CS3219 Software Engineering Principles and Patterns

AY22/23 Semester 1

Project Report

Team G52

Team Members	Student No.	Email
Lye Yi Xian	A0200813X	e0407794@u.nus.edu
Lim Zi Yang	A0199420B	e0406401@u.nus.edu
Marcus Duigan Xing Yu	A0199347L	e0406328@u.nus.edu
Lim Hern Fong, Jared	A0222603W	e0560135@u.nus.edu
Nikki Than Win	A0217715E	e0543751@u.nus.edu

Repository Link: https://github.com/CS3219-AY2223S1/cs3219-project-ay2223s1-g52
Deployed Website: https://frontend-rob2padjya-de.a.run.app

1. Introduction

1.1 PeerPrep

PeerPrep is a web application to help students ace their coding interviews through real-time collaborative coding, peer reviews and peer mock interviews. Our main feature is our matching system according to question difficulty and collaboration page. Features include a chat feature to facilitate peer reviews and peer mock interviews and a real time code editor for matched users.

1.2 Motivation & Purpose of the Project

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, you are tasked to create an interview preparation platform and peer matching system called PeerPrep, where students can find peers to practice whiteboard-style interview questions together.

2. Contributions

Member	Technical Contributions	Non-Technical Contributions
Lye Yi Xian	Implement Matching Service Implement Collaboration Service Implement Question Service Implement History page (Frontend)	Requirements for Matching, Collaboration and Question service Project report
Marcus Duigan	Implement user service login Implement communication service	Project report Requirements for communication and user service Presentation slides
Lim Zi Yang	Implement Difficulty Selection Page (Frontend) Implement Collaborative Room Page (Frontend) Implement Collaborative Editor (Frontend) Implement Chat in Collaborative Room (Frontend) Implemented Solo Room (Frontend) Worked on Collaboration Service (Backend)	Project report Requirements for Frontend
Lim Hern Fong, Jared	Implement User Service (change password & delete user & create user) Profile Page Implement JWT and Cookie Auth Setup Devops	Project report Devops Documentation
Nikki Than Win	Implement Login Page (Frontend) Implement Sign Up Page(Frontend) Implement Question Pane on Collaboration Room Page (Frontend) Implement History Service (Backend)	Project report Requirements for History Service and FrontEnd

3. Functional Requirements

3.1 User Service

S/N	Functional Requirement	Priority	Issue No.
1.1	The system should allow users to create an account with username and password.	High	#4
1.2	The system should ensure that every account created has a unique username.	High	#3
1.3	The system should allow users to log into their accounts by entering their username and password	High	#5
1.4	The system should allow users to log out of their account	High	#6
1.5	The system should allow users to delete their account.	High	#7
1.6	The system should allow users to change their password	Medium	#8

3.2 Match Service

S/N	Functional Requirement	Priority	Issue No.
2.1	The system should be able to match two waiting users who picked the same difficulty and put them in the same room.	High	#15, #23
2.2	If there is a valid match, the system should match the users within 30s.	High	#17
2.3	If there is no match within 30 seconds, the system should inform the users that no match is available.	High	#16
2.4	The system should only allow one match per user at the same time	High	#27

3.3 Question Service

S/N	Functional Requirement	Priority	Issue No.
3.1	The system should provide a random question of a selected difficulty	High	#85
3.2	The system should allow user to view a selected question	Medium	#129

3.4 Collaboration Service

S/N	Functional Requirement	Priority	Issue No.
4.1	The system should allow user to code on the same editor	High	#84
4.2	The system should send notification when a user left/join the room	Medium	#89
4.3	The system should display the users that are in the room	Medium	#84
4.4	The system should allow the users to be in a room for 30 mins	High	#98

3.5 Communication Service

S/N	Functional Requirement	Priority	Issue No.
5.1	The system should allow matched users to chat with each other	High	#138

3.6 History Service

S/N	Functional Requirement	Priority	Issue No.
6.1	The system should allow users to view questions that they have solved	High	#124
6.2	The system should add the question that the user has solved into the history	High	#124
6.4	The system should display a message to indicate if a user has not attempted any question before.	Medium	#124

3.7 Frontend

S/N	Functional Requirement	Priority	Issue No.
1.1	System should allow user to login and store user's login JWT as cookie	High	#19
1.2	System should allow user to sign up as a new user	High	#36
1.3	System should redirect user to login page if not authenticated	High	#41
1.4	System should redirect user to login page once signed up	High	#46
1.5	System should redirect user select difficulty page once user login	High	#41
1.6	System should prevent user from creating account with a non-unique username	High	#41
1.7	System should allow user to change password	Medium	#42
1.8	System should allow user to delete account	Medium	#42
1.9	The system should ask users to double confirm when deleting their account	Medium	#59
1.10	System should differentiate the login page and log out page for user to identify	Very Low	#36, #59
2.1	System should allow user to select their desired difficulty level of questions	High	#20, #21, #38
2.2	System should allow user to match with another user through a matching page with countdown timer	High	#20, #21, #38
2.3	System should handle the situation where there is a match for the user	High	#20, #21, #38
2.4	System should handle the situation where there is no match for the user	High	#20, #21, #38
2.5	System should disconnect the user from matching when the user click away from matching modal	Low	#20, #21, #38
3.1	System should display a question at the difficulty selected to the user	High	#105
3.2	System should display the same question to both users in the same room	High	#105
4.1	System should allow the user to write on a editor in the same window as the question	High	#76
4.2	System should allow a pair of matched users to work on	High	#76

	the editor concurrently		
4.3	System should allow user to leave the collaboration room via a button	High	#120
4.4	System should display countdown timer for user in the collaboration page	Medium	#91
4.5	System should allow reconnection of user if user accidentally leaves the collaboration page	Medium	#76 ,#91, #119
4.6	System should display the names of users currently working in the collaboration room	Medium	#76, #119
4.7	System should alert the user if there are issues connecting to the network in the collaboration room	Medium	#76
4.8	System should allow flexible resizing of question panel or editor	Low	#76
5.1	System should allow a pair of matched users to send messages to each other in a chat	Medium	#125
5.2	System should notify user if there are any unread messages in chat	Low	#125
5.3	System should show the timestamp of when the messages are sent in the chat	Low	#125
5.4	System should allow user to send messages in chat in a similar way compared to conventional chat application, for example to send messages via “Enter” button and use “Ctrl Enter” to create new lines	Low	#125
6.1	System should display history of user’s questions to user	Medium	#127
6.2	System should record question in user’s history once completed	Medium	#120
6.3	The system should display a message to indicate if a user has not attempted any question before.	Medium	#124
6.4	The system should display question history sorted by earliest time first.	Medium	#124
6.5	System should allow user to reattempt completed questions individually	Low	#132

4. Non-functional Requirements

We categorised our non-functional requirements under 4 main categories

1. Performance
2. Availability
3. Usability
4. Scalability
5. Security

In terms of prioritisation we decided to prioritise based on this order:

Performance > Usability > Availability > Scalability > Security

Justification:
Ultimately our web application is to be used in a scenario where users collaborate on code in real time. Hence performance is the key priority ast they should not face any issues related to latency as this drawback will be the most obvious fallacy in our project since it will directly affect the user experience.

We chose usability as our second prioritised requirement as we want the functionality to be user centric as this will indirectly affect user experience and would make users more likely to use our application over similar products.

We decided on availability for our next prioritisation since it is important for our application to be readily available to users so that they can consistently rely on our application for interview preparation. Having it available will also help retain usage since users will have the impression that our application is reliable.

Next is scalability. Our reason for prioritising it last is that even after finalising our project, in most cases for actual web development, it takes time for users to notice our web application and use it so traffic flow to our website would be slow initially and therefore manageable but once it picks up and the number of concurrent users start to increase, only then will scalability take priority. But given the stage we are at we decided to prioritise it last.

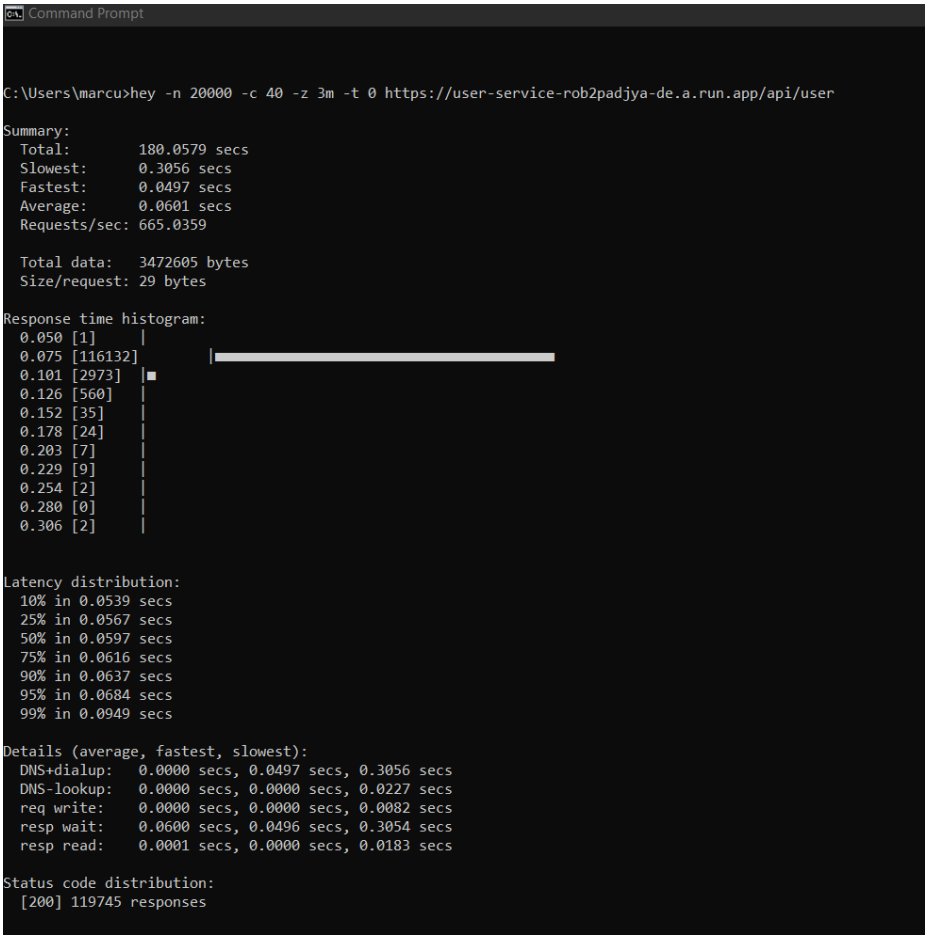
Lastly is security. Our reason for prioritising it last is that our application does not collect any sensitive data such as credit card information or handphone number so we feel like we should focus more on other aspects.

4.1 Performance Requirements

S/N	Non Functional Requirements	Priority
1.1	During collaboration of coding, delay between typing and code displaying on the UI should be less than 4 seconds	High
1.2	When switching from different frames in the web application, the time delay should not exceed 5 second	Medium
1.3	Retrieval of data from the database for any feature should not take longer than 5 second	Medium
1.4	Any standard API calls between frontend and backend should have a response time that does not exceed 2 seconds	Low
1.5	The latency in chat feature between sending and receiving messages should be under 1 seconds	High

Load Testing

To load test our application, we used a load testing tool called hey to test our application. We tested calling 20000 calls with 40 concurrent users. As it can be seen our average call took 0.06s and the slowest was only 0.3s, which meets our performance requirements.



sample of load stress testing results

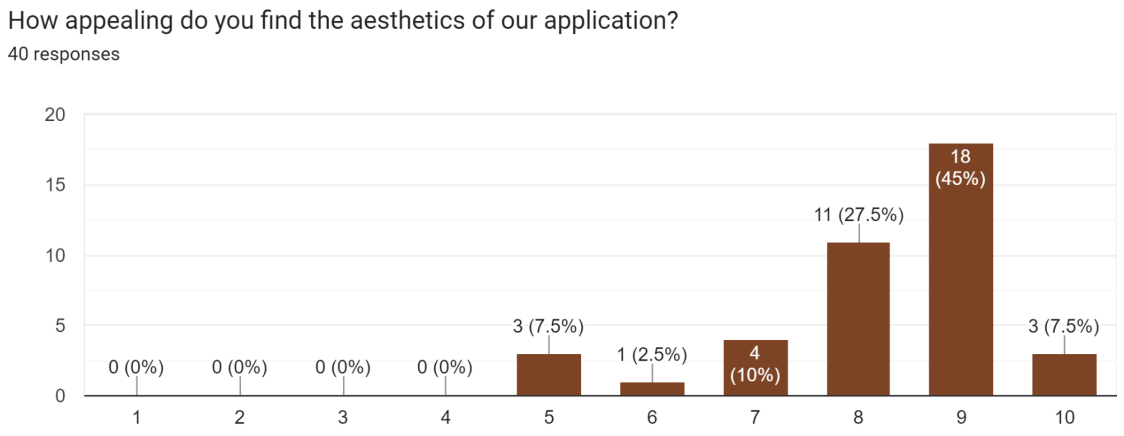
4.2 Usability Requirements

S/N	Non Functional Requirements	Priority
2.1	User-centric UI design that allows users to easily use features in the web application	High
2.2	UI design that adheres to the 7 Principles of Design. Link to design Principles	Medium
2.3	User can remain logged in for a certain period of time before they must re enter their credentials	Medium
2.4	The messages in the chat should be displayed sequentially	Low
2.5	The messages in the chat should show who sent the message.	Low
2.6	The messages in the chat should show the timestamp when it was sent	Very Low

To ensure that we met the requirements for usability, our team sat in COM 3 and invited random computing students to try out our application and fill out our evaluation survey.

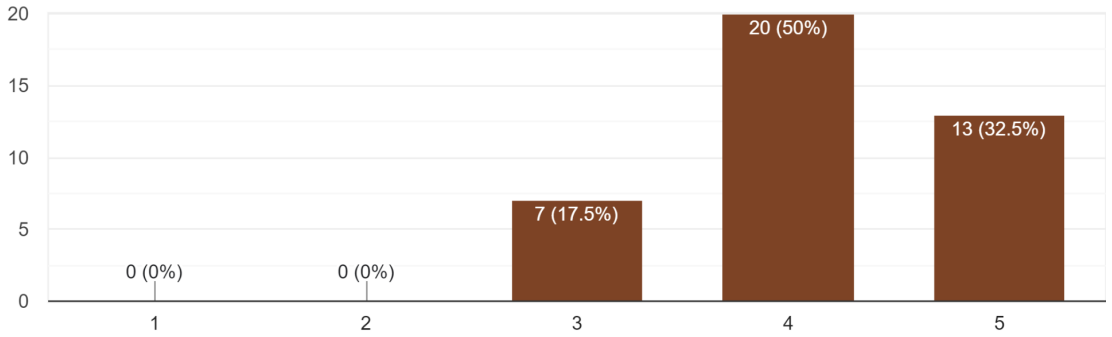
Link to google form: https://forms.gle/yewkhAhvw1Q6BWTX7

Below is a table of our results:

Results of our 10 Questions


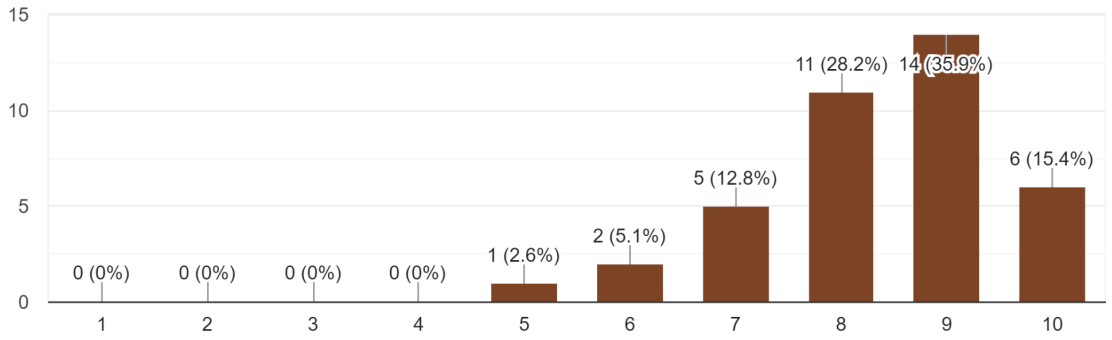
I find the features intuitive enough to use it

40 responses



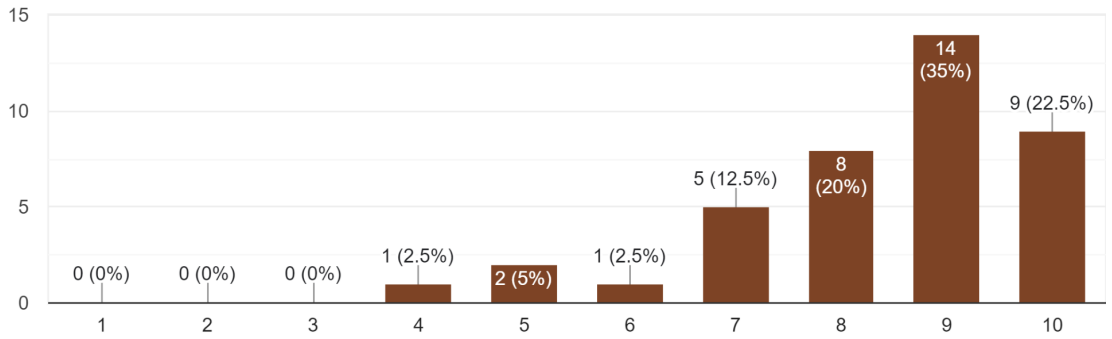
I found most of the features present useful

39 responses



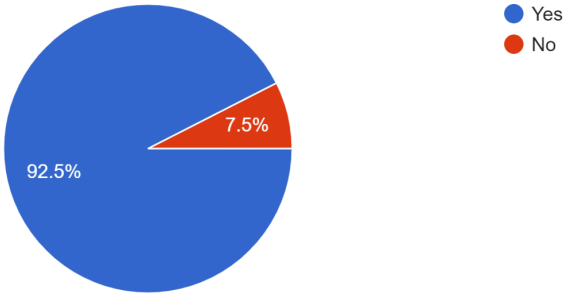
I understood how to use the application

40 responses



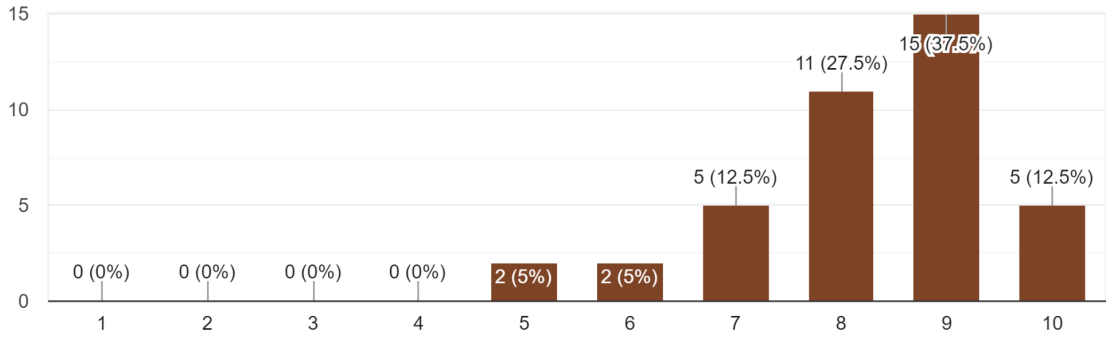
Did you find the system responsive enough to your liking?

40 responses



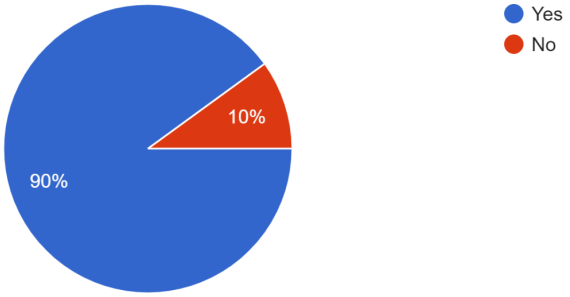
Did you find the layout and features of the collaboration page to be?

40 responses



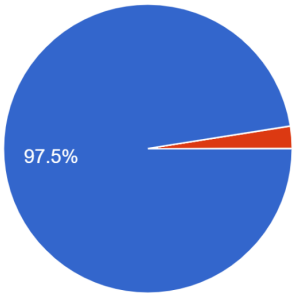
I can see many people using this system in the future

40 responses



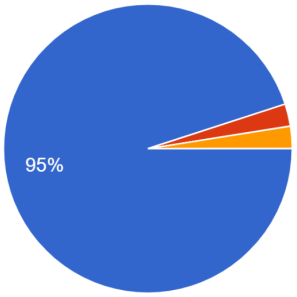
I found the system to be simple and not overwhelming

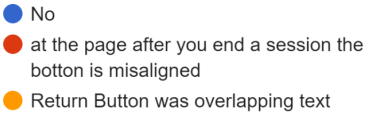
40 responses



Any bugs or visual errors to report?

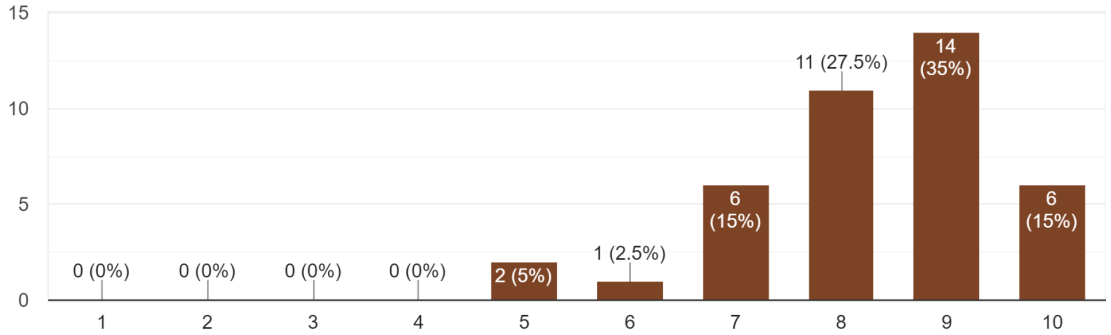
40 responses





How likely would use this application to prepare for my coding interviews

40 responses



Summary: Overall, those who tried our application rated it very highly with a large majority giving high points in every question. While there were some visual bugs spotted during our trial, the visual bugs were quickly updated and are no longer there. In terms of usability, based on our survey, we can confidently say that our usability has met the requirements that we set out on.

4.3 Availability Requirements

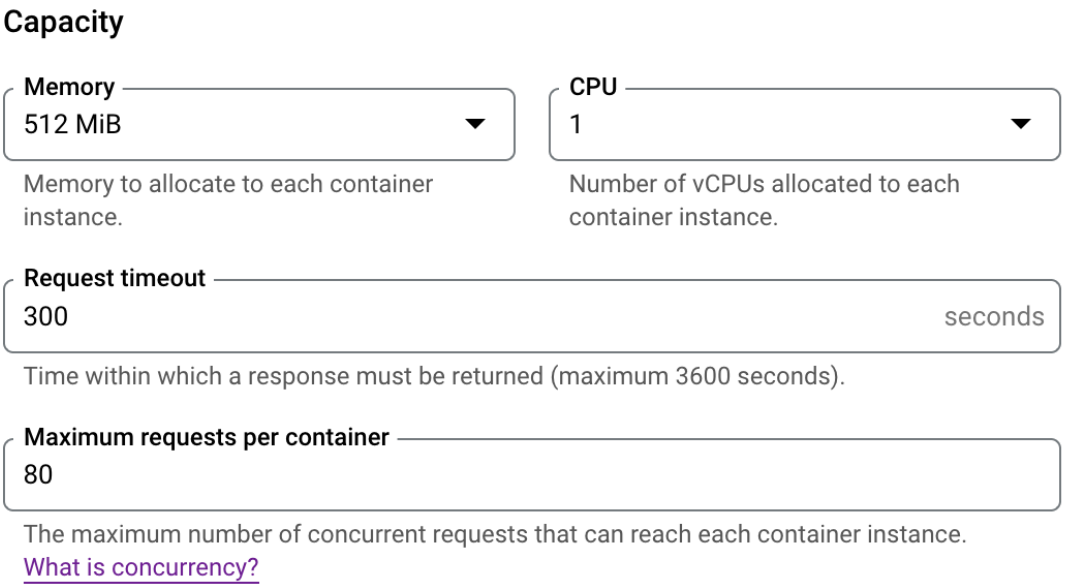
S/N	Non Functional Requirements	Priority
3.1	Application should have an uptime of 99% even under stress	High

For availability, our project's availability is dependent on the availability of our deployment service which is Cloud Run (uptime score of 99.9982 percent) and mongoDB Atlas (industry-leading uptime SLA of 99.995% across all cloud providers - MongoDB Atlas website). Since the majority of our dependencies are mainly on these services, we can safely conclude that our uptime can meet this requirement of > 99%.

4.4 Scalability Requirements

S/N	Non Functional Requirements	Priority
4.1	The application is able to handle 6 concurrent collaborations at any instance	High
4.2	The application should be able to handle at least 40 users using the application at the same time	Medium
4.3	The application should be able to register and save at least 600 accounts	Low

In Cloud Run, each service is automatically scaled to the number of container instances needed to handle all incoming requests. By observation of the stress tests we have conducted, the default configurations are more than sufficient to handle the performance and scalability requirements mentioned above. However, if necessary, Cloud Run also offers us numerous configurable options to increase hardware capability such as memory and cpu, and the number of concurrent requests per instance. This will allow our application to handle the future increase in loads and requests.



Hardware configurations in Cloud Run on GCP

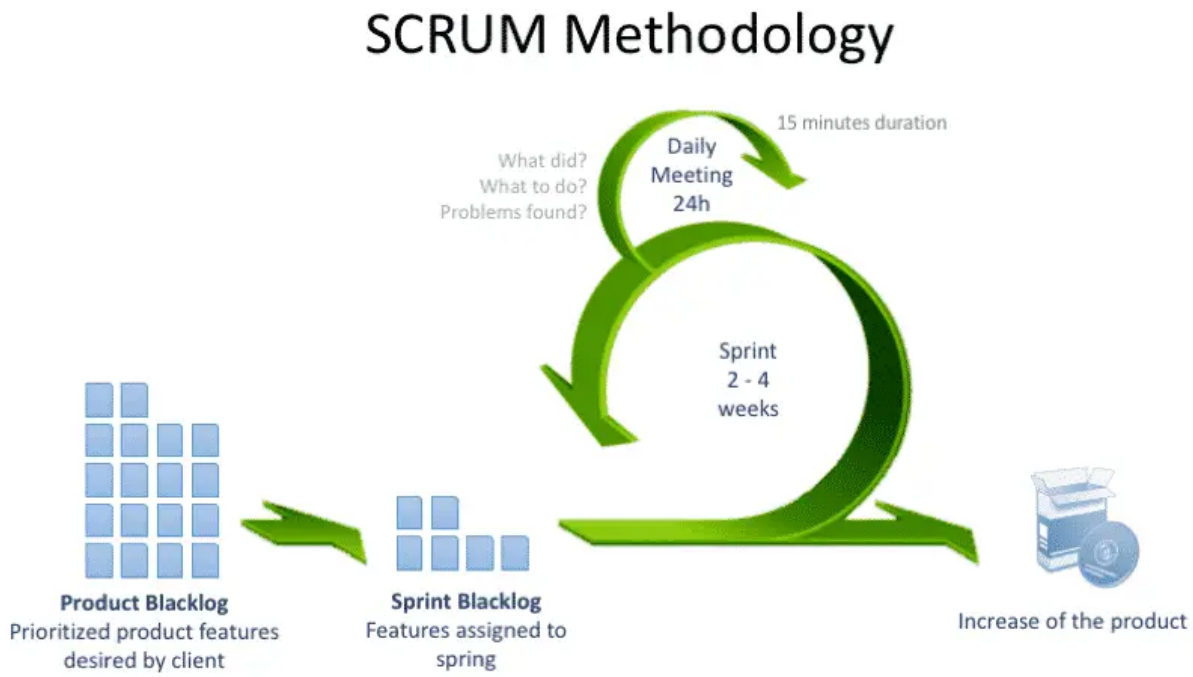
4.5 Security Requirements

S/N	Non Functional Requirements	Priority
5.1	Users should only be able to call some of the APIs if they are authenticated and logged in	High

Every feature in our application requires a check to verify the user using JWT tokens. If the user is not authenticated they will be redirected to the login page to re-enter their credentials.

5. DevOps

5.1 Scrum



Example of Scrum process (Reference)

We adapted the **Agile** Software Development Process Model in this project, more specifically we chose **Scrum**, a framework of the process.

Why **Scrum**? The features of the framework were relevant to the nature and scope of our project. The flexibility also allows for decisions and improvements in the design to be made along the way.

Our **Sprint** cycle is over the duration of 1 week, where we will complete the following:

1. Refine and identify our product & sprint backlog respectively
2. Develop and test our implemented features
3. Review code and provide feedback

We also hold a weekly scrum meeting on Wednesdays to encourage co-location among the team, ensure everyone is up to date with the project progress and provide assistance to one another if necessary. Below is a rough guide on what is done each scrum session:

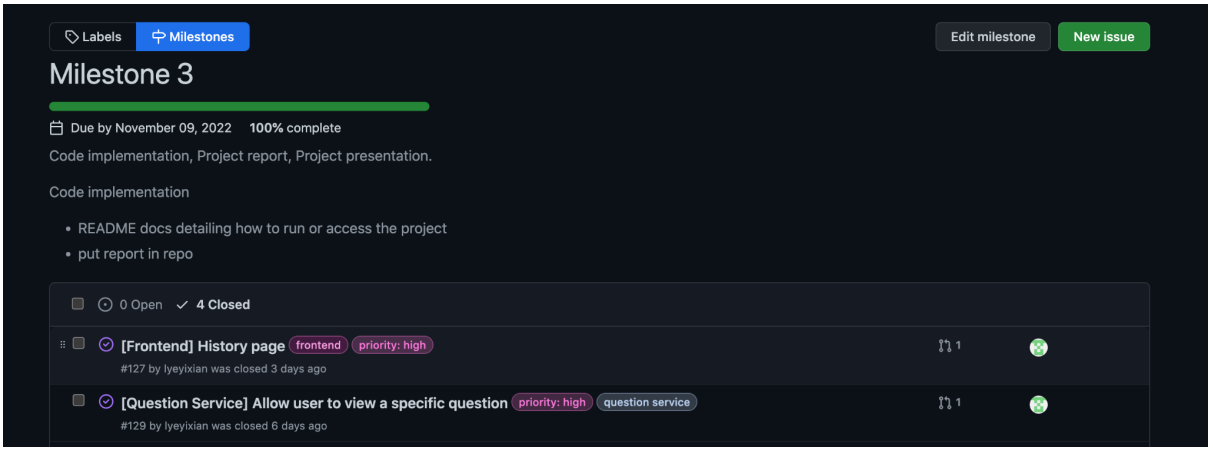
Time	Task Description
1500 - 1530	Update one another with progress and review the sprint
1530 - 1545	Refine product backlog

1545 - 1600	Identify next week's sprint backlog

Our Sprint Backlog is a simple table consisting of our tasks and duties for the week. It is used alongside our Github Milestone, Issues and Pull Requests to delegate and ensure transparency in our duties and work done.

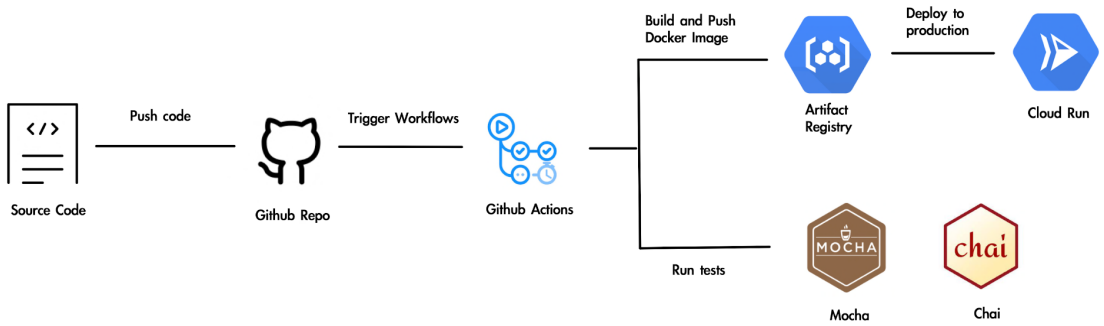
Recess Week	
Sprint Backlogs	**Allocated Person**
Unit Test User Service	Marcus, Jared
Unit Test Match Service	Yi Xian
Unit Test Frontend	Zi yang and Nikki
Setup CI using TravisCI/GitHub Actions	Jared, Yi Xian
Update Report	Everyone

Sample Sprint Backlog



Sample Github Milestone

5.2 CI/CD



CI/CD Pipeline

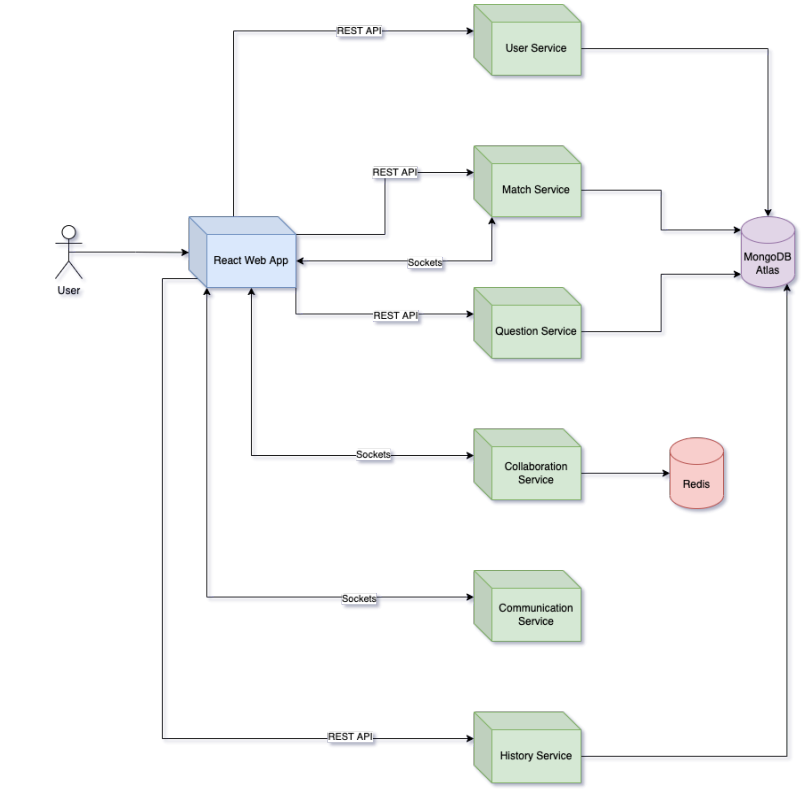
Our CI/CD pipeline mainly begins upon pushing our code to the **main** branch, or when a pull request (PR) is made to the **main** branch.

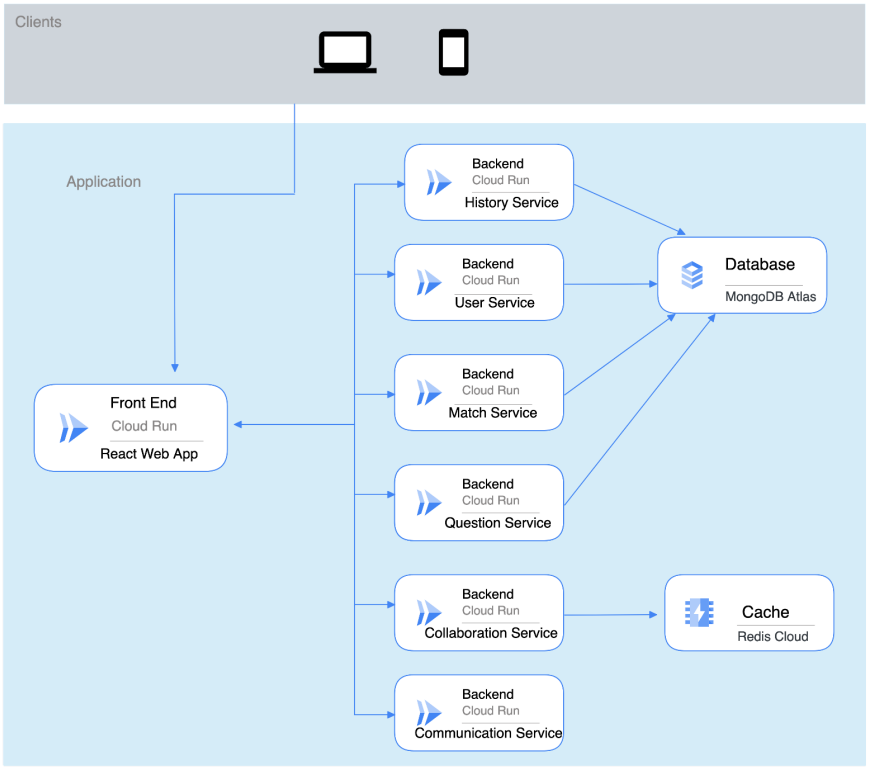
Once pushed, our Github Actions workflows will be triggered. These workflows are primarily dictated by the instructions written in **.yml** files located within our **.github/workflows/** directory. They are divided into two categories, CI and CD. The different microservices are also tested and deployed simultaneously to ensure an efficient pipeline.

CI files in our workflow will then instruct the tests to run and success or failures will be flagged accordingly in our PRs where merging will be blocked if tests failed. Our unit tests are written in Mocha, the JavaScript test framework, alongside with Chai, the assertion library.

CD files in our workflow will also instruct the deployment process of our pipeline. The Docker images will first be built according to the Dockerfiles present in the respective microservices directories. It will then be pushed to the Google Artifact Registry for storage. Subsequently, the container images will be deployed to production on Google Cloud Run.

6. Architecture Diagram





High level overviews of the system

7. Architecture & Patterns

7.1 Microservice Architecture

In our project, we chose the Microservices Architecture to help design our project. The major design decision was a choice between Microservices Architecture and Monolithic Architecture. The following are the benefits of having the Microservices Architecture.

Consideration of Team Size

- Our team consists of 5 people, hence choosing the Microservices Architecture benefitted us as the responsibility for developing the microservices could remain very separated.
- In a monolithic architecture, we would have had a harder time delegating responsibilities and could end up with many issues like merge conflicts and inconsistent patterns within all the services.

Reducing Coupling

- Microservices Architecture also allows us to reduce the coupling of different logical components.
- Additional major features could be implemented and deployed as new microservices without much downtime as the Main Application could continue running.
- Changes to each microservice could largely remain independent of other microservices and each microservice could be treated as a black box with other microservices or the client only being guaranteed what the input and output would be.

More Manageable Code

- Testing our code and managing bugs in the Microservices architecture was made easier as there is a clear separation of the code base based on the Microservice.

Better Scaling

- Each microservice is individually scaled based on the load received so scaling could be made a lot more efficient in our deployment.

7.2 MVC Pattern

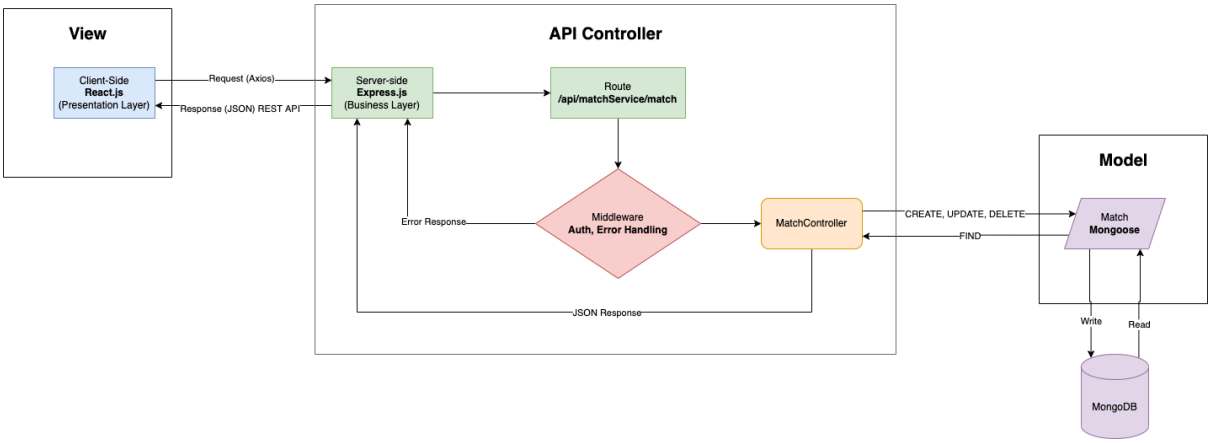
In our project, we applied the MVC pattern, specifically the Web MVC – SPA, in our User Service, Match Service, Question Service and History Service.

Separation of Concerns

- Output and UI presentation is handled by the View component which is in our Frontend using React.js
- Business logic of each individual microservices and incoming requests from the View component is handled by the Controller component using Express.js
- Database schema and data related logic is handled by the Model component using Mongoose ORM and MongoDB.

Open-Closed Principle

- Our design facilitates extensibility and we can easily extend the microservice with new features without modifying existing code.
- if we want to add a new resource to the service, we can easily add a new Route, Controller and Model for that resource.
- For example, if we want to have a new Friend feature. We can just add the following
    - Route: /api/matchService/friend
    - Controller: FriendController
    - Model: Friend



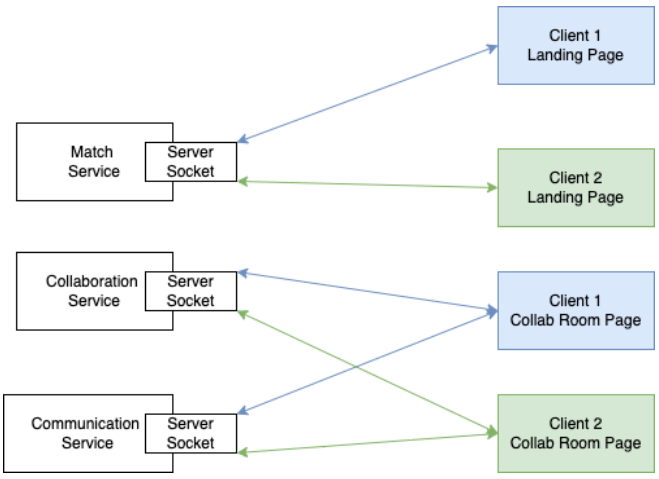
Matching Service Diagram

7.3 Pub-Sub Pattern

We applied the Publisher-Subscriber pattern in our Match Service, Collaboration Service, Communication Service where real-time bidirectional communication is needed. We use Socket.io for all the above services.

The services each contain a Server socket that allows the Frontend client to connect to it

- Match Service – Landing page
    - The server will subscribe to “matchFound” and “matchWaiting” topics and create a room for the two clients that connect to it. It will also handle the 30 seconds timer, publishing “failToMatch” topic and when a match is found, publishing “room” topic, which contains the room id to the two clients in the room.
    - The client will subscribe to “room” and “failToMatch” topics and connect to the Server socket. It will also publish either “matchFound” and “matchWaiting” according to the API response from Match Service.
- Collaboration Service – Collab Room page
    - The server will subscribe to “joinRoom” and “codeChanged” topics and similarly create a room using the room id from the clients. It will publish “userConnected”, “usersInRoom”, “codeUpdated”, “currentTime” and “userDisconnect” topics to update the state of the room and the content of the editor.
    - The client will subscribe to “userConnected”, “usersInRoom”, “codeUpdated”, “currentTime” and “userDisconnect” topics and display those states on the Collab Room page. It will publish “joinRoom” and “codeChanged” to indicate that a user just joined the room and the user made some changes to the code.
- Communication Service – Collab Room page
    - The server will subscribe to “joinRoom” and “chatMessage” topics to join the same room using the roomId and username to distinguish the user. The server publishes “message” where any incoming message will be emitted to the specific roomId
    - The client who will be in a room will subscribe to “roomUsers” and publish a “message” to indicate messages sent to be received by the server.



Socket Diagram

8. Tech Stacks

Domain	Technical Resources	Justification
Frontend	- ReactJS - Material UI	- Our members are most experienced in ReactJS compared to other frameworks - ReactJS allows for reusability of components, allowing us to reuse code and prevent repetitive code to produce same components - Material UI's existing components allow us to spend less time on developing the application by reusing the components provided - Material UI's customizability allows us to come up with creative designs for the web application. For example, the notification pop ups and dialog pop ups are components used from Material UI.
Backend	Express	- Most members have experience with express
In memory data structure store	Redis	- Redis provides low latency and high throughput. It also complements our non-functional requirements of high availability and scalability
Database	MongoDB	- Although new to some of us, some of our members have experience with mongoDB and it was recommended by the report
Pub-Sub Messaging	Socket.io	- Low-latency, bidirectional and event-based communication between a client and a server - The Rooms feature serve our needs of having isolated channel for matched users
Deployment	Cloud Run	- Cloud Run provides an easy deployment of microservices with minimal necessary service-specific configurations suitable in the context of our project. - Serverless platform that allows microservices deployed to scale automatically depending on the number of incoming requests. This is also

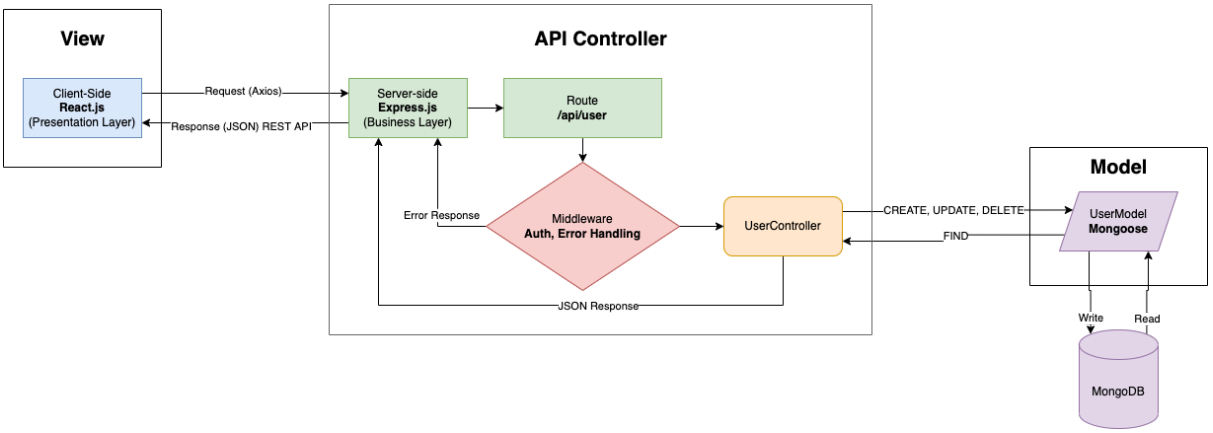
		customizable and can be sufficiently tailored to fit the needs of our project.
CI/CD	GitHub Actions	- Provides easy automation to customise and execute development workflows - Sample workflows and support for the tech stacks we have chosen are readily available
Testing	Mocha Chai React Testing Library	- Mocha, the JavaScript test framework, and Chai, the assertion library provide a comprehensive way for us to conduct unit tests on API functions within each of our microservices. - React testing library is a lightweight solution for the testing of our React components. It is sufficiently comprehensive to test our code implemented on the frontend.
Project Management Tools	GitHub Issues/ Google Docs	- All of our members have used github issues and google docs
Coding Style	- Prettier - ESLint - Husky	- Husky pre-commit hooks, combined with static code analyzer ESLint and code formatter Prettier, are used to check for problems in code and formatting issues before commits. This allows us to quickly identify and rectify problems with code.

9. Backend APIs

9.1 User Service

Port: 8000

This microservice stores user data and manages login, signup, etc.

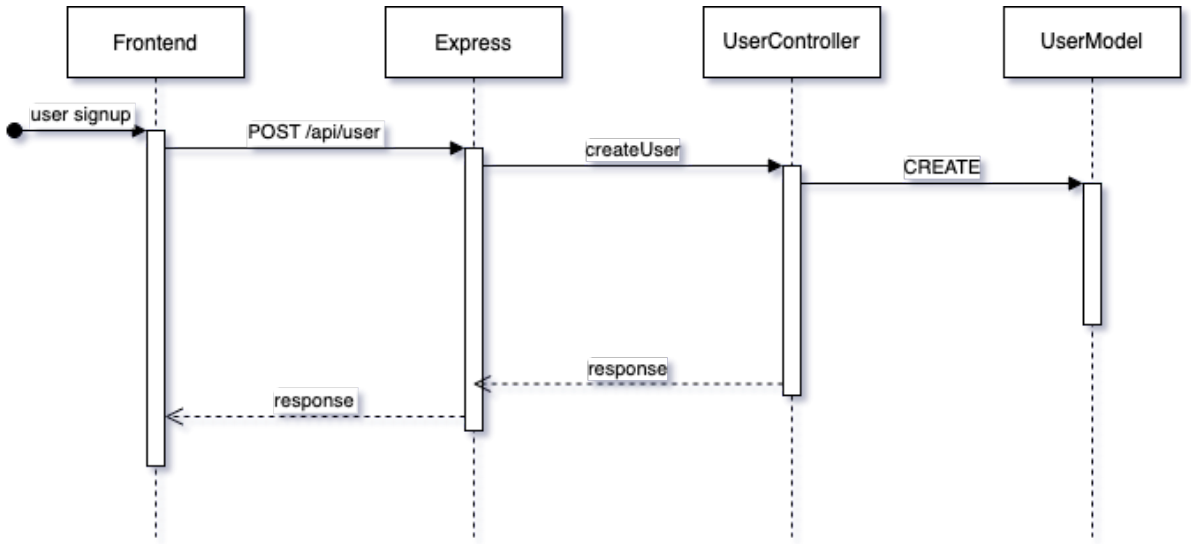


User Service Diagram

APIs

Create a user	
Method	POST
Route	/api/user
Request (JSON)	• username: String • password: String
Response (JSON)	• Message: String

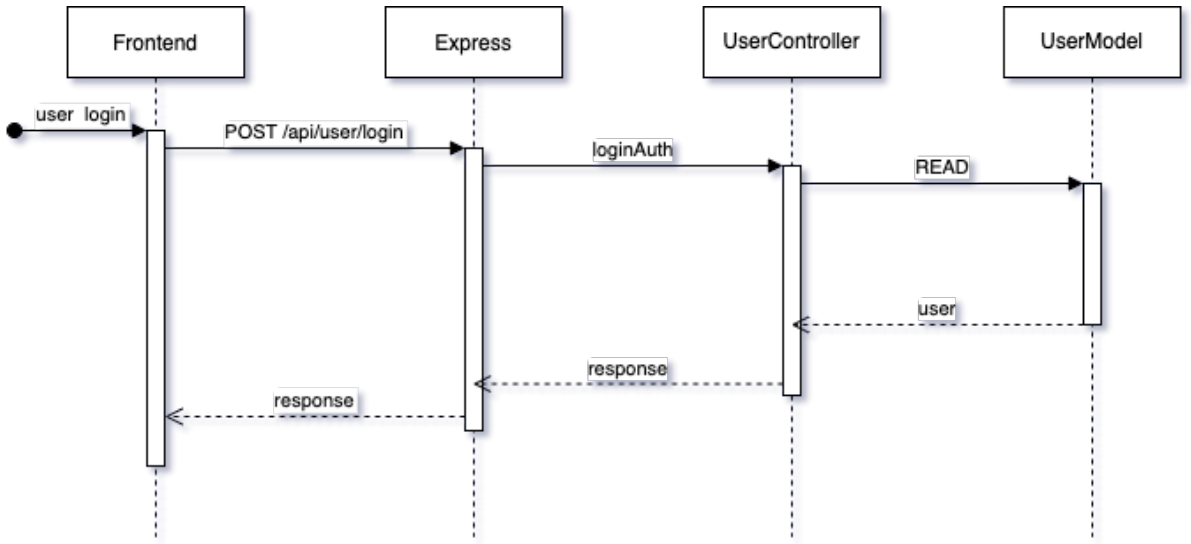
Example	Request:
	{
	"username": "miranda",
	"password": "1234dr"
	}
	
	Response:
	{
	“Message”: ”Created new user miranda successfully!”
	}



Sequence Diagram for creating a new user

Login with user credentials	
Method	POST
Route	/api/user/login
Request (JSON)	• username: String • password: String
Response (JSON)	• Username: String • Token: String

Example	Request:
	{
	"username": "miranda",
	"password": "1234dr"
	}
	
	Response:
	{
	"username": "miranda",
	"token":
	"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2MzUzOWRkYTA5ZGQ5ZjU2MjkxNGUzZmIiLCJpYXQiOjE2NjY0MjQ0OTIsImV4cCI6MTY2NjY4MzY5Mn0.1Te_JUWJMmp6n4UVguUS9Ie-RlHYiHWoXQNaDUrqT4Y"
	
	}



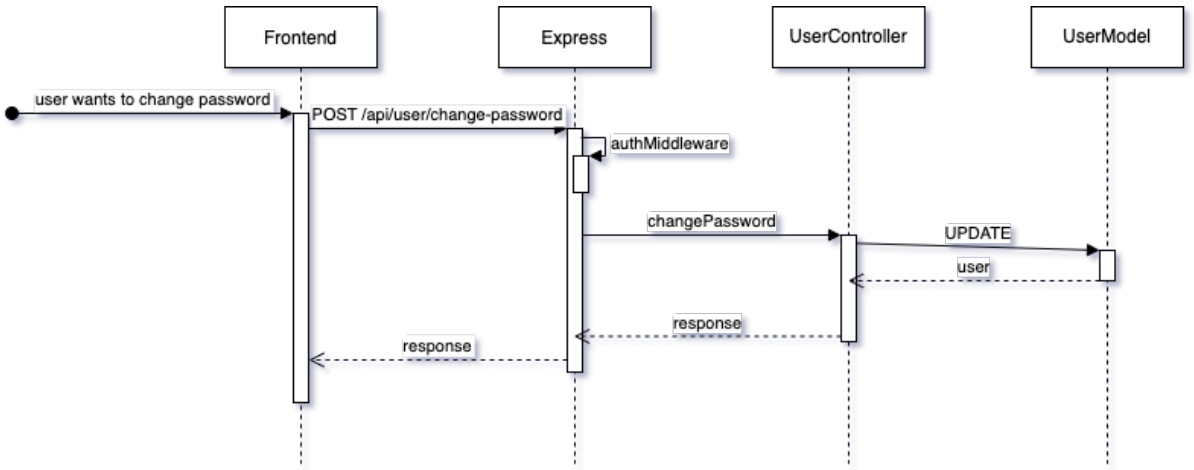
Sequence Diagram for login

Change password	
Method	POST
Route	/api/user/change-password

Header	{
	“Authorization”: `Bearer ${jwt}`
	}

Request (JSON)	• password: String
Response (JSON)	• message: String

Example	Request:
	{
	"password": "newpw"
	}
	
	Response:
	{
	“Message”: ”Password changed successfully!”
	}



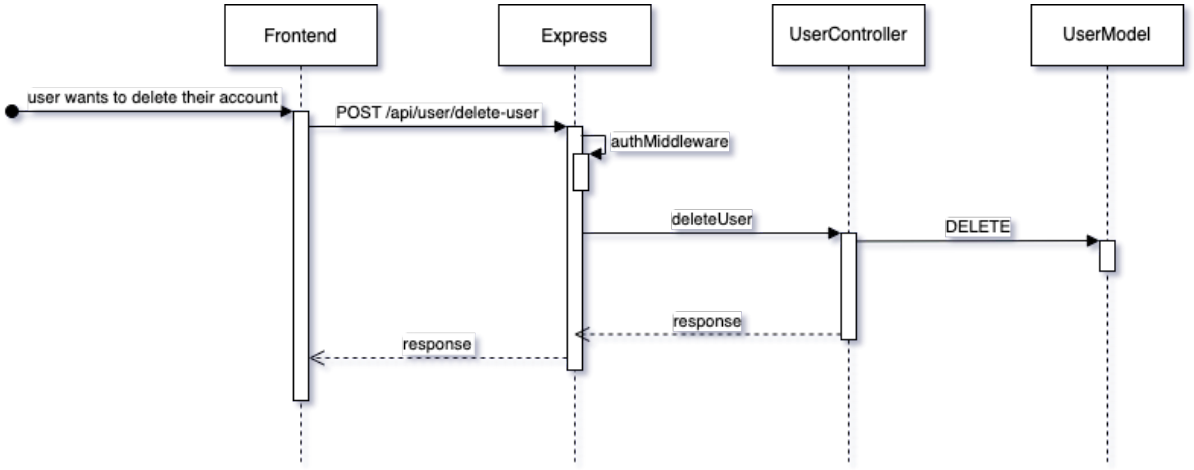
Sequence Diagram for changing password

Delete user	
Method	POST
Route	/api/user/delete-user

Header	{
	"Authorization": `Bearer ${jwt}`
	}

Response (JSON)	• message: String

Example	Response:
	{
	"Message": "User deleted successfully!"
	}



Sequence Diagram for deleting a user

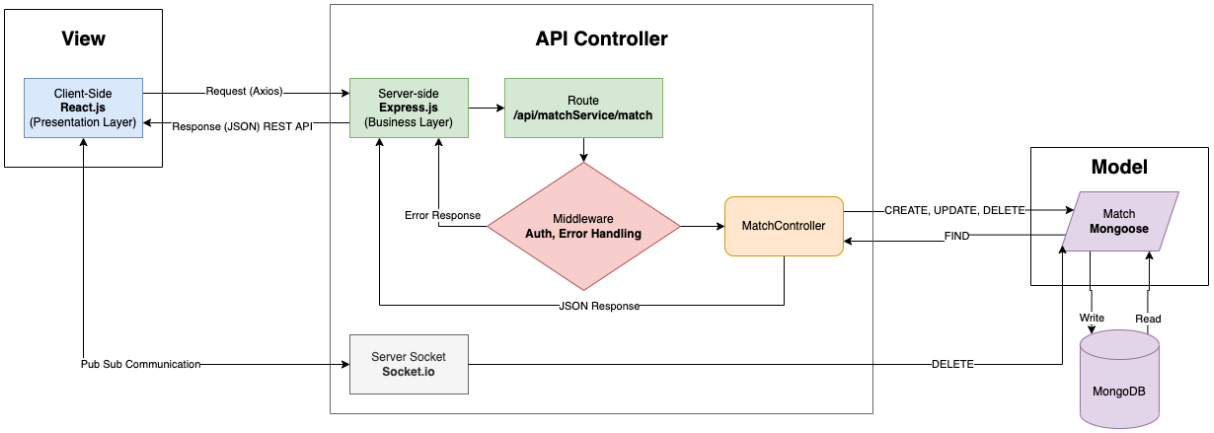
Database Schema

UserModel	
user (unique)	String
password	String

9.2 Match Service

Port: 8001

This microservice matches two users and returns a room id.

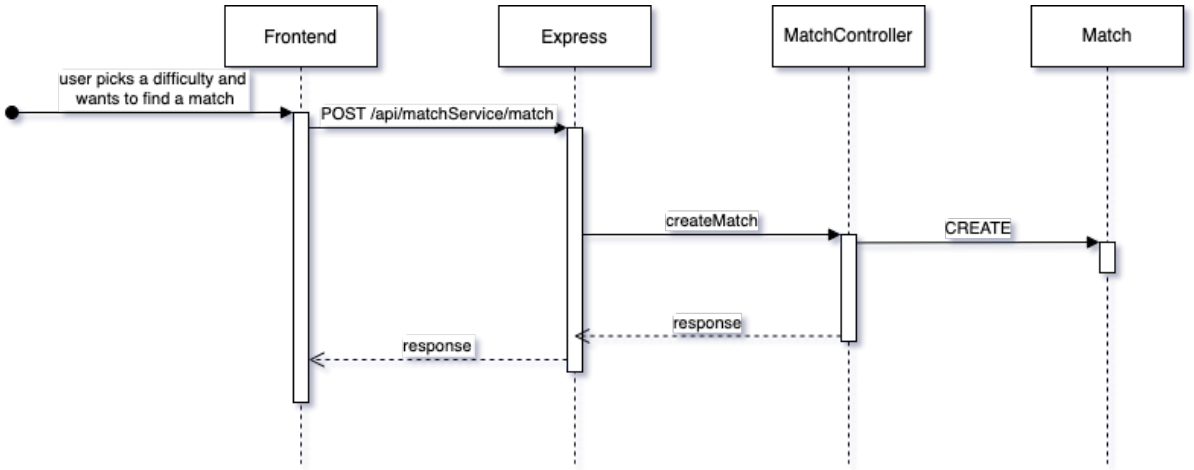


Match Service Diagram

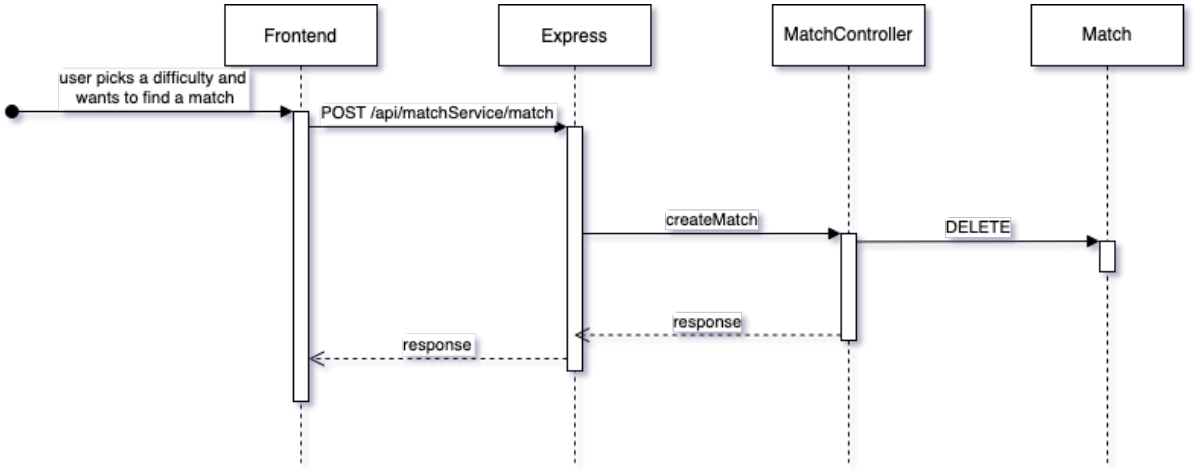
APIs

Find a match	
Method	POST
Route	/api/matchService/match
Request (JSON)	• difficulty: ‘easy’ | ‘medium’ | ‘hard’ • user: String
Response (JSON)	• id: String • room: String • isMatch: boolean

Example	Request:
	{
	"difficulty": "easy",
	"user": "abc123"
	}
	
	Response:
	{
	"id": "63539a14174292dba16ffd02",
	"room": "match-easy-4e070140-8fcf-49bf-a40d-dde960d5c1a0",
	"isMatch": **false**
	}



Sequence Diagram for finding a match



Sequence Diagram when a match is found

Sockets

Server

Subscribe	
Topics	**Data**
matchWaiting	• room: String
matchFound	• room: String

Publish		
Topics	**Target**	**Data**
room	every socket in the room	• room: String

failToMatch	sender	• msg: String

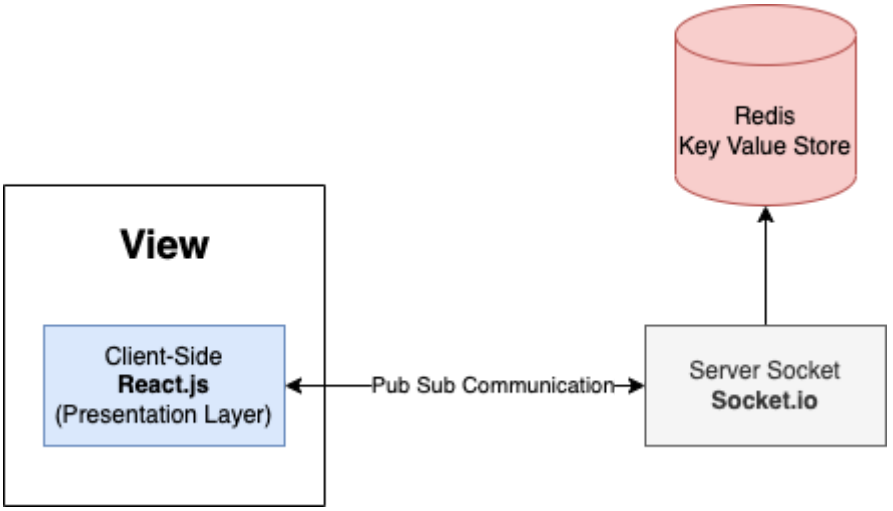
Database Schema

Match	
user	String
difficulty	String
room (unique)	String

9.3 Collaboration Service

Port: 8002

This microservice allows the matched users to code together in a shared code editor.



Collaboration Service Diagram

Sockets

Server

Subscribe	
Topics	**Data**
joinRoom	• roomId: String • user: String
codeChanged	• code: String

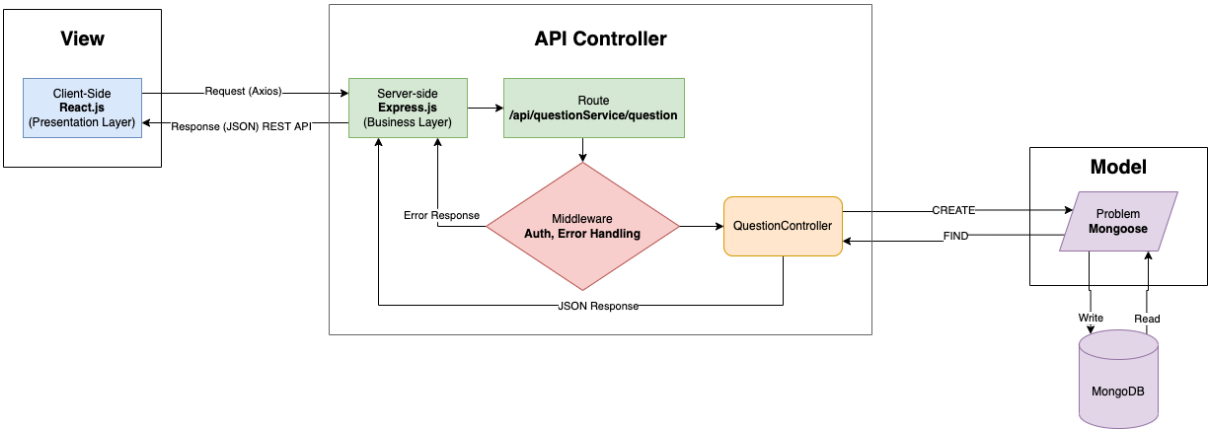
Publish		
Topics	**Target**	**Data**
usersInRoom	every socket in the room	• usersInRoom: Array
userDisconnect	every socket in the room	• user: String
codeUpdated	every socket in the room or every socket in the room excluding the sender	• code: String

error	sender	• message: String
userConnected	every socket in the room excluding the sender	• user: String
timesUp	every socket in the room	• message: String

9.4 Question Service

Port: 8003

This microservice stores the coding questions from leetcode.



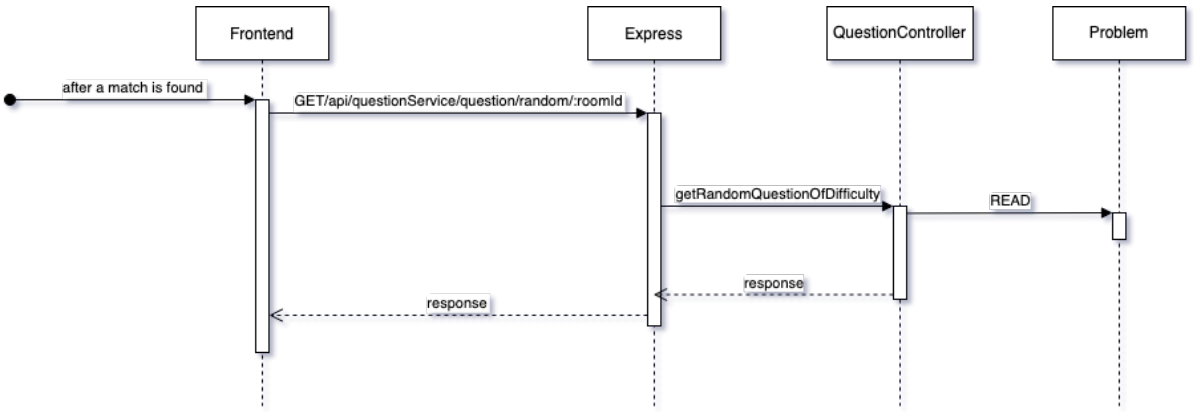
Question Service Diagram

APIs

Get a random question by difficulty	
Method	GET
Route	/api/questionService/question/random/:roomId
Params	• roomId: String
Response (JSON)	• _id: String • questionId: String • questionFrontendId: String • title: String • titleSlug: String • content: String • difficulty: String • likes: Number • dislikes: Number • similarQuestions: String • topicTags: Array • codeSnippets: Array • stats: String • hints: Array • sampleTestCase: String
Example	Params: match-Easy-a791ac0d-b89b-4a12-af37-a87d56abf

Response:

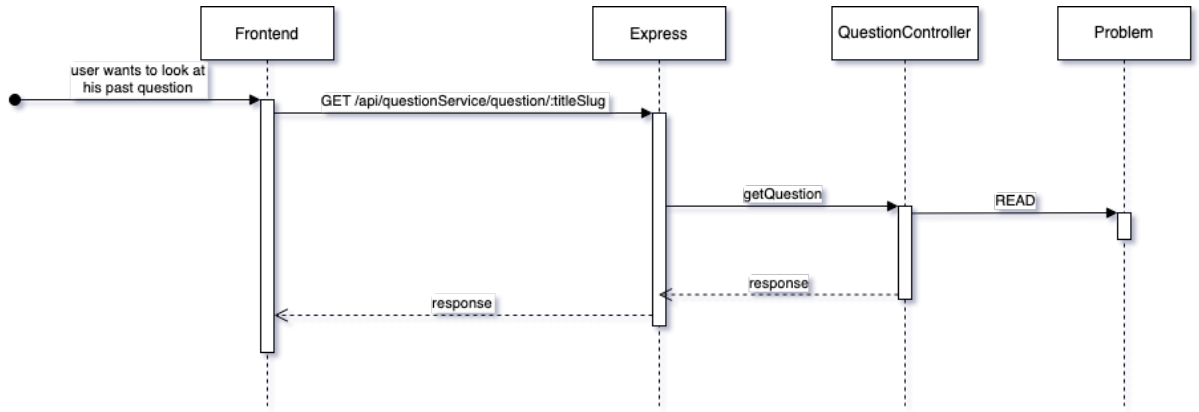

```
{
  "_id": "635223757920ab3888d9da09",
  "questionId": "14",
  "questionFrontendId": "14",
  "title": "Longest Common Prefix",
  "titleSlug": "longest-common-prefix",
  "content": "<p>Write a function to find the longest common prefix string amongst an array of strings.</p>\n\n<p>If there is no common prefix, return an empty string <code>&quot;&quot;</code>.</p>\n\n<p> </p>\n<p><strong class=\"example\">Example 1:</strong></p>\n\n<pre>\n<strong>Input:</strong> strs = [&quot;flower&quot;,&quot;flow&quot;,&quot;flight&quot;]\n<strong>Output:</strong> &quot;fl&quot;\n</pre>\n\n<p><strong class=\"example\">Example 2:</strong></p>\n\n<pre>\n<strong>Input:</strong> strs = [&quot;dog&quot;,&quot;racecar&quot;,&quot;car&quot;]\n<strong>Output:</strong> &quot;&quot;\n<strong>Explanation:</strong> There is no common prefix among the input strings.\n</pre>\n\n<p> </p>\n<p><strong>Constraints:</strong></p>\n\n<ul>\n\t<li><code>1 &lt;= strs.length &lt;= 200</code></li>\n\t<li><code>0 &lt;= strs[i].length &lt;= 200</code></li>\n\t<li><code>strs[i]</code> consists of only lowercase English letters.</li>\n</ul>\n",
  "difficulty": "Easy",
  "likes": 10866,
  "dislikes": 3464,
  "similarQuestions": "[]",
  "topicTags": [
    {
      "name": "String",
      "slug": "string",
      "translatedName": null
    }
  ],
  "codeSnippets": [...<truncated>],
  "stats": "{\"totalAccepted\": \"1.9M\", \"totalSubmission\": \"4.8M\", \"totalAcceptedRaw\": 1942133, \"totalSubmissionRaw\": 4770745, \"acRate\": \"40.7%\"}",
  "hints": [],
  "sampleTestCase": "[\"flower\",\"flow\",\"flight\"]",
  "__v": 0
}
```



Sequence Diagram for fetching a random question

Get a question	
Method	GET
Route	/api/questionService/question/:titleSlug
Params	• titleSlug: String
Response (JSON)	• _id: String • questionId: String • questionFrontendId: String • title: String • titleSlug: String • content: String • difficulty: String • likes: Number • dislikes: Number • similarQuestions: String • topicTags: Array • codeSnippets: Array • stats: String • hints: Array • sampleTestCase: String
Example	Params: longest-common-prefix Response: { "_id": "635223757920ab3888d9da09", "questionId": "14", "questionFrontendId": "14", "title": "Longest Common Prefix", "titleSlug": "longest-common-prefix",

```
  "content": "<p>Write a function to find the longest common prefix string amongst an array of strings.</p>\n\n<p>If there is no common prefix, return an empty string <code>&quot;&quot;</code>.</p>\n\n<p> </p>\n<p><strong class=\"example\">Example 1:</strong></p>\n\n<pre>\n<strong>Input:</strong> strs = [&quot;flower&quot;,&quot;flow&quot;,&quot;flight&quot;]\n<strong>Output:</strong> &quot;fl&quot;\n</pre>\n\n<p><strong class=\"example\">Example 2:</strong></p>\n\n<pre>\n<strong>Input:</strong> strs = [&quot;dog&quot;,&quot;racecar&quot;,&quot;car&quot;]\n<strong>Output:</strong> &quot;&quot;\n<strong>Explanation:</strong> There is no common prefix among the input strings.\n</pre>\n\n<p> </p>\n<p><strong>Constraints:</strong></p>\n\n<ul>\n\t<li><code>1 &lt;= strs.length &lt;= 200</code></li>\n\t<li><code>0 &lt;= strs[i].length &lt;= 200</code></li>\n\t<li><code>strs[i]</code> consists of only lowercase English letters.</li>\n</ul>\n",
  "difficulty": "Easy",
  "likes": 10866,
  "dislikes": 3464,
  "similarQuestions": "[]",
  "topicTags": [
    {
      "name": "String",
      "slug": "string",
      "translatedName": null
    }
  ],
  "codeSnippets": [...<truncated>],
  "stats": "{\"totalAccepted\": \"1.9M\", \"totalSubmission\": \"4.8M\", \"totalAcceptedRaw\": 1942133, \"totalSubmissionRaw\": 4770745, \"acRate\": \"40.7%\"}",
  "hints": [],
  "sampleTestCase": "[\"flower\",\"flow\",\"flight\"]",
  "__v": 0
}
```



Sequence Diagram for getting a specific question

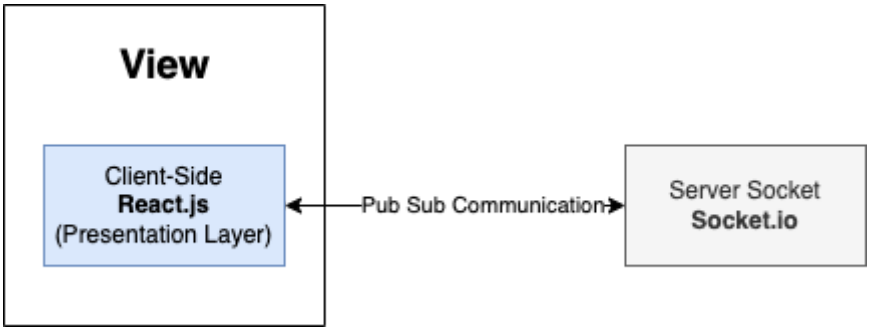
Database Schema

Problem	
questionId	String
questionFrontendId	String
title	String
titleSlug (unique)	String
content	String
difficulty (index)	String
likes	Number
dislikes	Number
similarQuestions	String
topicTags	Array
codeSnippets	Array
stats	String
hints	Array<String>
sampleTestCase	String

9.5 Communication Service

Port: 8004

This microservice allows the matched users to communicate with each other during the session.



Communication Service Diagram

Sockets

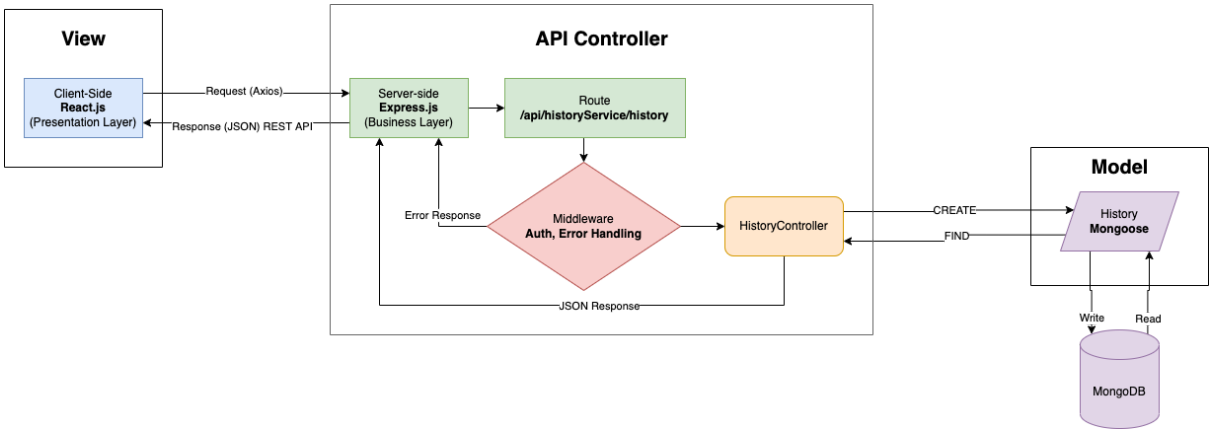
Subscribe	
Topics	**Data**
joinRoom	• roomId: String • username: String
chatMessage	• msg: String

Publish		
Topics	**Target**	**Data**
message	every socket in the room	• message:String
roomUsers	every socket in the room	• users: users • roomId: roomId
disconnect	every socket in the room	• socket: id

9.6 History Service

Port: 8005

This microservice stores and returns the past questions done by users.

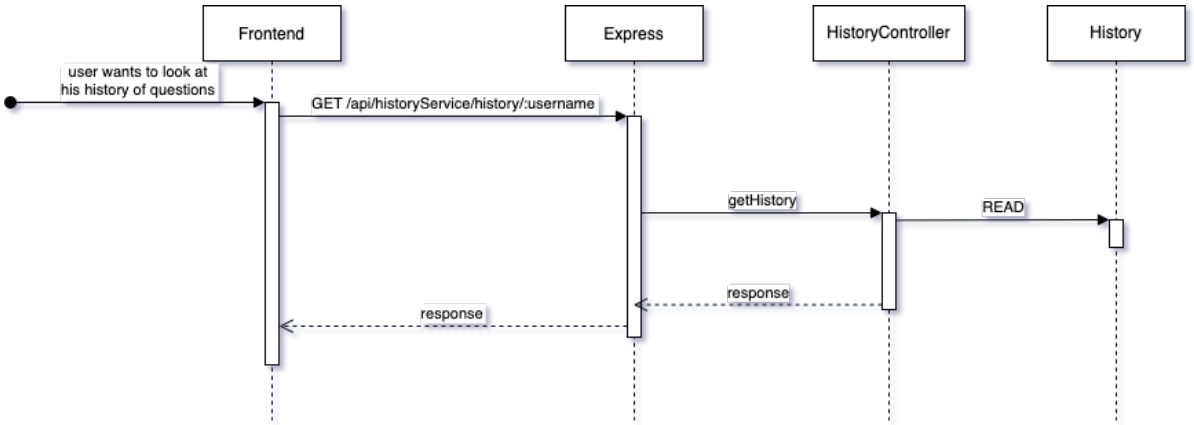


History Service Diagram

APIs

Get all history question for user	
Method	GET
Route	/api/historyService/history/:username
Params	• username: String
Response (JSON)	• _id: String • username: String • title: String • titleSlug: String • createdAt: String • updatedAt: String • __v: Number
Example	Params: tester1 Response: { "_id": "6368e9ca7c7ea530c5ea2e83", "username": "tester1", "title": "Zigzag Conversion", "titleSlug": "zigzag-conversion",

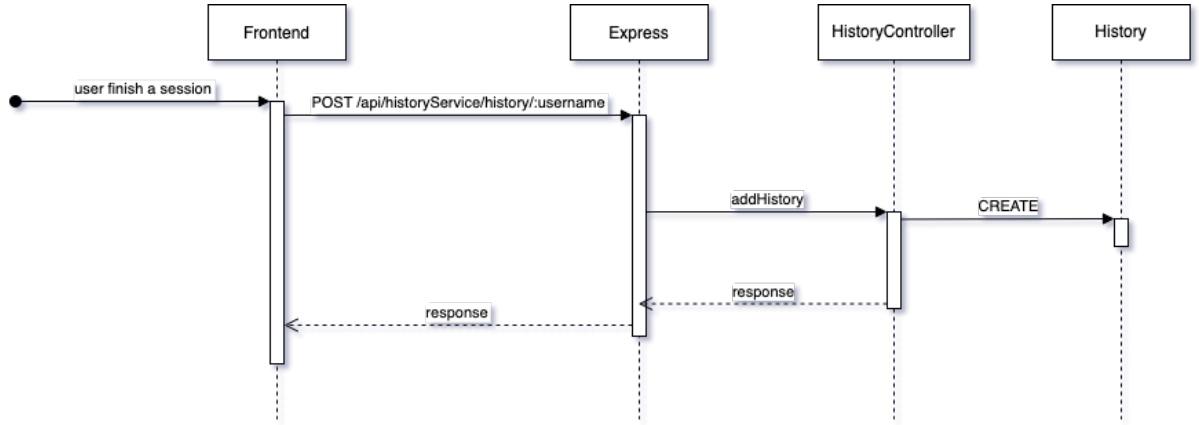
	"createdAt": "2022-11-07T11:19:38.381Z", "updatedAt": "2022-11-07T11:19:38.381Z", "__v": 0 }



Sequence Diagram for getting history for a user

Add new history for user	
Method	POST
Route	/api/historyService/history/:username
Params	• username: String
Request (JSON)	• title: String • titleSlug: String
Response (JSON)	• message: String • history: Object • history.username: String • history.title: String • history.titleSlug: String • history._id: String • history.createdAt: String • history.updatedAt: String • history.__v: Number
Example	Params: tester1 Body: {

	“title”: “titleTest”,
	“titleSlug”: “titleSlugTest”
	}
	
	Response:
	{
	"message": "History created successfully",
	"history": {
	"username": "tester1",
	"title": "titleTest",
	"titleSlug": "titleSlugTest",
	"_id": "6368ffde5a161cf24d10cb2b",
	"createdAt": "2022-11-07T12:53:50.604Z",
	"updatedAt": "2022-11-07T12:53:50.604Z",
	"__v": 0
	}
	}



Sequence Diagram for adding history after finishing a session

Database Schema

Problem	
username(required)	String
title (required)	String
titleSlug (unique)	String

10. Frontend

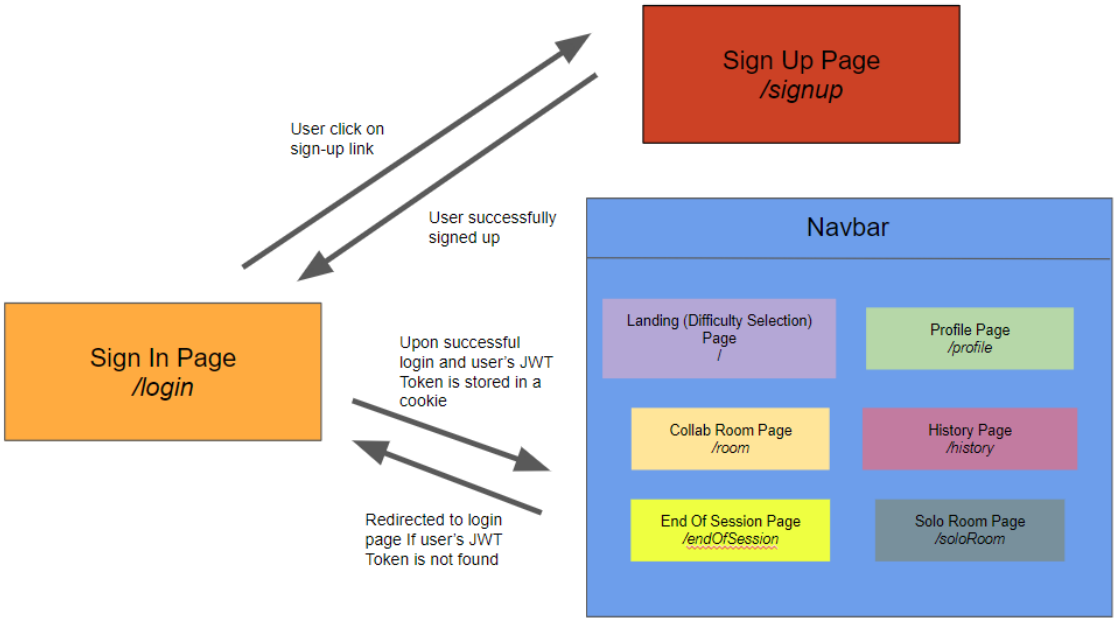
10.1 Introduction

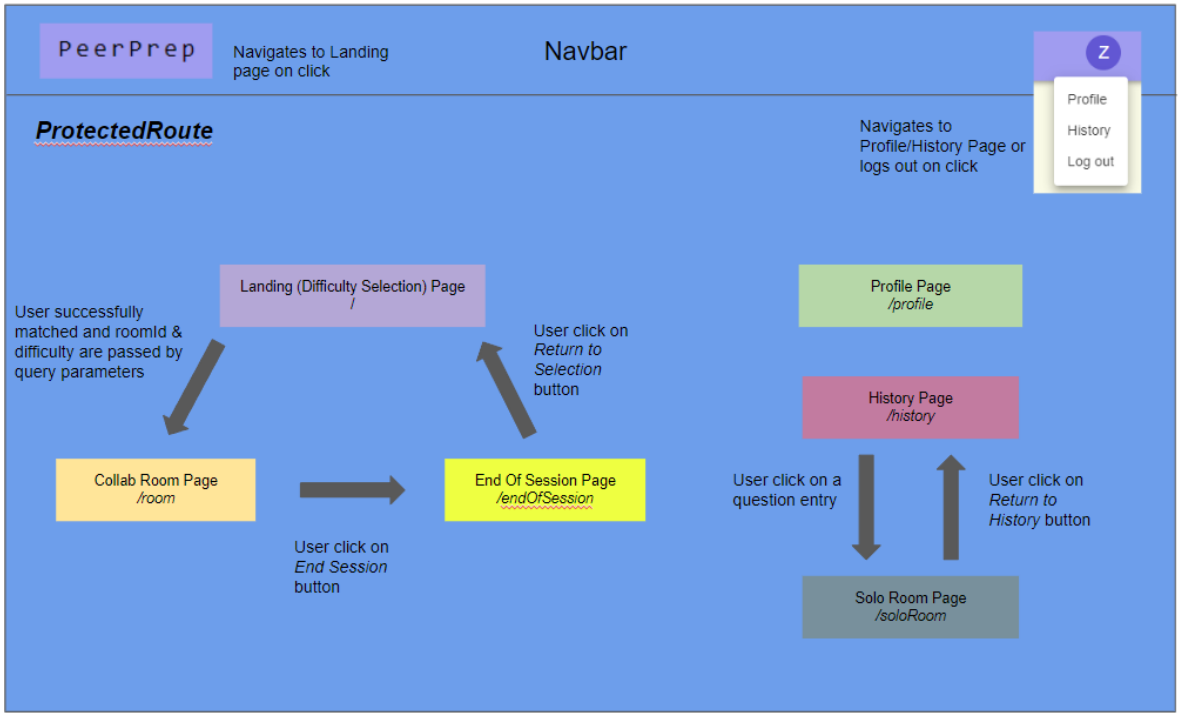
We used ReactJS framework to develop our web application. ReactJS is a free and open-source front-end Javascript library for building user interfaces based on UI components. It allows for:

- Easy creation of dynamic applications with the hooks that the framework provides
- Improved performance due to its usage of the Virtual DOM which allow for faster web applications creation
- Creating reusable components to reduce applications development time
- Dedicated tools for debugging and console logging available, such as Chrome extensions

The main component library that we use in development is Material UI. Material UI is an open-source component library for React components. It is built using Less, Leaner Style Sheets, a backward-compatible language extension for CSS. Material UI provides a collection of component types that are ready to use in development. The beauty of using Material UI comes in the customizability of each type, allowing developers to utilise components with the styles that they desired. The library also comes with a comprehensive documentation, enabling us developers to employ their components with ease and to their own custom needs. Hence, through using Material UI as the component library, we were able to reduce time and effort put into styling without compromising much on the looks of the components used.

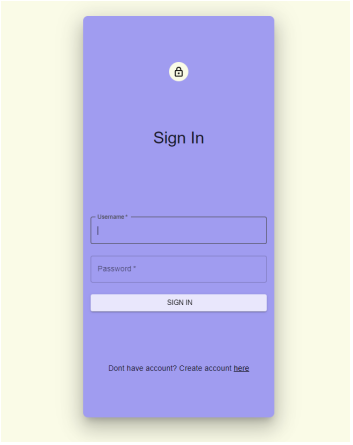
10.2 User Flow

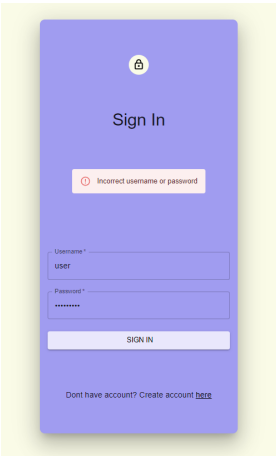




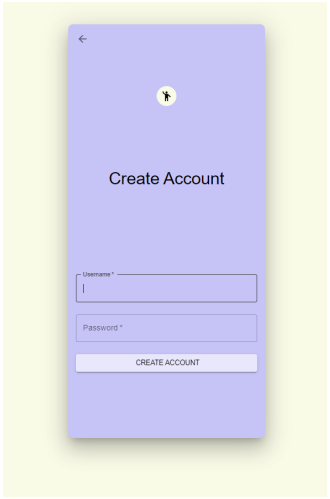
10.3 Application Showcase

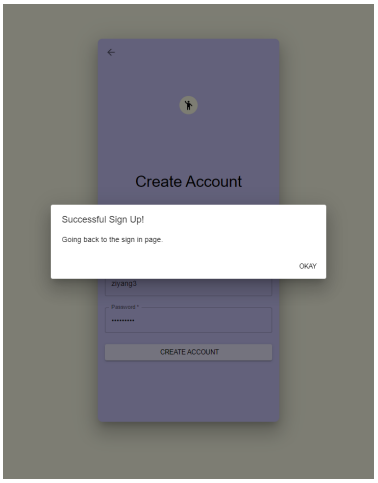
10.3.1 Signing Up & Login



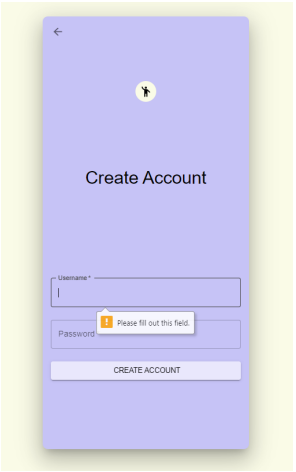


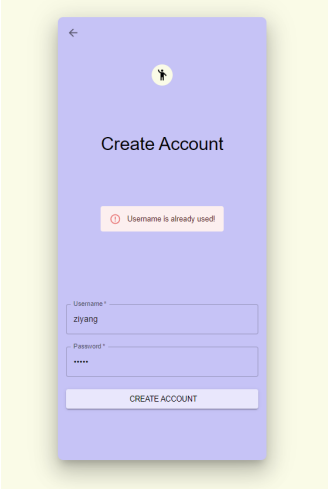
Login Page





Sign Up Page



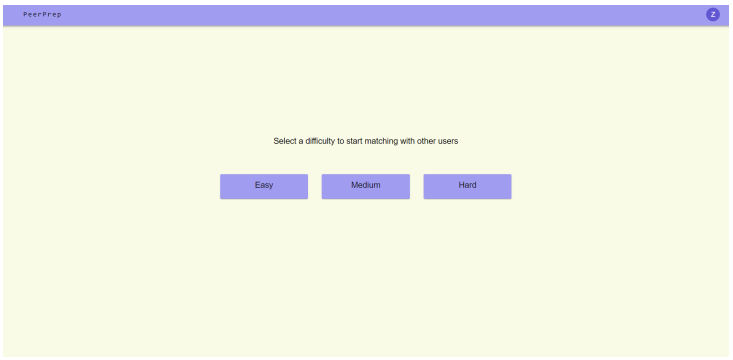


Sign Up Validations

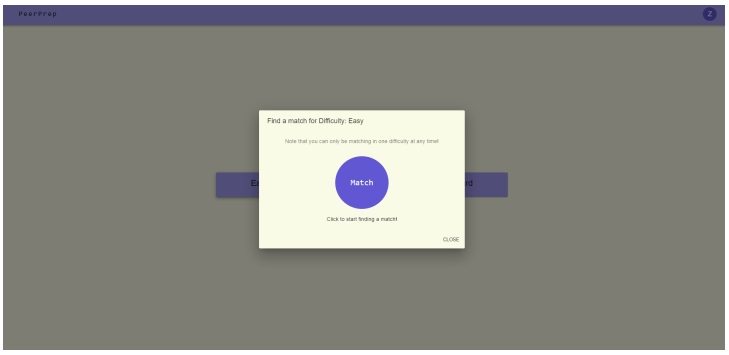
The sign up and sign in page simply requires a username and a password. For the sign up process, validations are included to check if the username is unique or if any fields are empty. Once the sign up is successful, the user will be guided to the sign in page to login with their newly created account.

Authentication is done by verifying the JWT Token created by user-service. We utilise a cookie to store the JWT Token and verify requests made to the microservices. We set the token to expire in 3 days.

10.3.2 Difficulty Selection and Matching

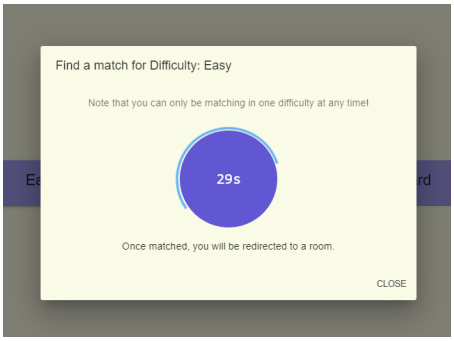


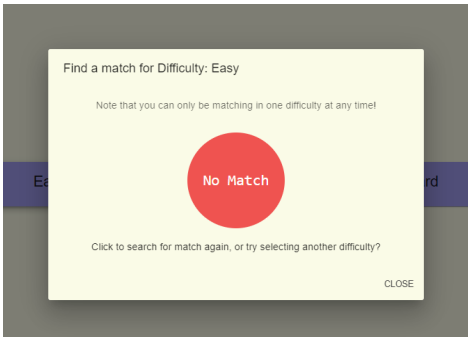
Difficulty Selection Page (Landing Page)

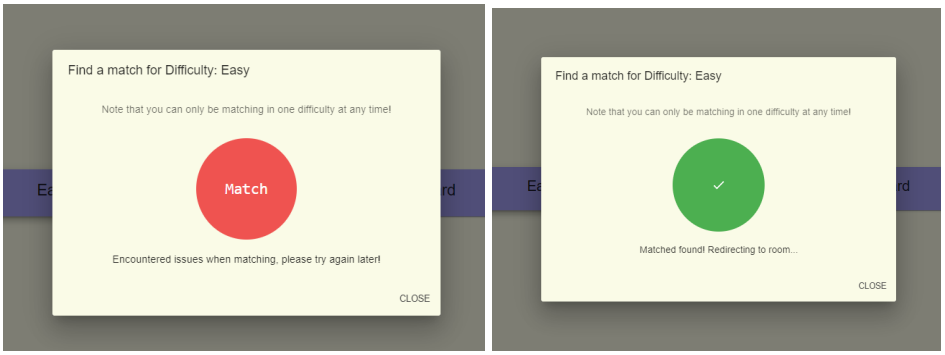


Difficulty Selection Page with Select Difficulty Popup

Once the user is logged in, the user will be brought to the difficulty selection page. On this page, the user can click on a difficulty which would present a pop up showing a match button.





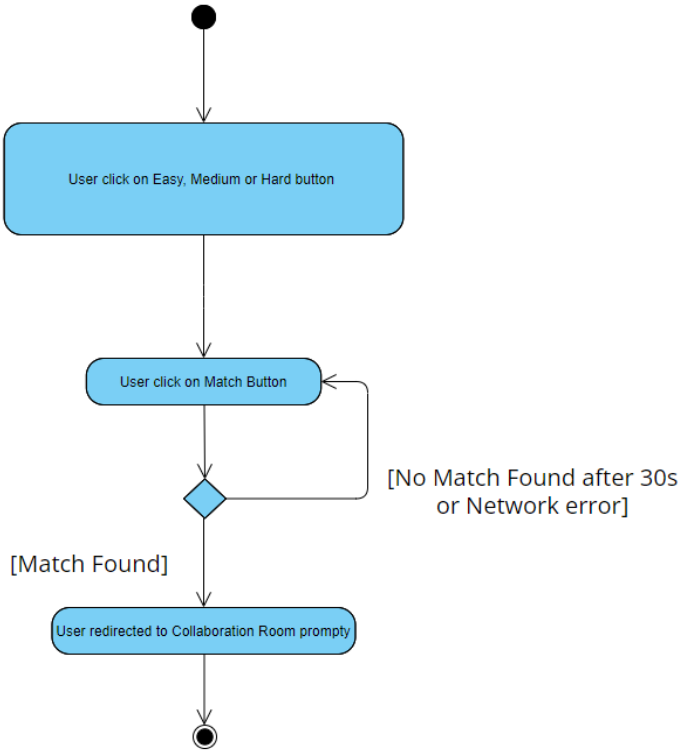


Matching Pop up with Different Statuses

The matching process begins and goes on for 30 seconds once the user clicks on the button. If there is no match after 30 seconds or if there is a network error, the button will turn red and the respective erroneous message will appear. From here, the user may choose to click on the match button to begin the matching process again.

If matched successfully, the user will see a green tick and will be brought to the collaboration room promptly.

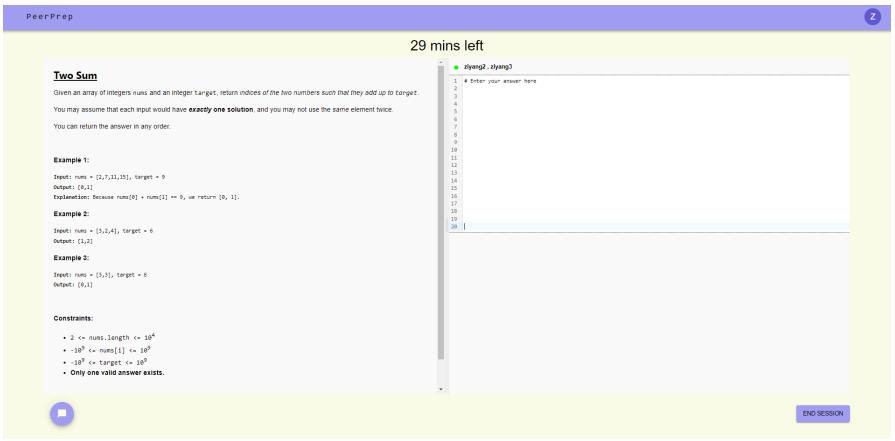
For a quick understanding of the flow for matching, please refer to the activity diagram below.



Activity Diagram for Matching

The matching system is implemented via communication with web sockets supported by matching-service. Once two users are matched, the matching-service would return a unique roomId to the frontend application which will be used to generate the collaboration room and question.

10.3.3 Collaboration



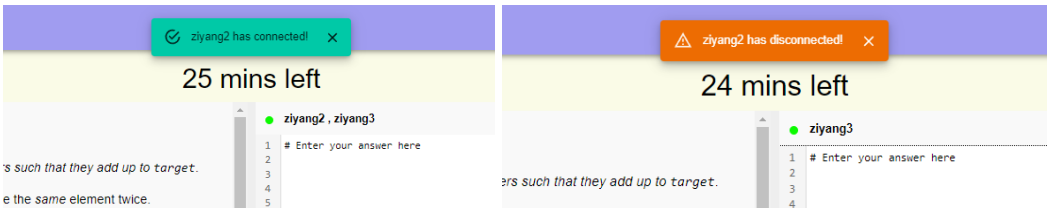
Collaboration Room Page

After two users are matched, they are promptly brought into the collaboration room to begin the process of the mock interview. The user is provided a real-time collaborative editor on the right panel and the question on the left panel. The panels are resizable by the interactive column in the centre.

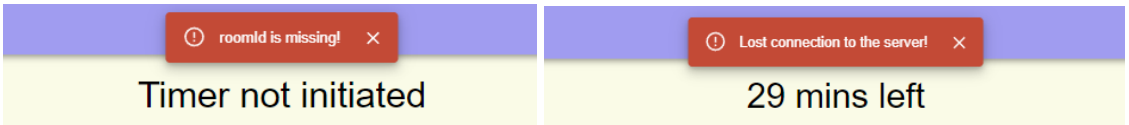
The user display right above the editor shows the users currently in the room. If a user disconnects, the display will update accordingly.

A timer that counts down from 30 minutes is presented at the top of the collaborative room. The timer mimics the process of an interview which typically lasts around 30 minutes. The timer displays the time in minutes if there is still more than 1 minute left, while it displays the time in seconds if there is less than a minute left. This is to allow the user to be fully aware of how much time there is left.

Regardless of whether the timer finishes counting or not, the user is able to continue work in the collaborative editor. Once the user clicks on the End Session button, the question will be recorded in the user's history and the user will be redirected to the End of Session Page.



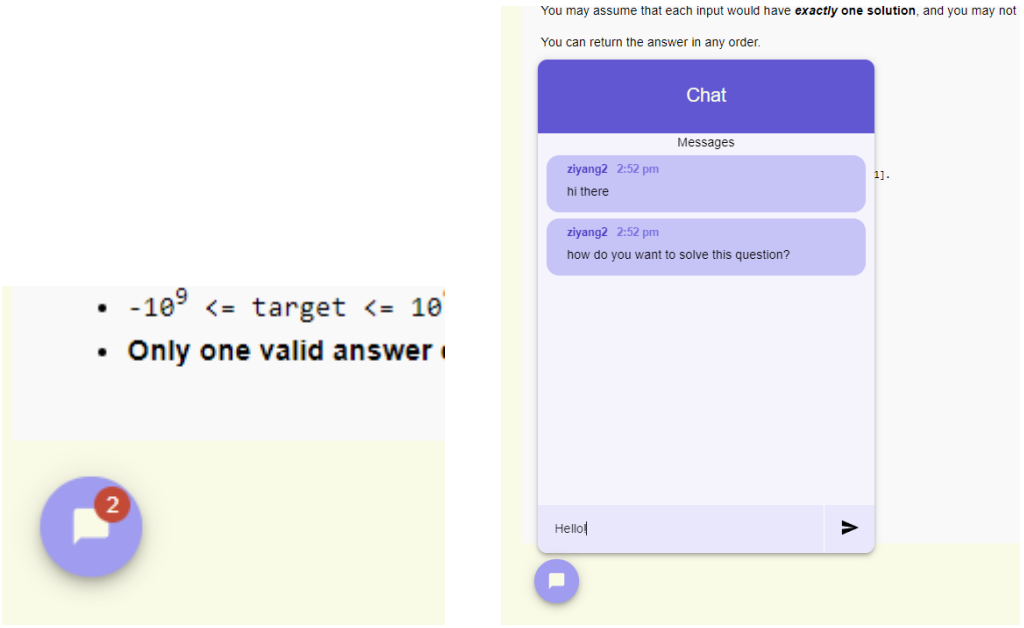
Notifications for reconnecting/disconnecting of partner



Notifications for errors

Note that the collaborative room is flexible in allowing the user to disconnect and reconnect to the same room. We decide to allow this behaviour to account for the event in which the

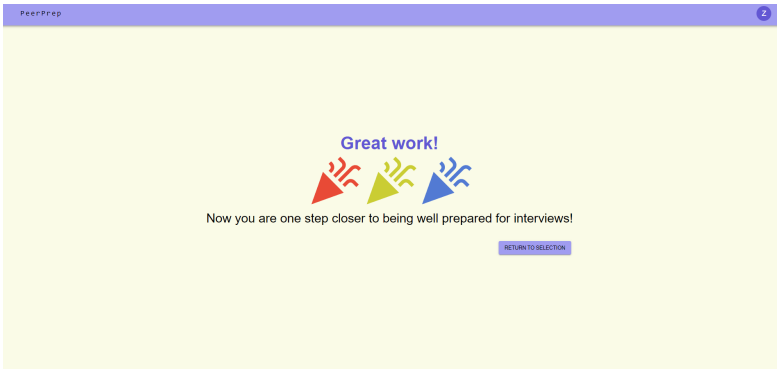
user may be disconnected forcefully due to network issues. We implemented the above notifications to show if a partner or one's self has disconnected, reconnected or has issues with the room.



Chat Function

A real-time messaging tool is also available in the Collaborative Room Page. Users in the same room are able to communicate with one another in this chat to accommodate better collaboration for the question.

A user may click on the chat button to display the chat feature. For any unread messages received while the chat feature is not displayed, the number of unread messages will be shown in a small icon in the chat button as shown above.

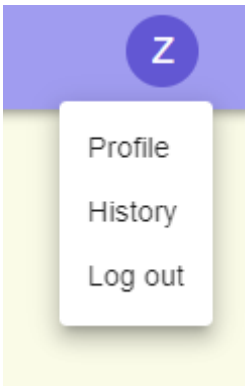


End of Session Page

At the End of Session Page, the user will be shown a short celebratory animation. The user may return to the difficulty selection page from there via the button.

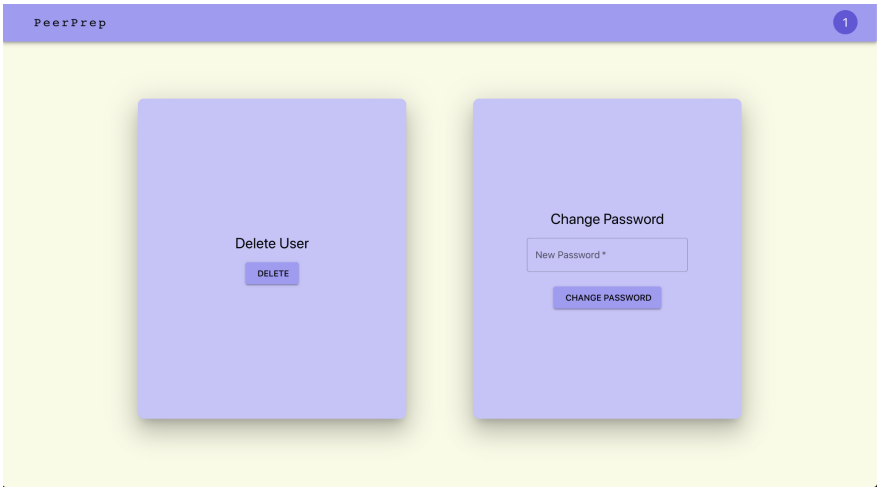
Similarly, the real-time collaborative editor and messaging chat feature were implemented with web sockets supported by collaboration-service and communication-service respectively.

10.3.4 Pages under Avatar Menu Items

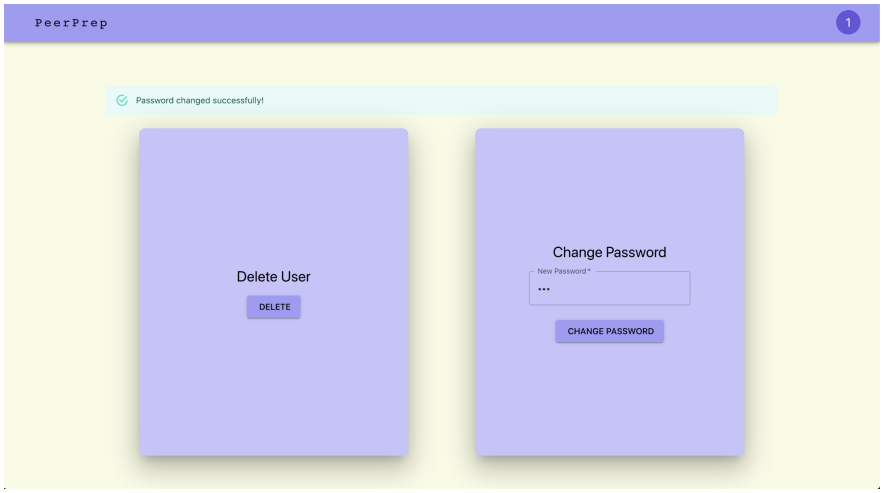


Dropdown Menu Items from Avatar

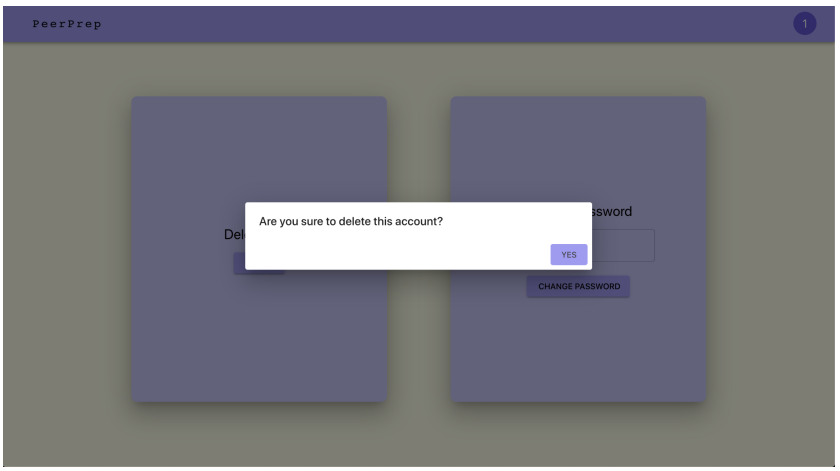
The avatar with the dropdown menu items is available on all authenticated pages. Hence, the Profile and the History Page are accessible from any authenticated pages.



Profile Page

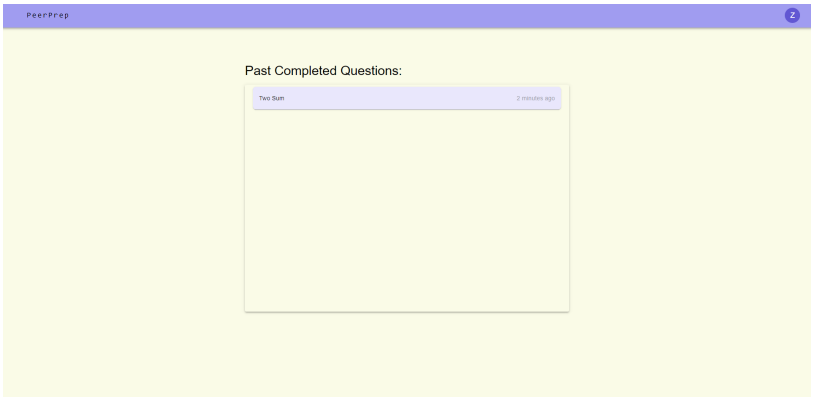


Password changed notification

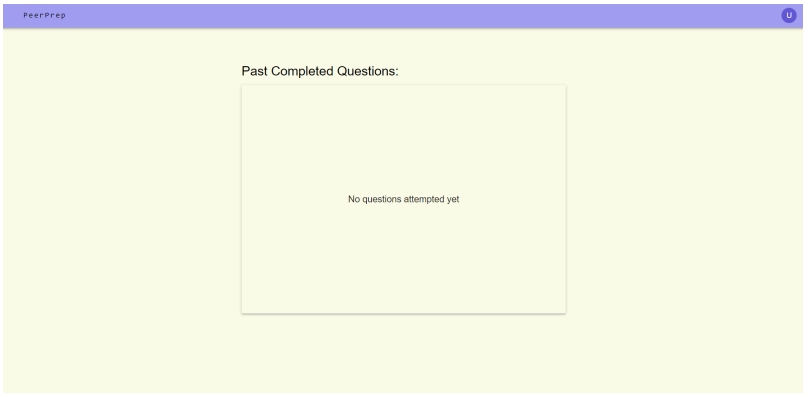


Account deletion confirmation

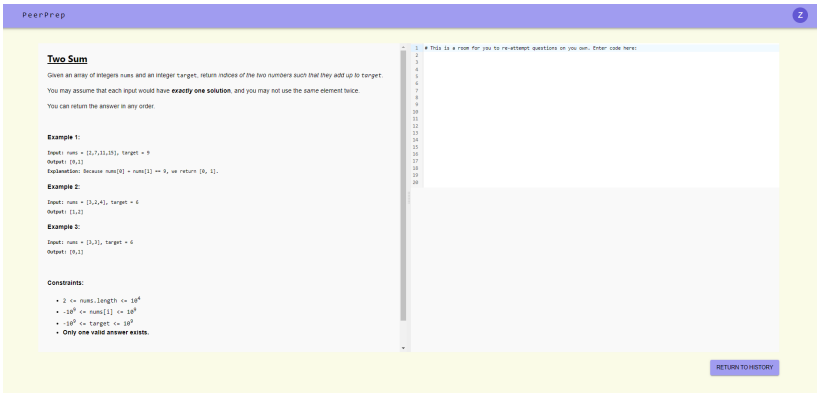
A simple UI to allow users to change their current password or delete their account. Any change in password or deletion of accounts will prompt a notification upon success or failure. For deletions, there will be a confirmation popup prior to ensure the user wants the account to be deleted. Once the user deleted their account, the user will be redirected back to the login page.



History Page



History Page with no question attempted



Solo Room Page

In the History page, users are able to see their past completed questions. If there are no questions completed yet, the empty table shows a message as shown in the screenshot.

If the user wishes to reattempt the question on their own, they can click on the respective question and they will be redirected to the Solo Room Page to reattempt the question individually. In the Solo Room, there is no real time collaboration on the editor. The user may click on the Return to History button to redirect back to the History Page.

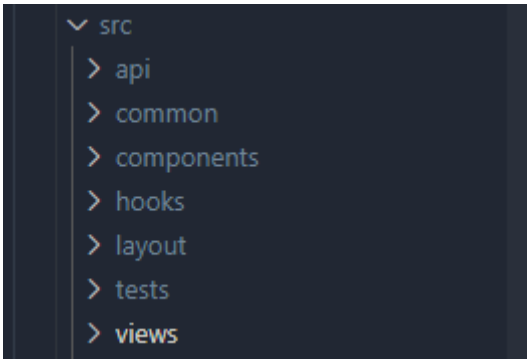
11. Design Considerations

11.1 Frontend

Project File Structure

The files for the frontend application are structured in a way to allow for better accessibility and scalability. Files are separated into these respective folders according to their purpose:

- api - logic that sends request to backend services
- common - common logic, configurations or utility that is used in multiple types of files
- components - reusable features of the application (Chat feature etc)
- hooks - logic to handle state managements and lifecycle processes
- layout - overall view structure of the app
- tests - unit tests
- views - pages with different routes

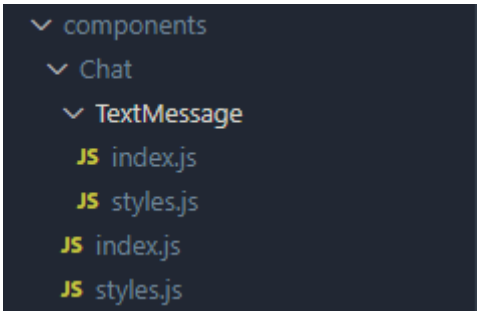


File structure of frontend project

Component File Structure

In general, components contain an index.js file that contains HTML and Javascript logic. If complex styling is required, a styles.js file is created to store Javascript objects containing styles in accordance to Material UI's inbuilt styling feature.

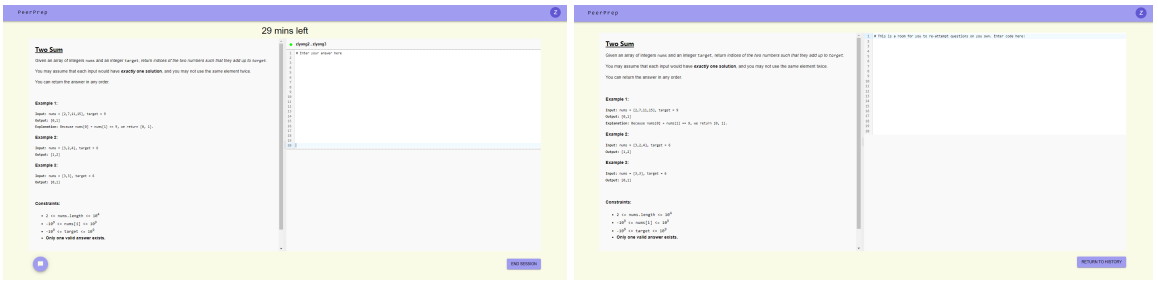
If the component has additional smaller components with complex logic, a nested component folder is created within that component. As shown in the example below, the TextMessage component folder is created within the Chat component folder.



File structure of a component

Reusable Components and CSS

To ensure that we do not write repeated code, we reuse the components created if applicable. This is evident in our Collaboration Room Page and Solo Room Page in which components like the editor, question pane and end button is reused.



Showcase of reusing components

Testing

We utilised React Testing Library to unit test specific functions in our frontend application.

For example, we tested if the correct text is rendered upon clicking on a difficulty button in the Difficulty Selection Page, or if the create account button is rendered properly in the Sign-Up Page.

```
it("renders text in cards correctly", async () => {
  render(SelectionContentRoute);
  expect(screen.getByText("Easy")).toBeInTheDocument();
  expect(screen.getByText("Medium")).toBeInTheDocument();
  expect(screen.getByText("Hard")).toBeInTheDocument();
});

test("renders easy dialog correctly", async () => {
  render(SelectionContentRoute);
  const difficulty = "Easy";
  await userEvent.click(screen.getByText(difficulty));

  expect(
    screen.getByText(`Find a match for Difficulty: ${difficulty}`),
  ).toBeInTheDocument();
  expect(screen.getByText("Match")).toBeInTheDocument();
});
```

```
test("renders create account button correctly", () => {
  render(
    <Router location={"/"}>
      <Routes>
        <Route path="/" element={<SignUpPage />} />
      </Routes>
    </Router>,
  );

  expect(
    screen.getByRole("button", {
      name: /Create Account/i,
    }),
  ).toBeInTheDocument();
});
```

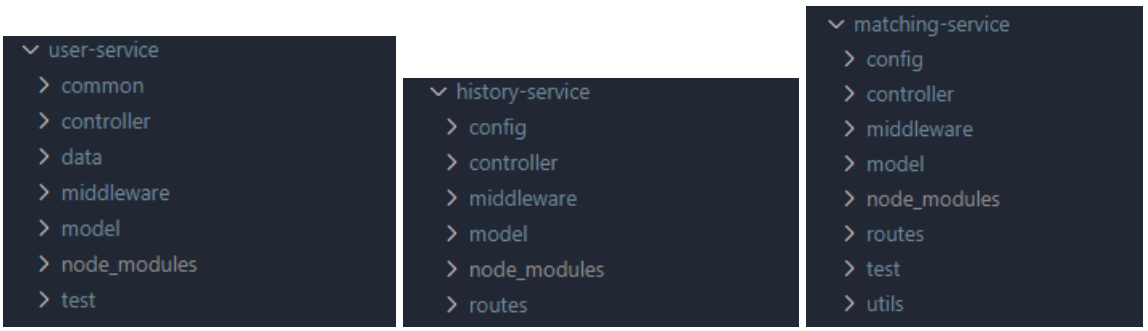
Example of unit test cases in the frontend

11.2 Backend

Project File Structure

Similarly, the files for each backend service are structured in a way to maintain consistency for better accessibility and scalability. In general, we follow this structure of folders that are each responsible for specific files:

- common/config - common logic or configurations that is used in multiple types of files
- controller - API business logic for models
- middleware - external utilities such as error handling
- model - mongoDB Schema structures
- routes - routers that routes controller logic to API endpoints
- test - unit tests



File structure of backend services

Standardised Database Technology for All Services

For our database in the backend, we decided to use the same MongoDB Atlas cluster for all our services. Each service will have their own individual database in the cluster with consideration of the performance and scalability of each database.

Initially, User Service was using MongoDB, while Match Service was using SQLite. After discussing, we found that if we use a different database for each service, we would have to have a different deployment setup for each database and that would be very troublesome. Also, scaling them would be non-trivial as well because we might need different ways to scale different databases. Hence, we decided to use MongoDB Atlas as our cloud database to provide consistency in our database deployments.

12. Remarks

12.1 Suggestions for Improvements

Frontend

Sign Up Experience can be improved

We acknowledge that the user may have a better sign up experience if they were authenticated and redirected to the landing page right after signing up. However, we settled for the current implementation since it does not obstruct the user flow. Instead, we prioritised other features such that the communication tool and History/Solo Room Page. Given more time and resources, we would be able to update the sign up experience.

User Service

Black Listing of tokens

To improve our security, we can blacklist previously authenticated tokens so that hackers cannot reuse the same token even though it was used before for authentication. It can be used to invalidate JWT tokens when logging out. However, since we prioritise other features more as explained in the Non-Functional Requirement section, we spent our resources working on other FRs and NFRs.

Collaboration Service

Better concurrent code editing experience

Currently, users aren't able to both edit the code at the same time. This problem occurs because we don't have a way to handle conflicting edits as of now. We only found out about this when we were testing in production after our deployment was set up. With more time and resources, we could use either operational transformation (OT) from CodeMirror or conflict-free replicated data types (CRDT) from Yjs as an approach to handle the conflicting edits.

12.2 Enhancements of the Applications

Reloading of Past Submissions on the Collaborative Code Editor

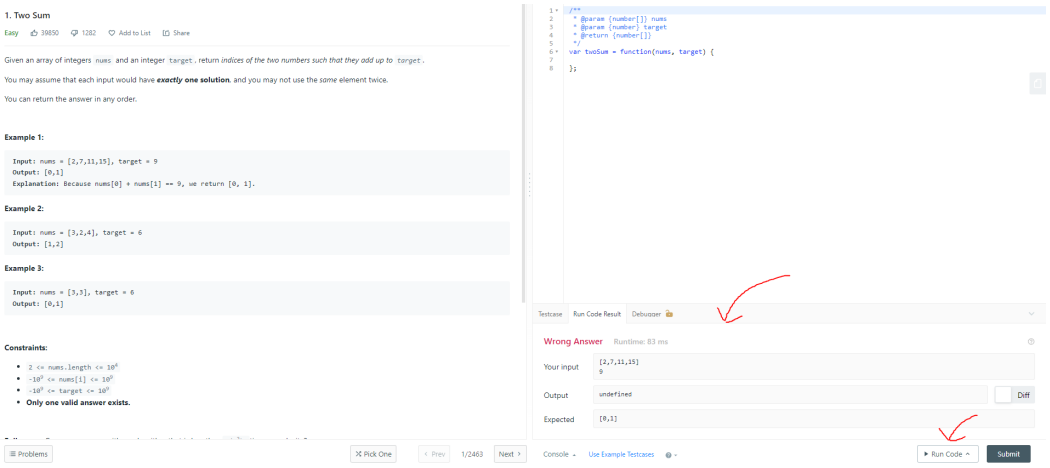
One useful feature we can look into implementing would be to reload past submission in Collaborative Code Editor. Users will be able to revisit the code that they submitted in the Collaborative Room back in the Solo Room Page.

This feature would require us to implement an extension in the history-service to store code bodies in past attempts together with the question details. When a user clicks on the End Session button in the Collaborative Room, the current code body will be stored in the history-service. From there, the frontend application can pull the code body into the Solo Room editor when the user navigates there from the History page.

Providing Test Cases and compilation of code on the Editor

Another useful feature we can implement would be to allow the compilation of code on the respective editors of the Collaborative and Solo Room Page. Currently the editors simply serve as a text editor. However, by allowing compilation and test cases which the editor can run the code through, the application would be able to better simulate an interview or an online assessment process in the job application.

Since the editor we are using is CodeMirror, extending the editor to accommodate this feature would be convenient. CodeMirror editors have a built-in compilation system to compile specific languages. We simply need to update the configurations of the editor in the frontend application and utilise packages to allow compilation of specific languages. On the other hand, the question-service will need to store test cases and provide the test cases via an API to the frontend application. New interfaces and buttons will need to be implemented in the Collaborative Room to accommodate running of test cases.



LeetCode's Interface for running testcases

We would model the interface with the Leetcode interface to run test cases. We can start by providing fixed test cases, then extend the feature towards custom test cases.

Audio & Video Communication

Lastly, a feature that we can implement would be to provide audio and video communication. By providing these functions, we can truly simulate an environment of an interview where one has to converse with the interviewer.

To implement this feature, we can extend the current communication service to include audio and video communication functions. For the frontend application, we can explore using React WebRTC libraries to allow for real time audio and video communication.

12.3 Reflections

Looking back, the project proved to be challenging and time-consuming for each of us. There was a high learning curve in the beginning of the project as each of us needed to research and study on the various technical resources involved. While we have the advantage of more manpower in a team, allocation of resources, management and communication in a team of 5 was not smooth-sailing especially since our groups were merged after each group had already started on the project. Nonetheless, we were able to put what we learned from lectures and online resources into practice. Each of us definitely grew in terms of technical expertise in the domains that we worked on (Frontend, Backend, DevOps, etc.) Finally, we were able to produce a satisfactory product that we are proud to showcase.

12.4 Learning Points

Overall, the project was a valuable and fulfilling experience. Here are some learning points that we felt we did or did not do well during the project phase.

What we did well:

- Clear communication in the team through the Telegram group. If there are any bugs or areas of improvement, we would share within the group. If meetings are required to align expectations, we were swift in organising one by communicating on the Telegram group.
- Willingness to share experience and learn from others. Some of us have some technical expertise that we gained outside of the module (from self-learning or internships) Members in the team were helpful and willing to share practices and resources that are useful in the project. We also shared resources within the team during the development phase of the project so that everyone can learn as well.
- Flexibility to work in any technical domain. Our arrangement of assigning manpower was flexible such that each of us was able to work on multiple technical domains (Frontend, Backend, DevOps). Hence, we were able to gain more technical experience during the development phase.

What we did not do as well:

- Time management. While we were able to produce a working product with several features, we believe that better management and planning can help us achieve much more in the application. Given more time, we could have worked on improvements or enhancements of the application.

13. References

- https://en.wikipedia.org/wiki/React_(JavaScript_library)
- https://mui.com/
- https://leetcode.com/
- https://github.com/yjs/yjs
- https://codemirror.net/examples/collab/
- https://www.diegocalvo.es/en/methodology-scrum-methodology-agil/
- CS3219 AY2022/23 Semester 1 Lecture 5 (MVC)
- CS3219 AY2022/23 Semester 1 Lecture 6 (Microservices)
- https://www.toptal.com/designers/gui/principles-of-design-infographic