



---

School of Computing

# CS3219 Software Engineering Principles and Patterns

AY22/23 Semester 1

Team 2

Wong Hon Wee, Howard A0220867B

Bryan Lim Yan Jie A0183054J

Yee Zong Shen Johnson A0211551W

<b>1.Introduction</b>	<b>4</b>
1.1 Background of Peerprep	4
1.2 Purpose of Peerprep	4
<b>2. Project Management</b>	<b>5</b>
2.1 Software Development Process	5
2.2 Project Scope	6
<b>Stretch Goal Features:</b>	<b>7</b>
2.3 Sprint Schedule	7
<b>3. Functional Requirements:</b>	<b>9</b>
3.1 User Service	9
3.2 Matching Service	9
3.3 Question Service	10
3.4 Collaboration Service	10
3.5 Chat Service	10
<b>4. Non-Functional Requirements</b>	<b>11</b>
4.1 Performance	11
Performance Requirements	11
4.2 Usability	12
Usability Requirements	13
4.3 Accessibility	14
Accessibility Requirements	14
4.4 Security	15
Security Requirements	15
4.5 Scalability	16
Scalability Requirements	16
<b>5. Architecture Design</b>	<b>17</b>
5.1 Architecture	17
5.2 Deployment	18
5.3 CI/CD	19
5.4 Architecture Pattern	20
Database per service Pattern	20
MVC	21
Pub-Sub Pattern	21
<b>6. Services</b>	<b>22</b>
6.1 Frontend	22
User Authentication	22
Difficulty Page	24

FindingMatch Page	25
Room Page	26
6.2 Matching Service	27
6.3 User Service	28
6.4 Collaboration Service	33
6.5 Question Service	33
<b>7. Testing</b>	<b>37</b>
Unit Testing	37
Load Testing	39
<b>8. Remarks</b>	<b>40</b>
8.1 Potential Improvements / Enhancements	40
Interviewer / Interviewee / Collaborator Roles	40
Voice Chat	40
8.2 Reflections, Challenges & Learning Points	40
Manpower	40
Time management	40

# 1.Introduction

## 1.1 Background of Peerprep

Technical interviews are now integral to the hiring process of many software jobs. However, many students are facing difficulties such as inability to communicate their ideas or inability to solve the given problem. Grinding out questions alone can also be tedious and monotonous.

Mock interviews are a solution to this but mock interview services by professionals online are often expensive to broke university students. Thus, the idea for an interview preparation platform with peer matching was born.

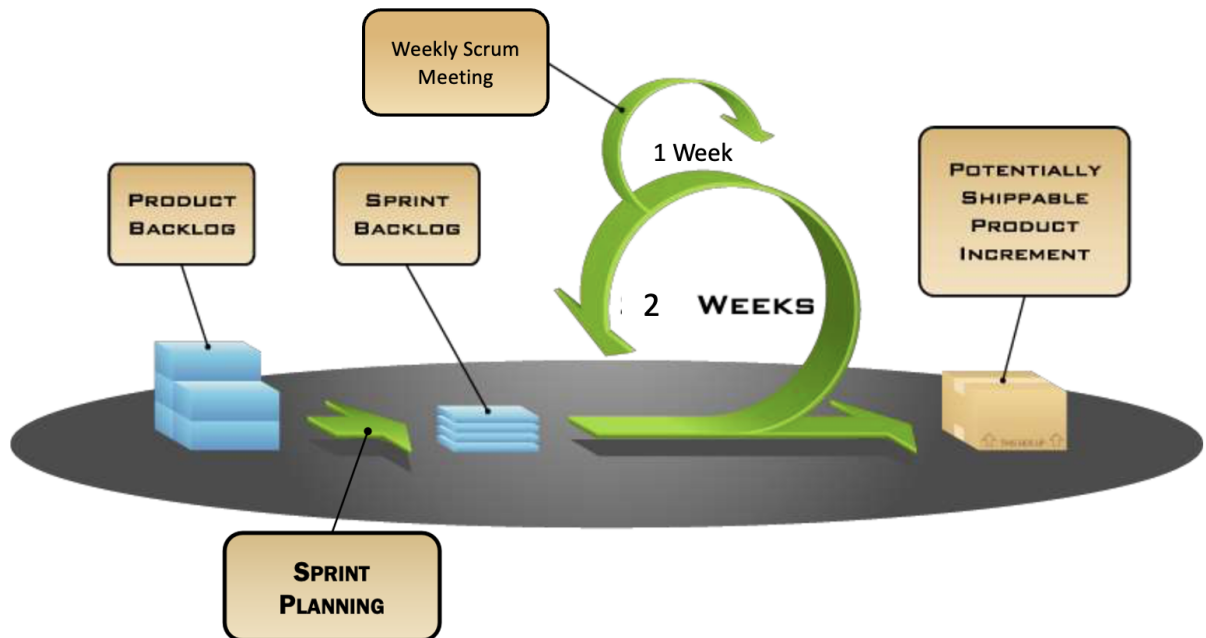
## 1.2 Purpose of Peerprep

Having noticed the difficulty of technical interviews in hiring processes, there is a need for collaborative technologies that allow users to code together, exchange ideas and learn from each other so that users have greater confidence when approaching such interviews.

The product Peerprep is a web application that aims to help users to work together in a collaborative environment to prepare for practice for technical interviews. This is achieved by programming questions of varying difficulties for users to solve as well as a real-time text editor and in-app chat system for collaborative purposes.

## 2. Project Management

### 2.1 Software Development Process



For our software development, we decided to use an agile process as we were all not sure which of the nice-to-have features we would pursue for the project nor the tech stack we would use.

Thus our preferences for the software and requirements is prone to change over the course of the project. Since responding to change is part of the Agile Manifesto, Agile processes would be a good fit for our project.

The iterative nature of an agile process also allowed us to continue to implement some of our features incrementally even when not all of them were fully fleshed out yet.

We used a modified version of Scrum where a sprint cycle would be 2 weeks but we would have a Scrum meeting every week on Tuesday in order to match everyone's schedules.

During each meeting, we would:

- Inform each other about what we had done
- Voice out which tasks were being blocked by others (if there were any)

At the end of each sprint, our meeting would also include a review session of the past sprint and planning session for the next sprint.:

- We would check what we were able to complete
- See if there was anything we could improve on
- Re-prioritise the tasks in our sprint backlog
- Assign tasks to everyone

We used Google Docs to keep track of our product backlog and Github Issues to track bugs.

## 2.2 Project Scope

This maps to the features stated in the project description, further details can be found in their respective sections (linked for convenience).

Core Features:

- [User Service](#)
  - To maintain user information
- [Matching Service](#)
  - To match users based on difficulty level user specifies
- [Question Service](#)
  - To store questions and provide a random question based on specified difficulty level
- [Collaboration Service](#)
  - To allow matched users to collaborate real-time on code

Additional Features:

- [Good User Interface](#)
  - For users to utilize the application effectively, efficiently and without being hampered by the UI.
- [Chat Service](#)
  - To allow users to communicate
  - To allow users to clearly distinguish between using the collaboration service for creating an answer and the chat service to communicate
- [Local deployment with containerization \(Docker\) & orchestration \(Kubernetes\)](#)
- Implement CI/CD on the project

Stretch Goal Features:

- Deployment of application on a production system

## 2.3 Sprint Schedule

Week	Howard	Bryan	Johnson	Alvin
4	<ul style="list-style-type: none"> <li>- Elicitation of FRs for Matching Service</li> <li>- Design Matching Service</li> </ul>	<ul style="list-style-type: none"> <li>- Elicitation of FRs for Matching Service</li> <li>- Design Matching Service</li> </ul>	<ul style="list-style-type: none"> <li>- Elicitation of FRs for User Service</li> <li>- Design User Service</li> </ul>	<ul style="list-style-type: none"> <li>- Set up Team Repo</li> <li><del>- Elicitation of FRs for User Service</del></li> </ul>
5	<ul style="list-style-type: none"> <li>- Implement FR 2.2, 2.3</li> </ul>	<ul style="list-style-type: none"> <li>- Implement FR 2.1, 2.4-2.6</li> </ul>	<ul style="list-style-type: none"> <li>- Implement FRs 1.1-1.6</li> <li>- Implement NFR5.1</li> </ul>	<ul style="list-style-type: none"> <li><del>- Implement FRs 1.1-1.4, 1.8-1.10</del></li> </ul>
6	<ul style="list-style-type: none"> <li>- Elicitation of FRs for Chat Service</li> </ul>	<ul style="list-style-type: none"> <li>- Elicitation of FRs for Question Service</li> <li>- Elicitation of NFRs for System</li> </ul>	<ul style="list-style-type: none"> <li>- Elicitation of FRs for Collaboration Service</li> </ul>	<ul style="list-style-type: none"> <li><del>- Elicitation of FRs for Collaboration Service</del></li> </ul>
Recess	<ul style="list-style-type: none"> <li>- Consider Architecture Patterns &amp; Designs for application</li> <li>- Design Chat Service</li> </ul>	<ul style="list-style-type: none"> <li>- Partial Implementation of FR3.1</li> <li>- Consider Architecture Patterns &amp; Designs for application</li> <li>- Design Question Service</li> </ul>	<ul style="list-style-type: none"> <li>- Consider Architecture Patterns &amp; Designs for application</li> <li>- Design Collaboration Service</li> </ul>	
7				
8	<ul style="list-style-type: none"> <li>- Implement FR5.1</li> </ul>	<ul style="list-style-type: none"> <li>- Implement FR 3.2, 3.3</li> </ul>	<ul style="list-style-type: none"> <li>- Implement FR 4.1</li> </ul>	

9	- Implement NFR1.4, 2.6		- Implement NFR 2.4	
10 (Post PPT)	- Set up deployment environment	- Implement FR 3.1	- Implement FR 4.2	
11	- Implement CI/CD	- Implement CI/CD - Implement NFR, 4.3	- Implement CI/CD - Implement NFR 1.2, 2.2	
12	- Setup deployment environment	- Implement NFR 1.3, 3.1, 3.2 - Improve FE styling	- Implement NFR 4.1, 4.2 - Beautify frontend	
13	Presentation			



## 3. Functional Requirements:

### 3.1 User Service

S/N	Functional Requirement	Priority	Done
FR1.1	The user service module should allow users to register a new account using name/email address and the desired password.	High	Y
FR1.2	The user service module should allow users to log into their own accounts with the registered name/email address and password.	High	Y
FR1.3	The user service module should allow users to log out when needed.	High	Y
FR1.4	The system should ensure that every account created has a unique username.	High	Y
FR1.5	The system should allow users to log out of their account.	High	Y
FR1.6	The system should allow users to change their password	Medium	Y

### 3.2 Matching Service

S/N	Functional Requirement	Priority	Done
FR2.1	The system should allow users to select the difficulty level of the questions they wish to attempt.	High	Y
FR2.2	The system should be able to match two waiting users with similar difficulty levels and put them in the same room.	High	Y
FR2.3	If there is a valid match, the system should match the users within 30s.	High	Y
FR2.4	The system should inform the users that no match is available if a match cannot be found within 30 seconds.	High	Y
FR2.5	The system should provide a means for the user to leave	Medium	Y

	a room once matched.		
FR2.6	User should be able to tell that the matching process is in progress	Low	Y

### 3.3 Question Service

S/N	Functional Requirement	Priority	Done
FR3.1	The system should have a pool of at least 100 different questions for each difficulty level	High	Y
FR3.2	The system should have different question pools for different difficulties	High	Y
FR3.3	The system should provide a way for the matching service to retrieve a random question by difficulty.	High	Y

### 3.4 Collaboration Service

S/N	Functional Requirement	Priority	Done
FR4.1	The system should allow both participants should be able to see each other's input in near real-time response.	High	Y

### 3.5 Chat Service

S/N	Functional Requirement	Priority	Done
FR5.1	The system should allow both users to send messages to one another	High	Y

## 4. Non-Functional Requirements

In descending order of priority, they are:

- 1) Performance
- 2) Usability
- 3) Accessibility
- 4) Scalability

### 4.1 Performance

We set performance as our highest priority as the majority of the user experience is centered around collaborative coding.

In order to benefit from collaboration, near real-time interaction is essential so that the flow of communication is uninterrupted. We want communication between our users to be smooth and efficient.

Thus, our top priority is to ensure our application is responsive when they are collaborating, otherwise this would jeopardize the core experience of our application.

Performance of our system leading up to the collaboration is also a high priority as we want to route users to our core features as quickly as possible.

#### Performance Requirements

S/N	Non-Functional Requirement	Priority	Done
NFR1.1	Room page should load within 5 seconds after a match is found.	Medium	Y
NFR1.2	At least 98% of the time, edits by one person on the collaborative editor should appear on the other persons editor within 1 second of occurring.	High	Y
NFR1.3	Question Service should respond to a request for a random question within 2 seconds.	Medium	Y
NFR1.4	At least 98% of the time, Chat Service should have lag of less than 2 seconds when sending messages between users	Medium	Y

To see how we achieved NFR 1.3, refer to [this](#).

## 4.2 Usability













We set usability as our second highest priority as we want to get as many users as we can for our application (this would also improve user wait times for difficulty matching).

Thus, users should feel comfortable using our application so they are more likely to stick with our software.

We target this in terms of ease of use (UI & syntax highlighting) and familiarity (syntax highlighting).

Within each browser session, your last-used language mode is stored for users' convenience.

For syntax highlighting features, we plan to support the languages below and possibly expand later on. These languages were chosen by cross-referencing the most popular languages based on the TIOBE index<sup>1</sup> with the list of supported languages from another popular interview prep website Leetcode<sup>2</sup>.

Nov 2022	Nov 2021	Change	Programming Language	Ratings	Change
1	1		 Python	17.18%	+5.41%
2	2		 C	15.08%	+4.35%
3	3		 Java	11.98%	+1.26%
4	4		 C++	10.75%	+2.46%
5	5		 C#	4.25%	-1.81%
6	6		 Visual Basic	4.11%	-1.61%
7	7		 JavaScript	2.74%	+0.08%
8	8		 Assembly language	2.18%	-0.34%
9	9		 SQL	1.82%	-0.30%
10	10		 PHP	1.69%	-0.12%
11	18		 Go	1.15%	-0.06%

TIOBE Index for Most Popular Programming Languages

### PeerPrep Supported Languages

- Python
- Go
- Javascript

---

<sup>1</sup> The TIOBE index is a popularity index for programming languages that is updated monthly.

<sup>2</sup>

<https://support.leetcode.com/hc/en-us/articles/360011833974-What-are-the-environments-for-the-programming-languages>

## Usability Requirements

S/N	Non-Functional Requirement	Priority	Done
NFR2.1	Intuitive and unobstructive UI for users to go from landing page to core experience of collaborative coding quickly and painlessly.	High	Y
NFR2.2	Support syntax highlighting for <a href="#">popular languages</a> within the code editor similar to IDEs or code editors that most coders are familiar with.	Medium	Y
NFR2.3	It should be easy for users to indicate which part of code they are referring to when they are discussing. (Line number)	High	Y
NFR2.4	Users should be able to identify which messages were sent by themselves easily when chatting with other users	High	Y

Mode

Python ▼

```
1 num = 29
2 flag = False
3 # prime numbers are greater than 1
4 if num > 1:
5     # check for factors
6     for i in range(2, num):
7         if (num % i) == 0:
8             # if factor is found, set flag to True
9             flag = True
10            # break out of loop
11            break
```

Mode

JavaScript ▼

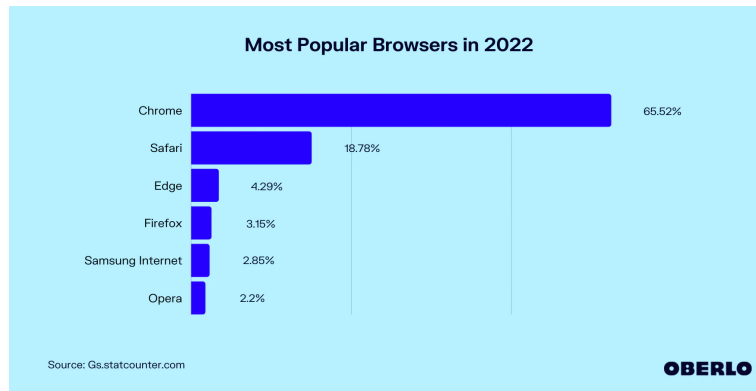
```
1 const isPrime = num => {
2     for(let i = 2, s = Math.sqrt(num); i <= s; i++)
3         if(num % i === 0) return false;
4     return num > 1;
5 }
```

As shown in the images above, syntax highlighting (NFR2.2) is supported by our code editor for users to have a more familiar experience.

## 4.3 Accessibility

In line with the idea of appealing to as many users as possible, accessibility is another high priority for us.

As preparing for coding interviews is a practice shared by coders of all backgrounds, each with their own preferences for their applications, it was a priority for us to ensure our application works on most of the browsers that are commonly used today.



Browser Market Share from [Oberlo.com](https://oberlo.com)

In our case, we targeted Chrome, Firefox, Safari and Edge.

Opera, Samsung Internet & many other browsers are based on the Chromium browser which is very similar to Chrome.

Chromium differs from Chrome in aspects such as: Media Codec support, PDF Viewer, Adobe Flash Plugin, Flash Player Support and several others<sup>3</sup>. Thus, we take additional care in not designing our apps around these features to prevent incompatibility issues.

Though we understand that the best action to take would be to test our application on each of those browsers, given our limited manpower and time, we hope to indirectly target Chromium-based browsers by ensuring our application works on Chrome.

However, this quality attribute is ranked lower than usability as coders that use unsupported browsers can always download another browser if they really want to use our application.

---

<sup>3</sup> <https://www.lambdatest.com/blog/difference-between-chrome-and-chromium/>

## Accessibility Requirements

S/N	Non-Functional Requirement	Priority	Done
NFR3.1	The app should be usable across several popular browsers Chrome, Firefox, Safari and Edge	High	Y
NFR3.2	Page should display properly for different screen sizes on PC. (16:9, 21:9, 4:3)	Low	Y

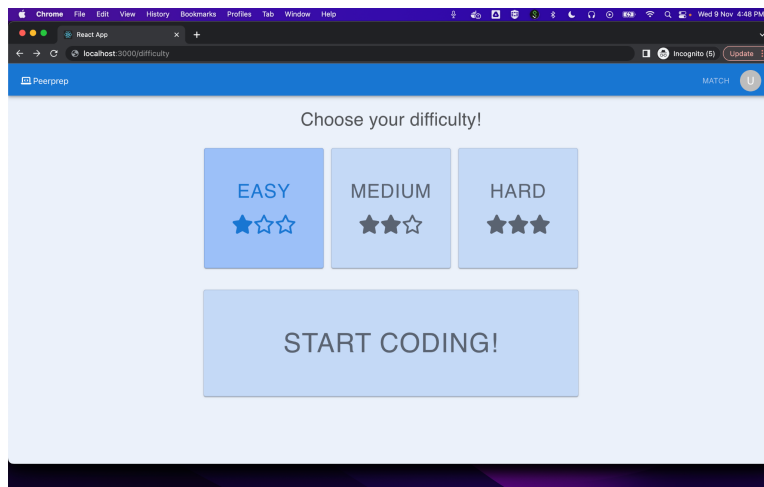


Image of Application in Chrome

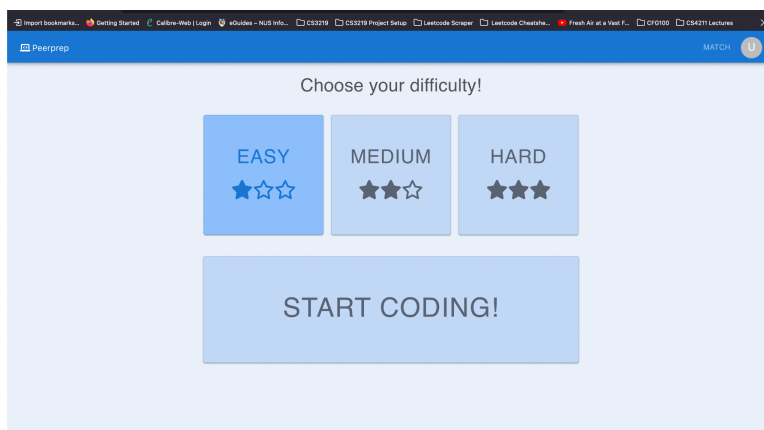


Image of Application in Firefox

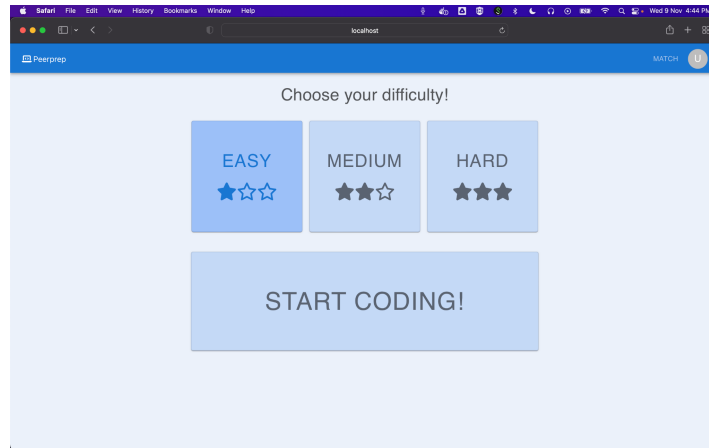


Image of Application in Safari

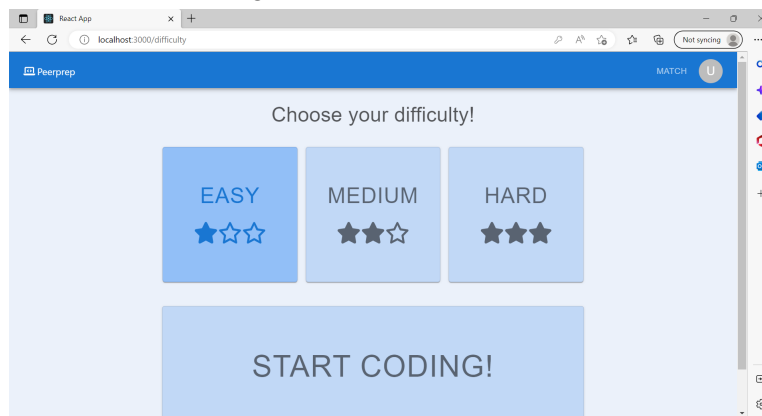


Image of Application in Edge

As seen in the images above, the application is compatible with Chrome, Firefox, Edge and Safari.

## 4.4 Security

In order to ensure that the users are comfortable using our application, security is required so that they will have a peace of mind when using our application. Thus, we ensure that the credentials of users are encrypted and kept securely so that it cannot be misused. An example of the encrypted password is shown below.



## Security Requirements

S/N	Non-Functional Requirement	Priority	Done
NFR4.1	Users' passwords should be hashed and salted before storing in the DB.	Medium	Y

```
const encryptedPassword = await bcrypt.hash(password, BCRYPT_COST);  
const resp = await _createUser(username, encryptedPassword);
```

```
__id: ObjectId('636b5c136459b929d416f7a4')  
username: "testUser123"  
password: "$2b$06$5tX6E99kt4ehW9iXvNjSCOMCQWI1RvrYglFykY7AIOxkAhKqMHptu"  
__v: 0
```

As seen from the images above, the password is salted and hashed before we store it in our database.

## 4.5 Scalability

As our user base grows, our application must be able to grow in order to support their usage and maintain our latency requirements.

The way we build and deploy our application will have ramifications on how easily we can scale in the future. Thus, scalability is important for us to consider and prioritize early on.

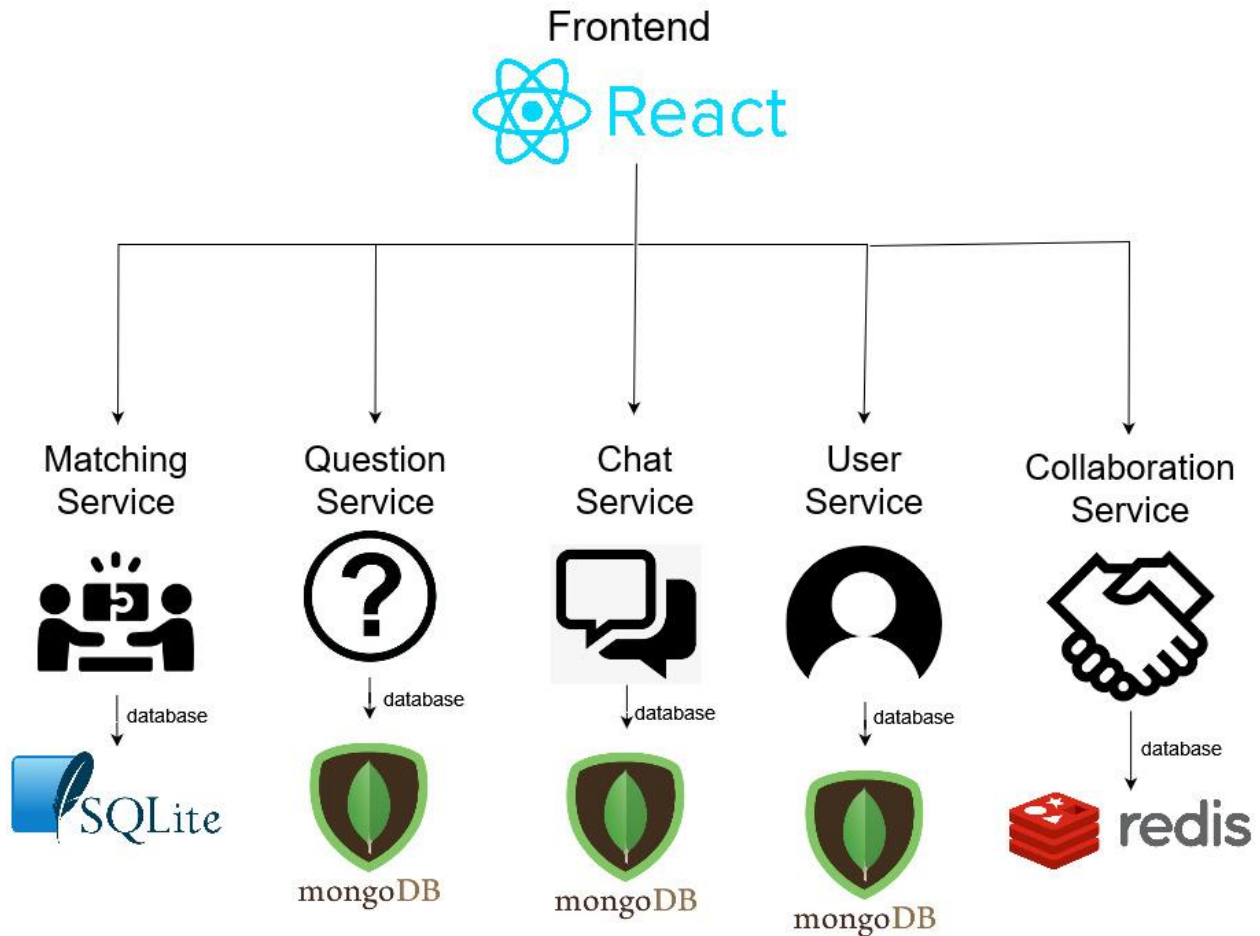
Scalability ranked the lowest in priority because our users must stick before the numbers can grow and scalability becomes an issue when our user numbers are large. Thus performance, usability and accessibility take precedence.

## Scalability Requirements

S/N	Non-Functional Requirement	Priority	Done
NFR5.1	The system should be able to support at least 5 concurrent collaboration rooms (with associated services) at any point in time.	High	Y
NFR5.2	The system should be able to support at least 1000 registered users.	High	Y
NFR5.3	The system should be able to support at least 300 questions in its database.	High	Y

## 5. Architecture Design

The application is built using a microservices architecture and deployed using kubernetes as seen in the figure below.



The application consists of 5 main services which are the following

1. **User Microservice:** Responsible for login, registration and authentication of user
2. **Matching Microservice:** Responsible for matching users who are looking for the same difficulty
3. **Question Microservice:** Responsible for providing questions to the user
4. **Collaboration Microservice:** Responsible for facilitating the collaboration between users by providing a shared text editor between users
5. **Chat Microservice:** Allows users to send messages to each other in real time

## 5.1 Architecture

During the design phase of the project, we considered two types of architecture to use for our project which are namely the Microservice and Monolith architectures and compared their benefits which can be found below in order to decide which of them we should use for our project.

	Monolith Architecture	Microservice Architecture
Benefits	<ul style="list-style-type: none"><li>• Simple to develop, scale and deploy for less complex applications</li></ul>	<ul style="list-style-type: none"><li>• Lesser coupling between logical components</li><li>• Allows more flexibility by scaling only services that consume more resources</li></ul>

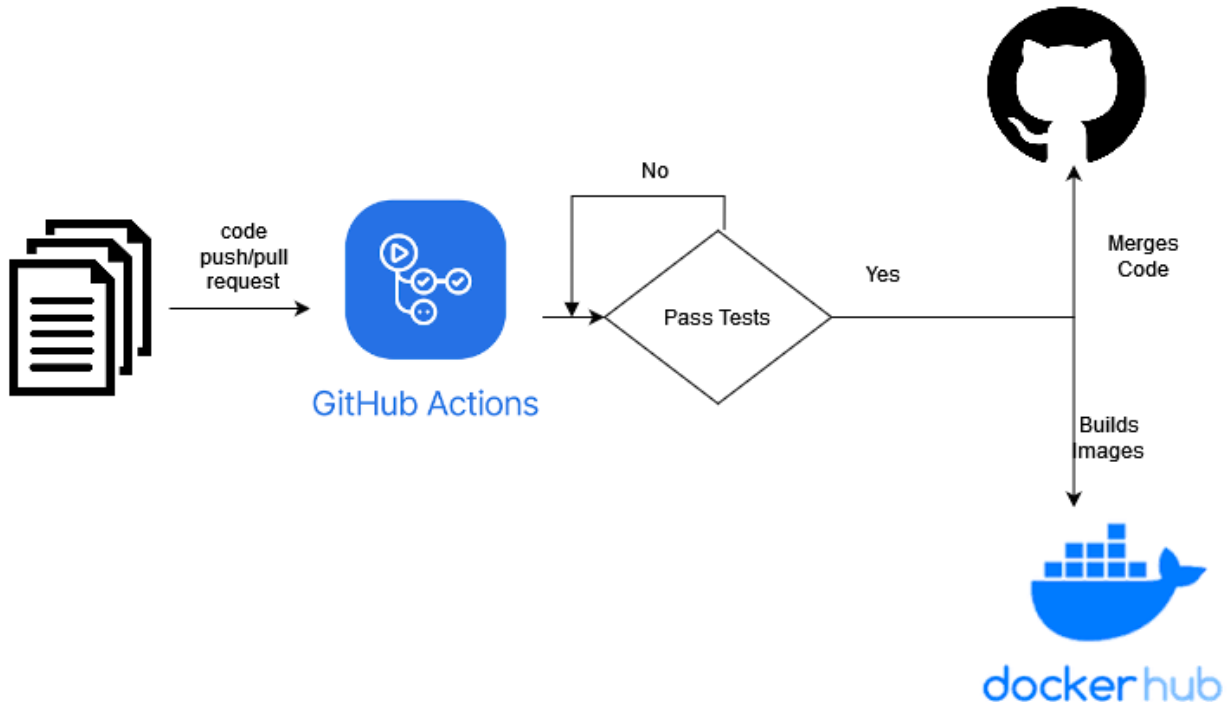
After some discussion, we decided on the Microservice architecture due to the potentially complex nature of our project due to each logical component having their own complex requirements which would make having less coupling between them easier to develop.

## 5.2 Deployment

	Docker Compose	Kubernetes
Benefits	<ul style="list-style-type: none"><li>• Can be deployed on a single host</li><li>• Simpler to set up</li></ul>	<ul style="list-style-type: none"><li>• Is deployed across multiple hosts</li><li>• More complicated to set up</li><li>• Has auto-scaling</li></ul>

We decided on using docker compose for our deployment due to being easier to set up and deploy.

## 5.3 CI/CD



Our team used github actions as our CI/CD tool because of its wide range of functionalities which suits our needs and can handle both deployment and testing of code.

### Continuous Integration (CI)

Our continuous integration tests the code using the mocha and chai testing libraries to help ensure that any code that has been pushed to master or any pull requests that are targeted to master works properly and does not break the existing functionality of the different services.

### Continuous Integration (CD)

The continuous deployment is triggered when there are any changes to the files of a particular microservice. The workflow builds a new docker image which is committed to docker hub which allows for any deployment to be able to pull the latest image for deployment at any time.

## 5.4 Architecture Pattern

### Database per service Pattern

Each microservices uses their own database so that the database is suited for their individual needs and to reduce coupling between the services. This allows the different services to have their own database that best suits their individual needs and ensures any changes to the database does not impact the other services.

However, we did not use the same database for each service.

For our Matching Service, we chose SQLite. We felt that it was better than MongoDB & Redis as SQLite is relatively lightweight and compact. Although it is not as fast as Redis, it was fast enough to meet our requirements.

For our Chat Service, we chose MongoDB over Redis as we wanted to persist our data. This enables us to potentially implement features such as allowing users to continue conversations after the session or storing chat logs to check for toxic behavior in the future.

MongoDB was chosen over SQLite as it can handle higher traffic than SQLite and can handle concurrent reads and writes better<sup>4</sup>, which will occur when multiple collaboration sessions are going on at the same time.

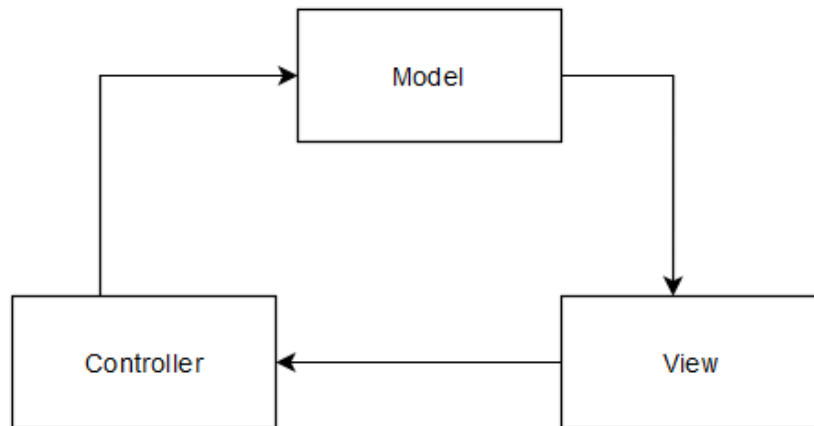
For the Question Service, MongoDB was chosen over Redis & SQLite for the similar reasons, requirement for data persistence and ability to handle higher traffic.

---

<sup>4</sup> <https://www.sqlite.org/whentouse.html>

## MVC

We used the Model-View-Controller pattern in all microservices of our application.



All three components of MVC have different functions. Model is used to handle data logic. View displays the information from model to user and controller directs the flow of information into model components and triggers view update accordingly. These distinct functions allow us to achieve separation of concerns where each component fulfills a non-trivial role and this allows for better code manageability.

## Pub-Sub Pattern

The team's Collaboration, Chat and Matching service uses Socket.io which is an implementation of the pub-sub pattern. Pub/sub is an asynchronous messaging service that facilitates bi-directional communication between server and client sockets.

In our applications, Socket.io allows for client and server sockets to listen for incoming events and react accordingly. For example, when a code change event is triggered, the server will receive this notification and broadcast the changes to all sockets in a particular room. This allows for a speedy exchange of information to support the low latency of our application.

The pub-sub pattern allows for scalability. The decoupled nature of pub/sub pattern allows for communication logic to be separated from business logic.

## 6. Services

### 6.1 Frontend

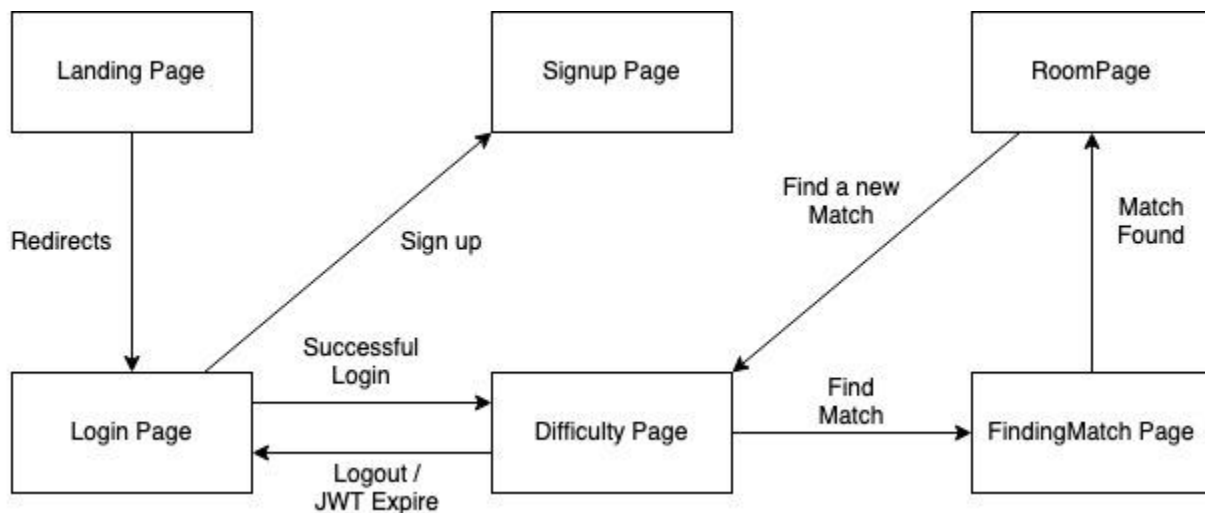
Component	Library/Framework Used
Frontend	React
Components	Material UI

We used the Material UI component library as it provides a wide range of beautiful frontend components with easy and flexible customization. This reduces the amount of code we have to write to create & style our own basic components from scratch. The MUI library also makes it easy to define breakpoints so each component can be customized to fit different screen sizes.

#### User Authentication

The user authentication consists of two pages which are the sign in and registration page.

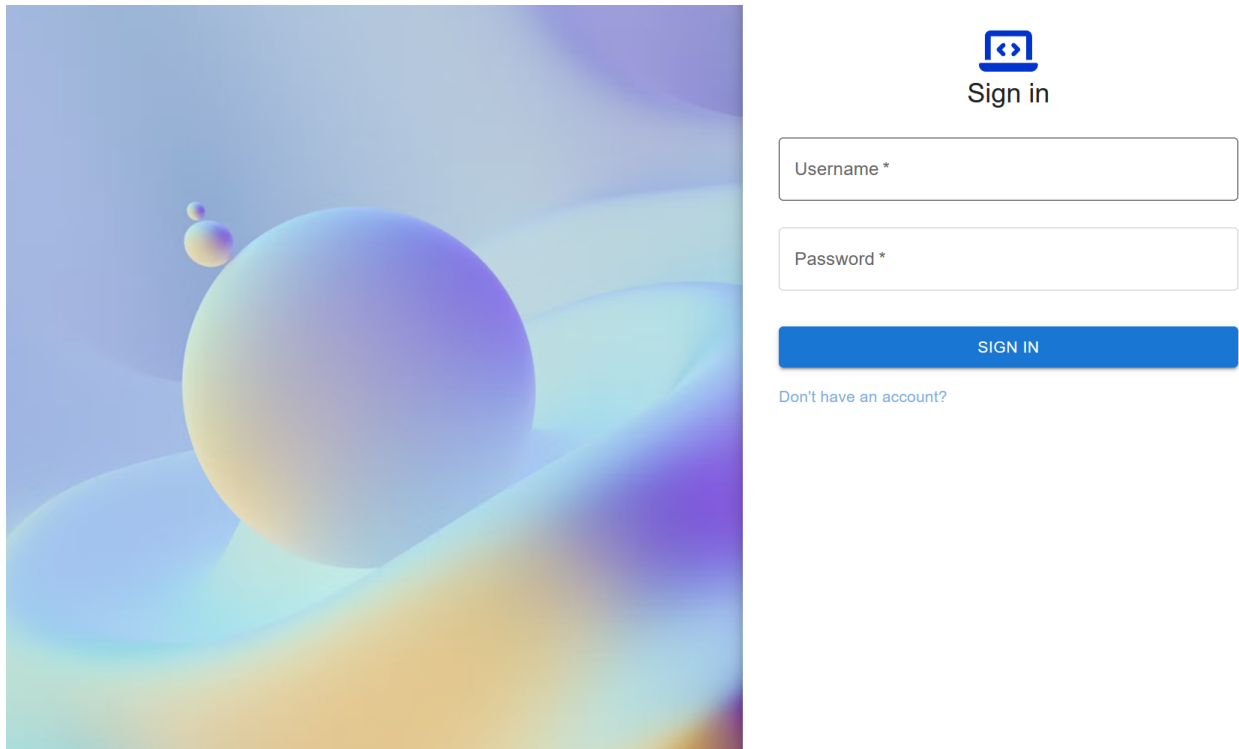
The sign in page requires the user to enter the credentials of an existing account in order to login.





The image above shows how each page transitions to the next. Note that our landing page does not exist, but it merely redirects the user to the Login Page (/login) when the user accesses the base url of our Frontend Service.

Our application has an App component that acts as a controller to provide routes for each of our 5 pages. Each page uses smaller custom components we have made (such as CodeEditor) as well as components from the MUI library.



The signup requires the user to sign up with a new unique username and a password that meets the complexity requirements.




Sign up



SIGN UP

Already have an account? [Sign in](#)

## Difficulty Page

 MATCH U

Choose your difficulty!

EASY  
★☆☆

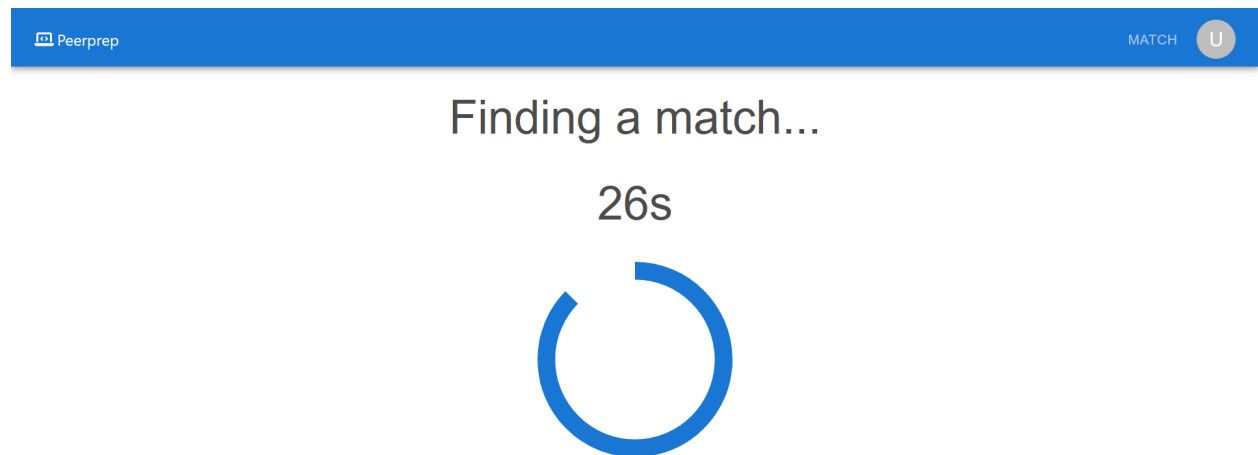
MEDIUM  
★★★

HARD  
★★★★

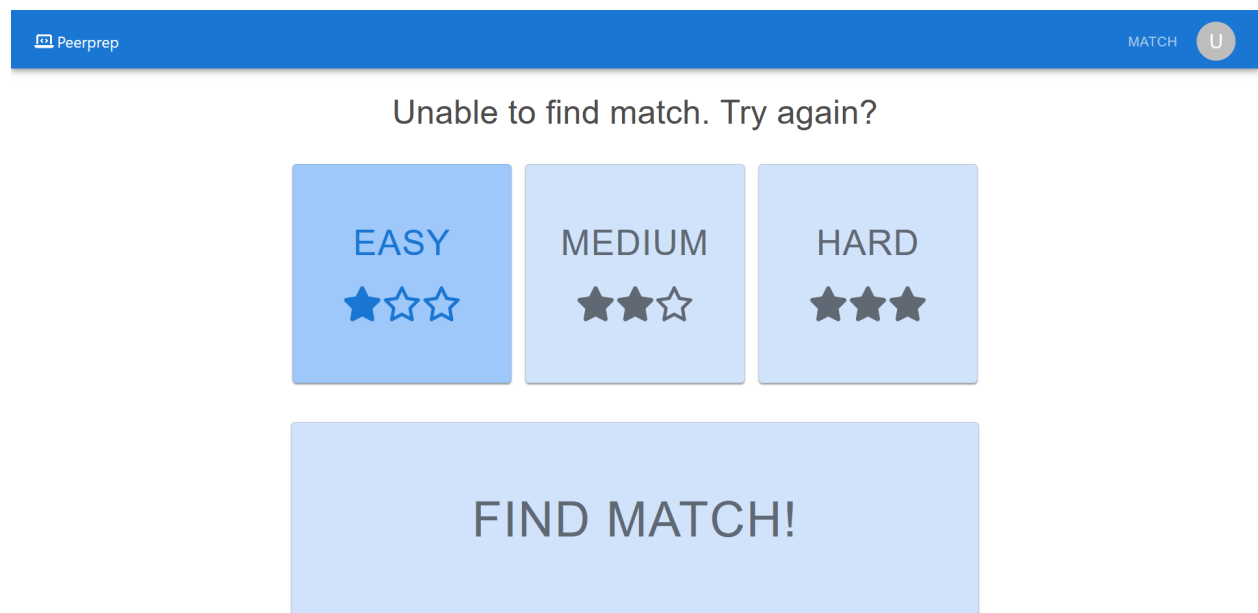
START CODING!

This page allows users to select their difficulty and initiate match finding by clicking the “Start Coding” button.

## FindingMatch Page



While finding a match, users are directed to the FindingMatch Page with a timer to display how much time is left.



If a match is not found within 30s, the user is allowed to select a difficulty level again and try to find a match.

If another user who has selected the same difficulty level is found, a match is made and both users are directed to the same Room Page with the same random question.

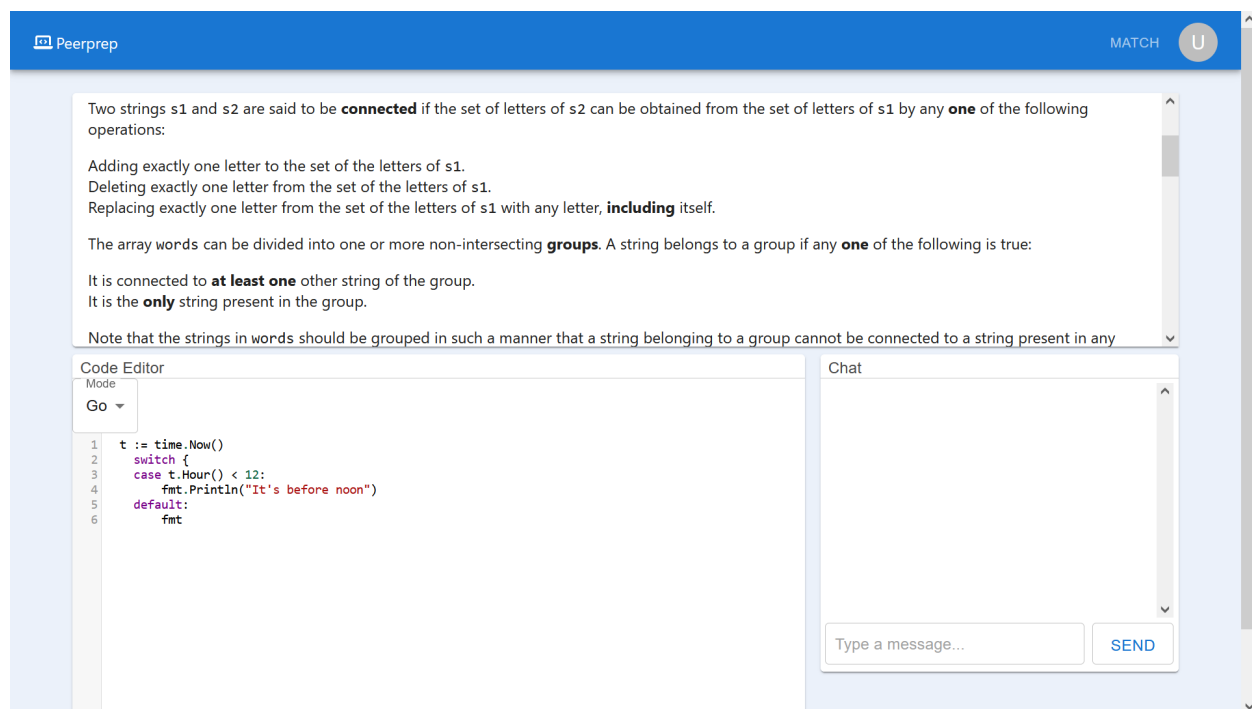
## Room Page

The interview room page consists of three main elements which display the question, a collaborative code editor and a chat component.

The question displayed is a random question of the selected difficulty level.

Editing text in the code editor occurs in real-time and offers syntax highlighting capabilities for several popular languages (Javascript, Python, Go) in order to enhance user experience.

The chat component also provides real-time messaging as well as multiline messages to allow users to structure their communication. In addition, the chat scrolls to the latest message whenever a new message is received or sent.



## 6.2 Matching Service

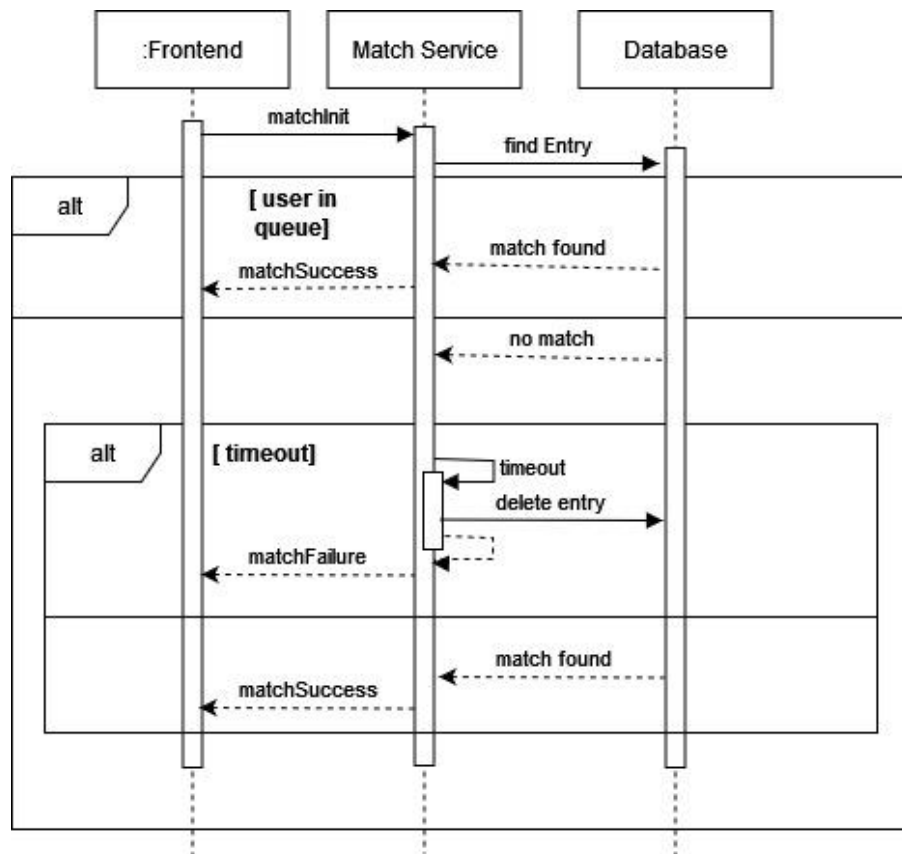
The matching service manages the matching of two users based on the difficulty selected. socket.io is used for communication between the user and the matching service server while sequelize is used for the matching server to store users in the queue while waiting for a match.

### Tech Stack

Component	Library/Framework Used
Backend	Node.js
Database	sequelize

### Process

The following sequence diagram shows the flow of how the matching service works



## matchInit

Sample Format:

```
{
  Username: "user1",
  Difficulty: "easy"
}
```

Sent from the server to the client on successful match.

```
{
  status: "success",
  message: "Random Question retrieved successfully",
  title: "2332. The Latest Time to Catch a Bus",
  questionDesc:
    ""PGRpdiBjbGFzc2oiY29udGVudF9fdTNJMSBxdWVzdGlvb11jb250ZW50X19KZmdSlj48ZGI2PjxwPllvdSBhcmUgZ2l2ZW4gYSA8c3Ryb25nPjAtaW5kZXhlZDwvc3Ryb25nPjBpbmRlZ2VlIGFycmF5IDxjb2RlPmJ1c2VzPC9jb2RlPiBvZiBsZW5ndGggPGNvZGU+bjwvY29kZT4sIHdoZXJlIDxjb2RlPmJ1c2VzW2ldPC9jb2RlPiByZXByZXNlbnRzIHROZSBkZXBhcnR1cmUgdGltZSBvZiB0aGUgPGNvZGU+aTxzdXA+dGg8L3N1cD48L2NvZGU+IGJ1cy4gWW91IGFyZSBhbHNvIGdpdmVulGEgPHN0cm9uZz4wLWluZGV4ZWQ8L3N0cm9uZz4gaW50ZWdlciBhcnJheSA8Y29kZT5wYXNzZW5nZXJzPC9jb2RlPiBvZiBsZW5ndGggPGNvZGU+bTwwY29kZT4sIHdoZXJlIDxjb2RlPnBhc3NlbmdlcuNba08L2NvZGU+IHJlcHJlc2VudHMgdGhllGFycml2YWwgdGltZSBvZiB0aGUgPGNvZGU+ajxzZXh0aGUgPGNvZGU+IHBhc3Nlbmdlc0gQWxsIGJ1cyBkZXBhcnR1cmUgdGltZXMgYXJlIHVuaXF1ZS4gQWxsIHh0aGUgPGNvZGU+Y2FwYWNpdHk8L2NvZGU+LCB3aGljaCBByZXByZXNlbnRzIHROZSA8c3Ryb25nPm1heGltZW08L3N0cm9uZz4gbnVtYmVylG9mlHBhc3NlbmdlcuMgdGhhdCBjYW4gZ2V0IG9ulGVhY2
  message: "Random Question retrieved successfully"
}
```

Sent from the server to the client if no match has been found after 30 seconds

```
{
  message: "failed to find match"
}
```

## 6.3 User Service

The User Service provides the restful APIs and data persistence to manage user related information. It supports key functions such as user account creation, password changing, validation of Json Web Tokens (JWT). The service utilizes BCrypt encryption algorithm to hash passwords for added security. User details include username and hashed password. These details could be further enhanced to store contact information.

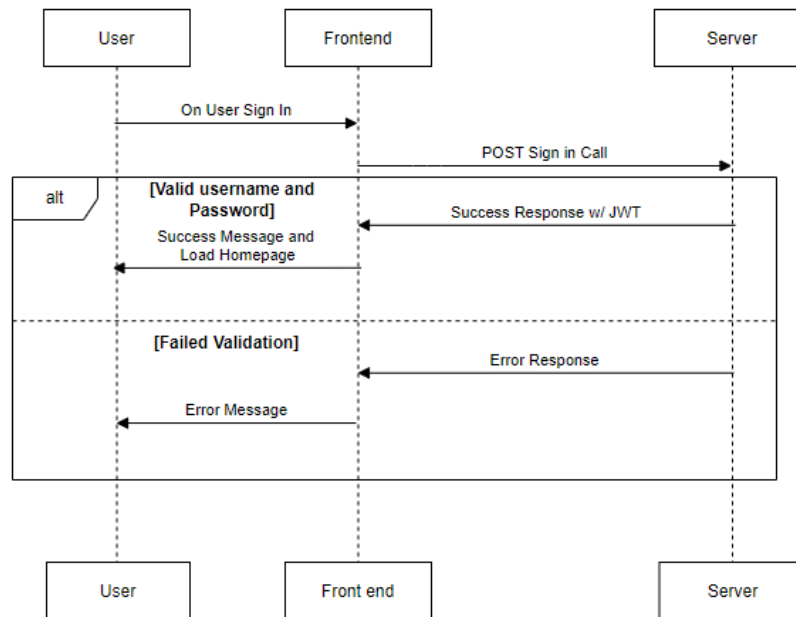
### Tech Stack

Component	Library/Framework Used
Backend	Node.js
Database	Redis, MongoDB
Hashing Algorithm	Blowfish Cypher
Authentication Scheme	JWT

### Process

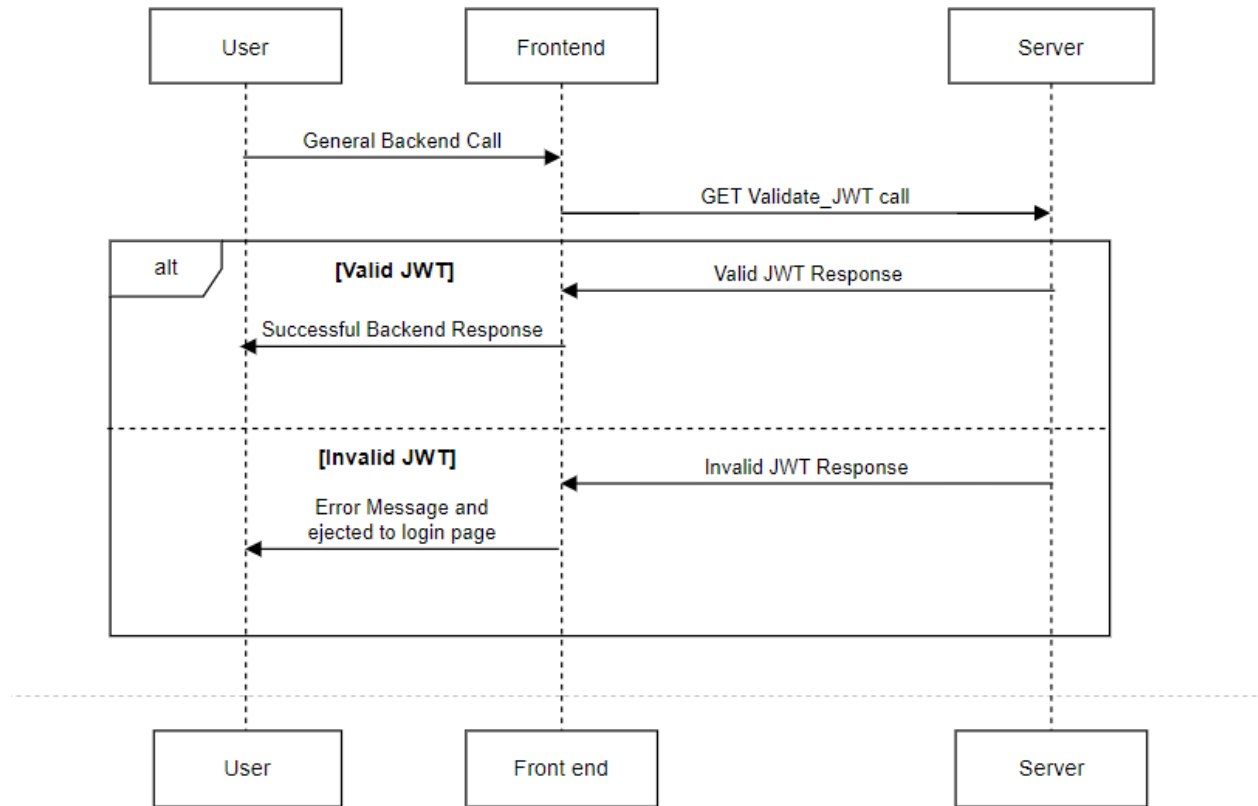
The flow is that upon successful sign-in, the user will receive the JWT token in the form of a HTTP response. This token will be stored in the local memory of the browser. Subsequently, backend calls such as changing of password will require a valid JWT token.

The following sequence diagram shows the process in which users receive their JWT token.



The following sequence diagram shows the validation process for every backend call to ensure that stored JWT token is valid and not expired or blacklisted.





Additionally, checks for expired and blocked tokens which are stored in an in-memory Redis cache are executed to reinforce the validity of the tokens held by clients.

## Alternatives for Consideration

Choice for User Authentication	
1. Token-based Authentication (JWT)	JWT created and issued by the server. Token is then stored client-side and included in every backend call.
2. Session-based Authentication	Session ID created by server and stored in both client and server applications. Server verifies the session ID and the corresponding entry in its memory.
Final Choice: JWT	Token-based authentication scales better than session-based authentication as server-side memory consumption is much lesser. The drawback of session-based authentication is that as number of concurrent users grow, conscientious effort to ensure that there is sufficient memory to store the numerous session IDs must be made. Hence, our team chose JWT due to its lighter footprint server-side.

## **REST Endpoints:**

The details of User Service's endpoints are provided below.

### **Create User**

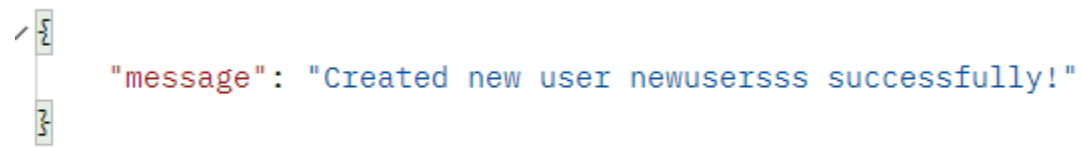
Endpoint: <Domain>/api/user

Request Type: POST

Input: JSON object with username and password

Example Request: localhost:8000/api/user

Example Successful Response:



```
{
  "message": "Created new user newusersss successfully!"
}
```

### **Response:**

status: States whether the request was successful or not.

message: Provides users with more information regarding the creation of their user

### **Log-in**

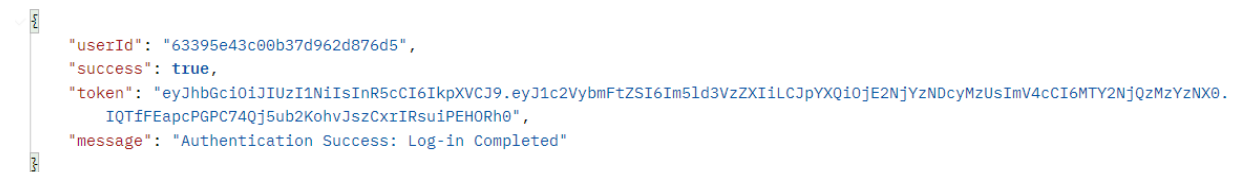
Endpoint: <Domain>/api/user/login

Request Type: POST

Input: JSON object with username and password

Example Request: localhost:8000/api/user/login

Example Successful Response:



```
{
  "userId": "63395e43c0b37d962d876d5",
  "success": true,
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im5ld3VzZXIiLCJpYXQiOiJlY2NjYzNDcyMzUsImV4cCI6MTY2NjQzMzYzNX0.IQTiFEapcPGPC74Qj5ub2KohvJszCxrIRsuiPEH0Rh0",
  "message": "Authentication Success: Log-in Completed"
}
```

### **Response:**

status: States whether the request was successful or not.

message: Provides users with more information regarding the log-in attempt

token: JWT token used for validation

### **Delete User**

Endpoint: <Domain>/api/user/deleteUser

Request Type: POST

Input: JSON object with username and password, JWT token in header

Example Request: localhost:8000/api/user/deleteUser

Example Successful Response:

```
{
  "userId": "63395e43c00b37d962d876d5",
  "success": true,
  "message": "User deleted"
}
```

### **Response:**

status: States whether the request was successful or not.

message: Provides users with more information regarding the delete attempt

userID: UID for user within database

### Log-out

Endpoint: <Domain>/api/user/logout

Request Type: POST

Input: JWT token in header

Example Request: localhost:8000/api/user/logout

Example Successful Response:

```
{
  "messsage": "User logged out w/ JWT invalidated"
}
```

### **Response:**

message: Provides users with more information regarding the log-out attempt

## 6.4 Collaboration Service

The Collaboration Service enables users to code in a shared editor with a near real-time response.

### Tech Stack

Component	Library/Framework Used
Backend	Node.js with Express Framework
Database	Redis
Messaging Scheme	Socket.io Pub/Sub
Additional	Codemirror

### Database Considerations

Choice for Database	
1. Redis In-memory Cache	Redis stores data in RAM and read/write operations are extremely fast.
2. Persistent Data Storage such as MongoDB	Persistent data storage is more robust and could handle larger amount of data.
Final Choice: Redis	Redis is chosen as the main memory storage as it is not needed for each coding sessions' details to be persistent. In addition, given the faster speed of read/write operations of Redis, it allows us to achieve near real-time response for the collaborative code editor.

Also, Socket.io is chosen for its pub-sub mechanism to facilitate instant event notifications for clients and servers. This allows clients/servers to react spontaneously accordingly to the event notifications. Hence, a real-time code editor where code changes are relayed to the other party immediately is created.

## 6.5 Question Service

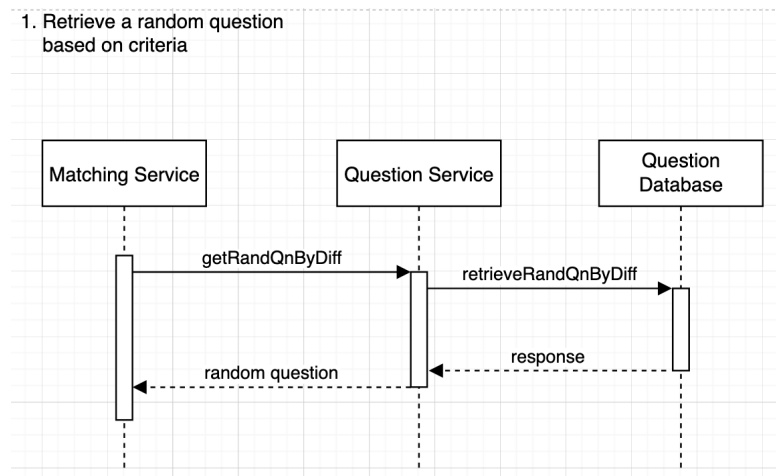
The Question Service is a microservice that provides a random question that matches the difficulty specified in its request from its database.

### Tech Stack

Component	Library/Framework Used
Backend	Node.js
Database	mongoDB

### Design:

The Question Service was designed as a microservice that provides a REST API. It is composed of the microservice itself as well as its own database that holds all the questions.



When the matching service manages to find a match for 2 clients, it makes a request to the Question Service REST API for a random question at a specific difficulty level. Question Service queries the database for a random question at that difficulty level and returns it to the Question Service, which then passes it to the Matching Service.

### REST Endpoints:

#### Get Random Question of Specified Difficulty

Endpoint: <Domain>/api/randomquestion?difficulty=<difficulty>

Request Type: GET

Example Request: localhost:5200/api/randomquestion?difficulty=Easy

## Example Successful Response:

```
{
  "status": "success",
  "message": "Random Question retrieved successfully",
  "data": [
    {
      "_id": "6342e4b1f97726c1c476179e",
      "title": "94. Binary Tree Inorder Traversal",
      "questionDesc": "'PGRpdiBjbGFzc20iY29udGVudF9fdTNJMSBxdWVzdGlvbi1jb250ZW50X19KZmdSIj48ZGl2PjxwPkdpmVuIHRoZSA8Y29kZT5yb290PC9jb2RlPiBvZiBhIGJpbmFyeSB0cmVLLCByZXR1cm4gPGVtPnRoZSBpbm9yZGVyIHRyYXZlcnNhbCBvZiBpdHMgbm9kZXNlIHZhbnVlcwVZW0+LjwvcD4KPHA+wqA8L3A+CjxwPjxz dHJvbmc+RXhhbXBsZSAx0jwvc3Ryb25nPjwvcD4KPGltZyBhbHQ9IiIgc3JjPSJodHRwciovL2Fzc2V0cy5sZWV0Y29kZS5jb20vdXBsb2Fkcy8yMDIwLzA5LzE1L2lub3JkZXJfMS5qcGciIH N0eWxlpSJ3aWR0aDogMTI1cHg7IGhlaWdodDogMjAwcHq7Ii8+CjxwcmU+PHN0cm9uZz5JbnB1dDo8L3N0cm9uZz4gcm9vdCA9IFsx LG51bGwsMiwzXQo8c3Ryb25nPk91dHB1dDo8L3N0cm9uZz4gWzEsMywyXQo8L3ByZT4KPHA+PHN0cm9uZz5FeGFtcGxldi6PC9zdH Jvbmc+PC9wPgo8cHJlPjxz dHJvbmc+SW5wdX06PC9zdHJvbmc+IHJvb3QgPSBbXQo8c3Ryb25nPk91dHB1dDo8L3N0cm9uZz4gW10K PC9wcmU+CjxwPjxz dHJvbmc+RXhhbXBsZSAz0jwvc3Ryb25nPjwvcD4KPHByYT48c3Ryb25nPk lucHV00jwvc3Ryb25nPiByb290ID 0gWzFkCjxz dHJvbmc+T3V0cHV00jwvc3Ryb25nPiBbMV0KPC9wcmU+CjxwPskgPC9wPgo8cD48c3Ryb25nPkNvbN0cmFpbmRz0jwv c3Ryb25nPjwvcD4KPHVsPgo8bGk+VGhlIG51bWJlciBvZiBub2RlcjYpbiB0aGUgdHJlZSBpcyBpbiB0aGUgcFuz2UgPGNvZGU+Wz AsIDEwMF08L2NvZGU+LjwvbGk+CjxsaT48Y29kZT4tMTAwICZsdDs9IE5vZGUudmFsICZsdDs9IDEwMDwvY29kZT48L2xpPgo8L3Vs Pgo8bD7CoDwvcD4KPHN0cm9uZz5Gb2xs3cgdXA6PC9zdHJvbmc+IFJlY3Vyc2l2ZSBzb2x1dGlvbiBpcyB0cm12aWFsLCBjb3VsZC B5b3UgZG8gaXQgaXRlcmF0aXZlbHk/PC9kaXY+PC9kaXY+PGJyPjxicj48aHI+PGJyPg=='",
      "difficulty": "Easy",
      "topic": [
        "Stack",
        "Tree",
        "Depth-First Search",
        "Binary Tree"
      ],
      "_v": 0
    }
  ]
}
```

## Response:

status: States whether the request was successful or not.

message: States if question was successfully retrieved.

data.\_id: unique identifier for question

data.title: title of question

data.questionDesc: Valid HTML string in UTF-8 Charset encoded in Base64 converted to a string

data.Difficulty: Difficulty of the question

data.topic: List of topics that the question belongs to.

## 6.6 Chat Service

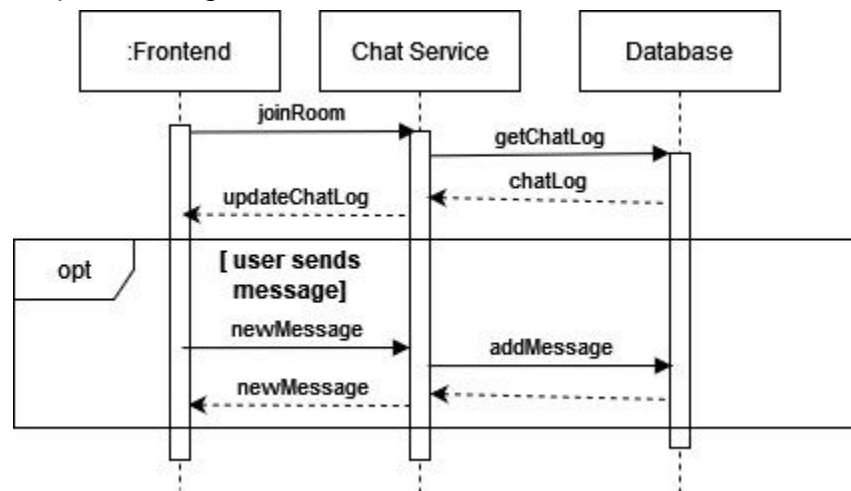
The chat service is a microservice that uses socket.io for communication between the server and clients. It allows users to send text messages to one another so that users can communicate with each other to discuss questions together.

### Tech Stack

Component	Library/Framework Used
Backend	Node.js
Database	mongoDB

### Process

The following sequence diagram shows the the flow of how the chat service works



When a client loads the page containing the chat service, it sends a `joinRoom` event to the server with the room ID to join. The server would then add that socket to a room using that room ID before sending the chat log of that particular room. Subsequently, any new messages send from the client will be added into the database before the server sends back the same message to the client as confirmation that the message has been sent successfully.

## **Websocket Events**

List of event types in the chat microservice and their data formats.

### **newMessage**

Sent between the clients and server for any new messages

Sample format:

```
{
  roomId: '4bb59cf6-e0ee-4cac-965a-df3dc4f3ccd6',
  message: 'new message',
  username: 'user2'
}
```

### **updateChatLog**

Provides clients with the chat history if any on connection

Sample format:

```
{
  _id: new ObjectId("6351039c281f906c0ce0b917"),
  roomId: '4bb59cf6-e0ee-4cac-965a-df3dc4f3ccd6',
  username: 'user1',
  message: 'message from user 1',
  sent_date: 2022-10-20T08:15:24.681Z,
  __v: 0
},
{
  _id: new ObjectId("63510674a0a29ac644d263af"),
  roomId: '4bb59cf6-e0ee-4cac-965a-df3dc4f3ccd6',
  username: 'user2',
  message: 'new message',
  sent_date: 2022-10-20T08:27:32.698Z,
  __v: 0
}
```

### **joinRoom**

Sent by the client to join a particular chat room

Sample format:

```
{
  roomId: "4bb59cf6-e0ee-4cac-965a-df3dc4f3ccd6"
}
```



## 7. Testing

### Unit Testing

Unit testing was done using the Mocha and Chai frameworks. There are unit tests written for the various microservices to ensure that the different API endpoints work as expected.

```
15 Db connected successfully
16 Running Question Service on port 5200
17
18
19 Question Service tests
20   Question Service Valid REST API Calls
21     ✓ it should GET random Easy Question
22     ✓ it should GET random Medium Question
23     ✓ it should GET random Hard Question
24   Question Service Invalid REST API Calls
25     ✓ it should safely fail to GET invalid difficulty value question
26     ✓ it should safely fail to GET invalid URL
27
```

```
16 Chat service tests
17   Chat controller test
18 Room joinedA1BCDEFG
19 []
20   ✓ join Room test (109ms)
21 { roomId: 'A1BCDEFG', username: 'user', message: 'Hello World' }
22   ✓ send message test
23 message added[object Object]
24
```

# Load Testing

In order to ensure that our Question Service is able to respond to requests for random questions within 2 seconds, we decided to stress test it using hey.

```
+ Downloads ./hey_darwin_amd64 -n 100000 -c 100 -z 5m http://localhost:5200/api/randomquestion\?difficultly=Hard
```

```
Summary:  
Total:          300.0179 secs  
Slowest:        0.2311 secs  
Fastest:         0.0011 secs  
Average:         0.0300 secs  
Requests/sec:   4611.9177  
  
Total data:     5347115107 bytes  
Size/request:   5347 bytes  
  
Response time histogram:  
0.001 [1] |  
0.024 [730204] | ████████████████████████████████████████████  
0.047 [255274] | ████████████████████████████████████████  
0.070 [12407] | █  
0.093 [1602] | |  
0.116 [305] | |  
0.139 [97] | |  
0.162 [18] | |  
0.185 [44] | |  
0.208 [44] | |  
0.231 [4] | |  
  
Latency distribution:  
10% in 0.0147 secs  
25% in 0.0172 secs  
50% in 0.0202 secs  
75% in 0.0246 secs  
90% in 0.0309 secs  
95% in 0.0360 secs  
99% in 0.0505 secs  
  
Details (average, fastest, slowest):  
DNS+dialog:    0.0000 secs, 0.0011 secs, 0.2311 secs  
DNS-lookup:    0.0000 secs, 0.0000 secs, 0.0060 secs  
req write:     0.0000 secs, 0.0000 secs, 0.0383 secs  
resp wait:     0.0300 secs, 0.0011 secs, 0.2310 secs  
resp read:     0.0000 secs, 0.0000 secs, 0.0381 secs  
  
Status code distribution:  
[200] 1000000 responses
```

In our example above, we had 100 concurrent workers send a total of 100k requests to our Question Service over 5 minutes. As shown, all 100k requests are resolved successfully with status 200 and the slowest request takes 0.2311 secs, thus we meet the time requirement in NFR1.3.

## 8. Remarks

### 8.1 Potential Improvements / Enhancements

#### Interviewer / Interviewee / Collaborator Roles

Since there are 2 users in every group, we can leverage this by giving a pair of users corresponding interviewer / interviewee roles. This will be useful as this can allow users to practice as an interviewee to give them a more realistic preparation experience.

However, we also leave the option of both of them being collaborators open, in case they would like to just work on the problem together.

#### Voice Chat

Most technical interviews are conducted with voice communication (Zoom, etc...). Thus, to further facilitate a mock interview experience, voice chat would further enhance the experience.

Furthermore, even as collaborators, it is often faster to speak to communicate ideas / debug than it is to communicate over text. Thus, voice chat would make the overall user experience smoother as well.

### 8.2 Reflections, Challenges & Learning Points

#### Manpower

We had a sudden disappearance of a teammate, halfway through the project. This threw a wrench in our plans so we had to adapt our sprint schedule to reallocate the work. This event further solidified the importance of Agile as it allowed us to continue working on different parts of the project and change our requirements on the fly while we simultaneously tried to catch up on his workload. Had we used a waterfall approach, the rest of us would be blocked and it would be harder to change requirements as those would have been set.

#### Time management

More buffer time could have been allocated for our development as there was a bit of a rush towards the submission deadline as minor bug fixes and slight changes had to be made in preparation for submissions.