (C++, C#, Go, Java, Node, PHP, Python, Ruby), which is important if we want to create a different microservice that uses the message queue using another programming language in the future. Google Pub/Sub is also suited here since we already have our services deployed in the GCP environment, and we are able to do monitoring on the message queues as well as other monitoring such as deployment monitoring all in the same platform provided by GCP.

### 5.3.11.7 Deployment Tech Stack

- Google Cloud Platform (GCP)
- Google Kubernetes Engineer (GKE)
- Google CDN
- Docker
- Kubernetes

We wanted our deployment to happen all within the same place, and hence we decided to use the services provided by GCP. For a deeper discussion on the deployment tech stack and details, please refer to the section titled deployment details.

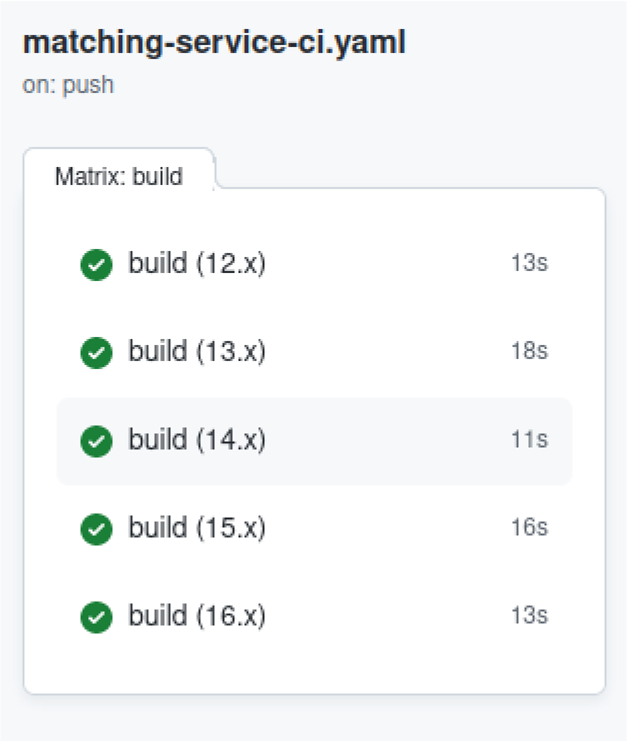
### 5.3.11.8 Other External API

- Rapid API
- SendGrid API
- Leetcode API

To compile and run the code in the editor, we used Rapid API. It offers 50 requests per day within the free tier. We used SendGrid API to send reset password emails to users, and Leetcode API to fetch the programming questions to be used in the session.

## 5.3.12 Development Process

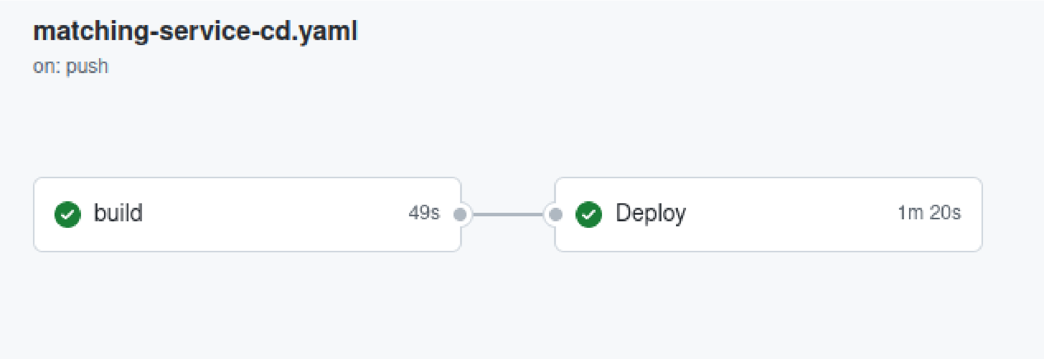
### 5.3.12.1 CI/CD Details



matching-service-ci.yaml
on: push

Matrix: build

| | |
|---|---|
| ✓ build (12.x) | 13s |
| ✓ build (13.x) | 18s |
| ✓ build (14.x) | 11s |
| ✓ build (15.x) | 16s |
| ✓ build (16.x) | 13s |



matching-service-cd.yaml
on: push

✓ build 49s → ✓ Deploy 1m 20s

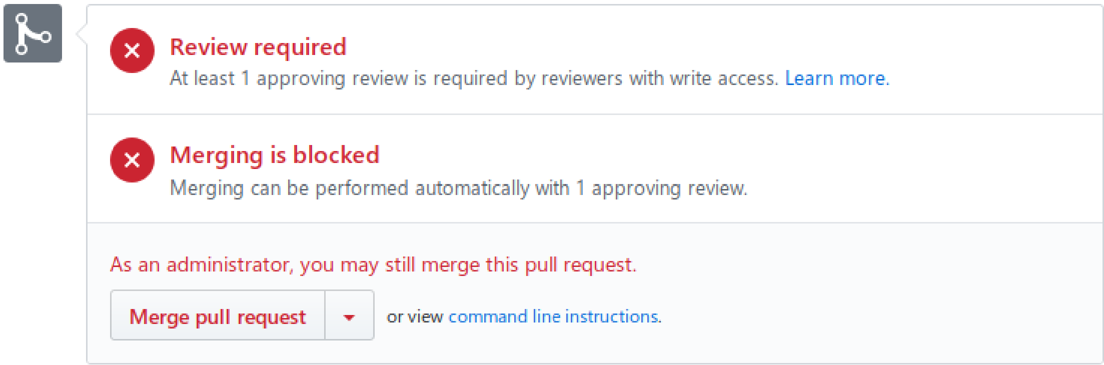
In the CI pipeline, all microservices are built and tested. Each pipeline runs on a different node version, ensuring that our microservices can be built and pass test cases before merging.

We adopted the practice of Continuous Deployment as the deployment to the production environment is automatic. In the CD pipeline, all microservices have their docker images built and pushed to dockerhub. We then deploy the microservices on GKE with the relevant yaml files found in the gke folder. For each of the microservices, we write a service object in

Kubernetes which provides server side service discovery. For the frontend, we upload to google bucket and enable live reloading capability to automatically update the CDN with the latest version. As such, the latest updates are always available on the production url.

We used GitHub actions to run our pipelines.

### 5.3.12.2 Quality Assurance



Review required

At least 1 approving review is required by reviewers with write access. Learn more.

Merging is blocked

Merging can be performed automatically with 1 approving review.

As an administrator, you may still merge this pull request.

To provide a minimal level of quality assurance, we protected our main branch and prevented direct push to the main branch. We then developed using feature branching, merging into the main branch frequently, so developers can work in parallel without making breaking changes. We also ensured that pull requests cannot be merged into the main branch unless approved by a reviewer.

### 5.3.12.3 Development Process

We have weekly stand ups where we discuss the work we have done for the week, as well as any blockers. We discuss if there are any changes required to the requirements as well, updating our product backlog if required. We then do sprint planning, and assign tasks to each member. By dividing the work into manageable tasks, we are able to make small incremental changes, which are easier to review, and provide us with a potentially shippable product increment at the end of every sprint. Our code review process happens at any time of the week and is ideally done by someone who is familiar with the microservices involved.

### 5.3.12.4 Development Plan

The table below shows the development plan that the team has come up with for the second half of the project. It consists of the functional and non-functional requirements for each specific component/service, to be completed by the particular week highlighted in **light gray**. The boxes highlighted in **dark gray** refer to requirements that are not met, which will be briefly explained below.

| Development Planning | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | Recess | W7 | W8 | W9 | W10 | W11 | W12 |
| | **F1. User Service** | | | | | | | |
| F1.1 | The system should allow users to reset their password using their email address. | ▨ | | | | | | |
| F1.2 | The system should maintain a record of the user’s past attempted questions, including a record of the text editor contents and chat history. | | | | | ▨ | | |
| NF1.1 | The system should be able to handle 100 concurrent requests of the same type. | | | | ■ | | | |
| | **F2. Matching Service** | | | | | | | |
| NF2.1 | The system should support concurrent matchmaking without any error. | | | | | ■ | | |
| NF2.2 | The system should support up to 50 concurrent matchmaking requests. | | | | | ■ | | |
| | **F3. Real-time Collaboration Service** | | | | | | | |
| F3.1 | The system should allow the Interviewer to send qualitative questions. | | ▨ | | | | | |
| F3.2 | The system should provide questions of the right difficulty from a database of relevant coding questions. | | ▨ | | | | | |
| F3.2 | Questions should include an examples section to help students better understand questions. | | ▨ | | | | | |
| F3.3 | The system should allow users to see other users’ edits in real-time. | | | ▨ | | | | |
| F3.4 | The system should notify either user if the other collaborator in the session has disconnected and provide a link for the remaining user to continue on an external coding practice website. | | | ▨ | | | | |
| F3.5 | The system should allow users to assume the roles of Interviewer and Interviewee and swap roles. | | | ▨ | | | | |
| F3.6 | The system should provide a link for users to attempt questions from past attempts on an external coding | | | ▨ | | | | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | practice website. | | | ■ | | | | |
| NF3.1 | A session should be set up within 7 seconds | | | ■ | | | | |
| F3.7 | The system should assign Interviewer/Interviewee roles to users. | | | ■ | | | | |
| | **F4. Communication Service** | | | | | | | |
| F4.1 | The system should allow users to communicate via text and voice call or video call to train their communication skills. | | | ■ | | | | |
| F4.2 | The system should provide a list of interviewer questions for the Interviewer. | | | | ■ | | | |
| | **F5. Basic UI for User Interaction** | | | | | | | |
| F5.1 | The system should have a button for the users to leave the matching queue before the timer ends. | | ■ | | | | | |
| F5.2 | The system should provide a reset password page for users who have forgotten their passwords and wish to retrieve their accounts. | | ■ | | | | | |
| F5.3 | The system should have a pop-up to notify the user if the other collaborator has disconnected from the session. | | ■ | | | | | |
| F5.4 | The system should provide a collaborative text editor for users to edit their code. | | | ■ | | | | |
| F5.5 | The system should display the programming problem to the user. | | | ■ | | | | |
| F5.6 | The system should provide a real-time chat UI for two collaborators to communicate during the same session. | | | | ■ | | | |
| F5.7 | The system should add an “Interviewer/Interviewee” role tag in the chatbot corresponding to the user’s role to notify them about their role. | | | | ■ | | | |
| F5.8 | The system should allow the two collaborators to swap roles upon clicking a button. | | | | | ■ | | |
| F5.9 | The system should allow users to access the history of previously attempted questions at the homepage. | | | | | ■ | | |
| F5.10 | The system should provide a dropdown of interview | | | | | | ■ | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | questions for the user with the "Interviewer" role. | | | | | | ▨ | |
| F5.11 | The system should position the code editor and question display side-by-side for the user to refer to the question more conveniently while coding. | | | | | | ▨ | |
| | **F6. Deployment** | | | | | | | |
| NF6.1 | The application should be accessible via a public URL | | | ▨ | | | | |
| NF6.2 | All requests to the application should be facilitated by an API Gateway. | | | ▨ | | | | |
| NF6.3 | The application should be available, scalable and performant. | | | | | | ▨ | |

NF1.1, NF2.1 and NF2.2 were not met because it is hard to test if the requirements are met. That is not to say that the metrics to meet the requirements are impossible to measure and test. However, given current circumstances, unless we perform user testing, we are unsure how we could go about testing that the system is capable of handling 50 concurrent matchmaking requests.

### 5.3.12.5 Project Management

We followed an Agile way of development, where development happens in weekly sprints that start and end on our weekly meetings on Thursdays.
Each meeting lasts for 30 minutes, where we discuss the following details:

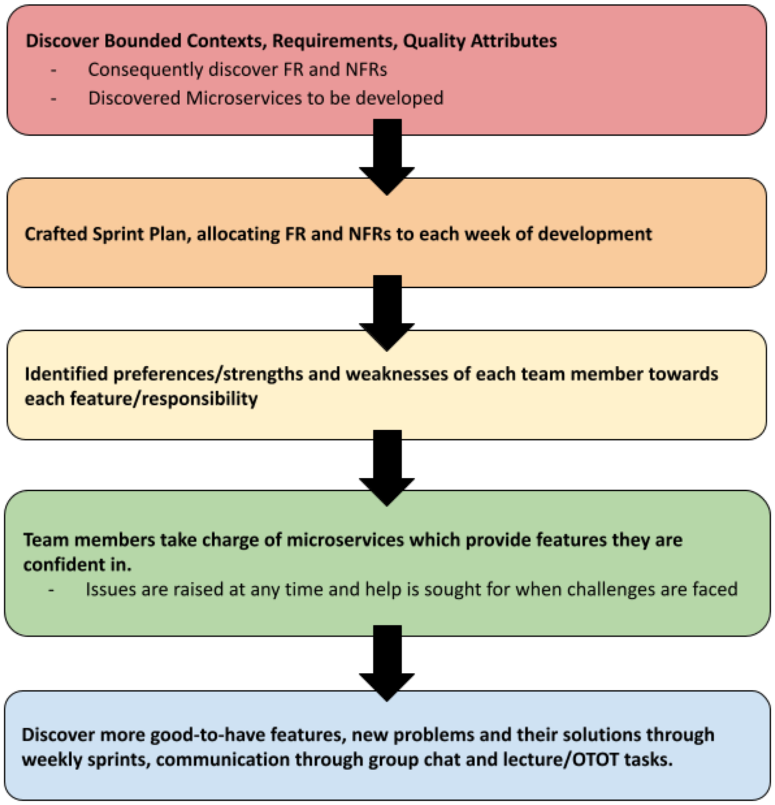
- What I have done in the past week
- What I plan to do in the coming week
- Raise issues/concerns with my own/ teammates features
- Share interesting and useful online resources that pertain to each other’s responsibilities.

The approach to development within our team was independent but supportive development.

By adopting a microservice architecture, development of each service could happen in parallel and team members could work independently.

However, the tools and solutions needed to build this product were often new domains for us, therefore it was crucial for teammates to support each other in knowledge sharing through sharing of relevant tutorials and online resources.

As such, having weekly sprints proved to be really helpful in balancing between independent and supportive development among team members.



**Discover Bounded Contexts, Requirements, Quality Attributes**

- Consequently discover FR and NFRs
- Discovered Microservices to be developed

**Crafted Sprint Plan, allocating FR and NFRs to each week of development**

**Identified preferences/strengths and weaknesses of each team member towards each feature/responsibility**

**Team members take charge of microservices which provide features they are confident in.**

- Issues are raised at any time and help is sought for when challenges are faced

**Discover more good-to-have features, new problems and their solutions through weekly sprints, communication through group chat and lecture/OTOT tasks.**

*Development Approach Across The Weeks*

We used GitHub issues to track and assign tasks. Since we are using GitHub as our repository, GitHub issues allowed us to trace code changes back to issues created.

Our team has used Continuous Deployment and Devops Tools such as Kubernetes and GKE since the beginning of development. This allowed us to incorporate a DevOps practice where a team member develops their own microservices and debug relevant deployment problems quickly. This practice allowed every member in the team to troubleshoot issues for one another, and identify potential bugs when Reviewing each other’s Pull Requests.

# 6. Improvements & Enhancements

## 6.1 Video Chat Functionality

A possible enhancement to our application is the addition of a video chat functionality on top of the chat functionality that the application currently already provides. According to a recent Indeed survey, 82% of employers surveyed use virtual interviews, and 93% of employers plan to continue using them[9]. This highlights the fact that virtual interviews are getting more common, and possibly even more common than offline interviews. Thus, having only a chat functionality may not be sufficient in providing the full experience of a virtual interview. An added video chat functionality will more accurately mirror an actual virtual interview, hence enhancing the efficiency of the coding session as they will better prepare themselves for such interviews.

## 6.2 Forum Page

Another possible enhancement to the application is to integrate a forum page, where users can discuss the possible solutions to the questions that they have attempted. Providing a discussion platform saves users the time to navigate to a different search engine to search up the answers to these coding questions, which may not be readily available. Furthermore, having a forum can help to foster a sense of community and encourage peer-to-peer interaction[10], which is the main objective of this application. This can help improve learner engagement, enhancing the users' experience all together.

## 6.3 Rating

The last enhancement to the application is to introduce a rating system, and attempt to pair users of similar ratings together. Perhaps, this rating could be influenced by the number of matches completed, the solve rates or even feedback from the other user after every match. This can improve user experience, and allow users who are good to be able to match with good users so that they can learn more from each other.

[9] https://www.forbes.com/sites/forbesbusinesscouncil/2022/05/05/the-virtual-interview-is-the-new-resume-what-you-need-to-know

[10] https://blogs.ed.ac.uk/ede/2021/03/03/the-importance-of-discussion-forums-for-online-learning

# 7. Reflections

## 7.1 Problems Faced

We could have done better planning for the component design, and decide on a design for the component before coding. This would reduce the model-code gap and allow us to apply the patterns which comply with good principles.

We should have also attempted to use more Gangs of Four (GoF) design patterns in our code. However, this topic was introduced towards the end of the module, so we were unable to apply much of what we have learnt into practice without making big changes and refactoring.

## 7.2 Knowledge Gained

Throughout the entire project, we definitely learned to work better in a team of software engineers, and picked up both technical and non-technical skills along the way.

We also managed to discover requirements, record them, plan them and eventually trace back to them. We also identified subdomains from the problem space, and mapped them to the respective bounded contexts.

As we learn the theory and importance of design patterns and design principles in lectures, we were able to put these knowledge into practice by applying them to the project directly while working on it. We learned to better structure our architecture, and adopted the microservice architecture to integrate the different services. This structure makes it easy for us to introduce new services to the application in the future, making this application extensible and highly cohesive, while increasing the maintainability.

Besides the technical knowledge gained, the team definitely picked up some valuable non-technical skills which can certainly be applied to similar team settings in the future. Some skills include communication skills within the team members, personal management and time management. We realized the importance of effective communication within the team, especially during our weekly stand-up meetings, to ensure that everyone is on the same page. Clear communication within the team also helps us with clearing any misunderstandings with regards to the functionalities and scope of the application, especially for nice-to-haves add-ons that were not specified in the development guide. Personal management and time management are also a key in helping us complete our assigned tasks on time, following the development plan that the team has come up with. While refining these skills, we were able to voice out any concerns regarding internal deadlines set for certain features, and follow the development plan closely to ensure that nobody is lacking behind.