CS3219 AY 2022/23 G59

Final Report

Royce Ho Shou Yee A0199477A
Ong Wei Sheng A0206042X
Hiong Kai Han A0199814M
Herrick Koh Yu Kan A0199870J

Contributions

Name	Technical Contributions	Non-Technical Contributions

Herrick Koh Yu Kan	• Backend
	  ○ Question Microservice
	  ○ Collaboration Microservice
	• Frontend
	  ○ Code Editor component
		• Requirements documentation
		• Project Report

Royce Ho Shou Yee	• Backend
	  ○ Matching Microservice
	  ○ Question Microservice
	  ○ DevOps
		• Requirements documentation
		• Project Report

Hiong Kai Han	• Backend
	  ○ User microservice
	  ○ Chat microservice
	• Frontend
	  ○ Login Page
	  ○ Signup Page
	  ○ Settings Page
	  ○ NavBar component
	  ○ Chat component
		• Requirements documentation
		• Project Report

Ong Wei Sheng	• Backend
	  ○ History Microservice
	• Frontend
	  ○ Home Page
	  ○ Search Page
	  ○ Interview Page
	  ○ History Page
		• Requirements documentation
		• Project Report

Introduction

Technical interviews have been a problem that many students struggle to overcome during job applications. Reasons for this could range from being unable to solve the questions they were presented with or the inability to formulate their thoughts into words to show their understanding.

Thus, to address this issue, we have created a web app, PeerPrep, where users can collaborate in real-time to solve coding interview questions. PeerPrep will allow users to work in pairs and attempt a question of their selected difficulty. Furthermore, they will be able to work in the same codespace and interact through a chatbox in real-time, allowing discussion and sharing of ideas which could further help improve a student's learning experience via collaborative learning.

Functional Requirements

User-Service

S/N	Functional Requirements	Priority
US-1	The user shall be able to create an account with username and password	High
US-2	The system should ensure that every account created has a unique username	High
US-3	The user shall be able to login to his/her created account using his/her username and password	High
US-4	The user shall be redirected to the home page upon successful login	High
US-5	The user shall be able to logout from his/her account from the home page	High
US-6	The user shall be asked to input correct login details or create an account upon unsuccessful login	High
US-7	The system should allow users to delete their account	Medium
US-8	The system should allow users to change their password	Medium

Matching-Service

S/N	Functional Requirements	Priority
MS-1	The user shall be able to select the difficulty of question to attempt in the home page	High
MS-2	The user shall be able to match with another available user in 30 seconds that selected the same difficulty of question to attempt	High
MS-3	The user shall be informed of an unsuccessful match if another available user that selected the same difficulty of question to attempt cannot be found in 30 seconds	High
MS-4	System will stop finding a match if no match was found for the user after a 30 seconds window.	High
MS-5	The user shall be redirected to the home page upon an unsuccessful match.	High
MS-6	The user shall be redirected to the interview page upon a successful match together with the other matched user	High
MS-7	The user shall be redirected back to the home page after clicking the end interview button in the interview page	High
MS-8	The user shall be able to terminate the matching with another user while the matching process is ongoing	Medium

Question-Service

S/N	Functional Requirements	Priority
QS-1	The user shall be provided with a random question of the difficulty he/she chose in the interview page	High
QS-2	Questions should include a title, a description, and a link as its body.	Medium
QS-3	Questions should be saved and rendered using a markdown style.	Low

Collaboration-Service

S/N	Functional Requirements	Priority
COS-1	The user shall be provided with a text field to type his/her solution to the question.	High
COS-2	The user should be able to type their input at the same time as the other user they are matched with.	High
COS-3	The text field shall be updated in real-time to update the inputs of both users that are matched.	High

History-Service

S/N	Functional Requirements	Priority
HS-1	The user shall be able to track his/her previous questions attempted question difficulty and the user matched with that question from the history page.	High
HS-2	The user shall be able to click the leetcode link to his/her previous questions attempted from the history page.	Medium

Chat-Service

S/N	Functional Requirements	Priority
CHS-1	The user shall be able to communicate with the matched user through the chat on the interview page.	High
CHS-2	The user should be notified when their corresponding user has disconnected.	Medium

Non-Functional Requirements

General

S/N	Non-Functional Requirements	Implementation	Priority	Quality Attribute
G-1	The webpage should not take more than 5 seconds to load.	-	Medium	Performance
G-2	The system should be able to run 24/7.	-	Medium	Availability
G-3	The font size of words on the page should be easy to read (size 11 or bigger)	-	Low	Usability

User-Service

S/N	Non-Functional Requirements	Implementation	Priority	Quality Attribute
US-1	The user's password should be protected.	Passwords are hashed using bcrypt’s hash function.	High	Security
US-2	The user database should be protected against unauthorized deletion or editing of user info.	JWT authorization token is given to users and required for change of password and account deletion	High	Security
US-3	The database should be able to contain 1000 unique users on a normal basis.	MongoDB	Medium	Integrity

Matching-Service

S/N	Non-Functional Requirements	Implementation	Priority	Quality Attribute
MS-1	The user should be able to reconnect to the interview after a connection is lost.	-	Medium	Robustness
MS-2	The user should be able to view how long they have	-	Low	Usability

	been in the matching queue.			

Question-Service

S/N	Non-Functional Requirements	Implementation	Priority	Quality Attribute
QS-1	The question should be presented to the user in a consistent format. E.g., Title, question, examples then limitations.	-	Low	Usability

Collaboration-Service

S/N	Non-Functional Requirements	Implementation	Priority	Quality Attribute
COS-1	The code update between users should not be longer than half a second under normal circumstances.	Socket-io allows users to connect to each other using rooms.	High	Performance
COS-2	The user's progress should be loaded after reconnecting from a sudden disconnect.	Redis to cache data.	Medium	Integrity
COS-3	The webpage should cache the user's work for the duration of 30 mins in case the user disconnects suddenly and wishes to reconnect and continue where they left off.	Redis to cache data.	Medium	Robustness

Chat-Service

S/N	Non-Functional Requirements	Implementation	Priority	Quality Attribute
CHS-1	The chat delay between users should not be longer than half a second under normal circumstances.	Socket-io allows users to connect to each other using rooms.	High	Performance

Possible improvements

S/N	Non-Functional Requirements	Service	Quality Attribute
FW-1	The webpage should be able to handle 100 users at the same time without crashing.	Deployment	Reliability
FW-2	The website should be able to be viewed on mobile devices with a dedicated UI to make it easier.	Frontend	Portability
FW-3	The user should be notified of possible slower response time when there are 80 users online.	General	Efficiency
FW-4	The system should be able to handle 100 concurrent users and must be able to increase it to 250 concurrent users in 8 hours.	General	Scalability
FW-5	The user will need to link an email for verification during account creation.	User	Security
FW-6	The user should not be able to design a weak password and will be forced to include numbers and certain special symbols	User	Security
FW-7	The system should be backed up daily and verify the current database with the backup copy.	User	Integrity
FW-8	The user should be able to check how many other users are currently accessing the site.	Matching	Usability
FW-9	The user should be able to add questions attempted before from other sites as a list to their history to keep track of questions done outside the site.	History	Compatibility
FW-10	The user should be able to select the language type they prefer between Python, Java, Javascript and C++.	Collaboration	Usability

Schedule / Plan

Software Development LifeCycle

We used an agile approach to our software development lifecycle. Atlassian describes agile as

> *An iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches. Instead of betting everything on a "big bang" launch, an agile team delivers work in small, but consumable, increments. Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for responding to change quickly.*

This involved creating a basic version of our product with limited functionality first, followed by iterations of adding features to the product. We conducted a weekly meeting to

1. Sync up on updates
2. Discuss any challenges to address
3. Plan new tasks/features

The next section describes the individual milestones in more detail.

Milestones

Completed	In progress	Planned / Unconfirmed

Milestone 1 (Weeks 4-6)

Feature	Expected outcome	Assignee	Status
User service	Complete user service logic.	Kai Han	
Question service	PR was created for review. An endpoint should return a random question of the given difficulty when called by the matching service.	Herrick	
Matching service	Complete 1st iteration of backend logic. Users should be able to match with each other and edge cases should be handled.	Royce	
Frontend (Login and Home)	Complete working version of frontend, integrated with user service.	Kai Han	
Frontend (Match)	Complete working version of frontend, integrated with matching service.	Wei Sheng	

Milestone 2 (Weeks 7-8)

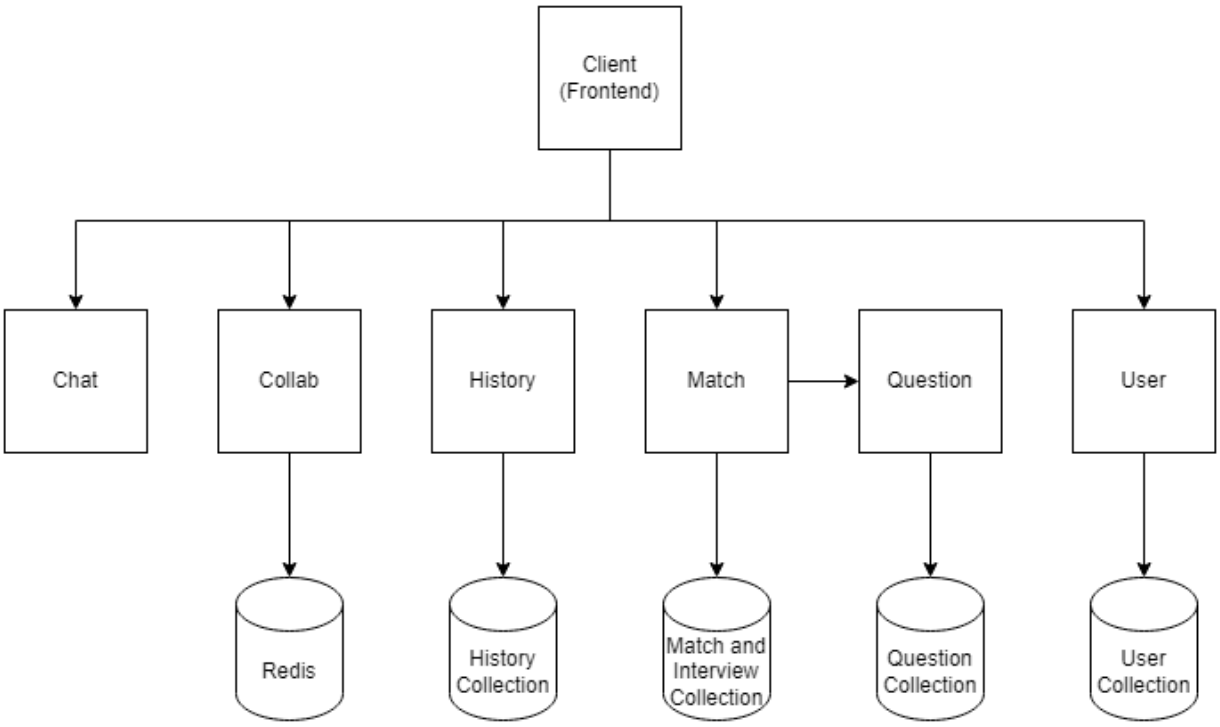
Feature	Expected outcome	Assignee	Status
Frontend (Login, Home)	Clean up the front end.	Kai Han	
Question service	Complete service.	Herrick	
Matching service	Adjust the matching service to meet any other needs of the front end. Dockerize microservice and add tests.	Royce	
Frontend (Matching, Interview)	Clean up frontend for matching and interview.	Wei Sheng	
Collaboration service	Complete working version of collaboration service for real-time code editor.	Herrick	
User service	Dockerize microservice and add tests.	Kai Han	

CI Github Actions Workflow	Add workflow scripts	Herrick / Royce	
History service + Frontend	Complete service.	Wei Sheng / Royce (Frontend)	

Milestone 3 (Weeks 10-12)

Feature	Expected outcome	Assignee	Status
Chat service + Frontend		Kai Han	
Update UI	Clean up frontend	Herrick / Royce	
Testing and Fixing Bugs		All	

System Architecture



The frontend/client is the presentation layer where users interact with the user interface. The frontend interacts with the various microservices through HTTP requests.

The backend is divided across 6 separate microservices:

1. Chat service: Allows users to send messages in real-time to each other
2. Collab service: Allows users to work on a text editor in real-time collaboratively.
3. History services: Store past interview details and allows users to view their history
4. Match service: Handles matchmaking of users who are searching for another user to do an interview with.
5. Question service: Stores interview questions and allows the Match service to retrieve a question from its question bank whenever an interview is to be created.
6. User service: Handles login, registration and JWT validation.

The database used is MongoDB, and stores all the required data for the application.

More details about each microservice will be found below.

Tech Stack

Domain	Technology
Frontend	React
Backend	Express.js/Node.js
Database	MongoDB
Cache	Redis
Pub-Sub Messaging	Socket.io
Testing	Mocha
CI/CD	GitHub Actions
Orchestration Service	Docker-Compose
Project Management Tools	GitHub Issues

Design Considerations

Frontend (ReactJS)

React JS offers tons of benefits. The component-based approach, well-defined lifecycle, and use of just plain JavaScript make React very simple to learn, build a professional web (and mobile applications), and support it.

Components allow developers to break down complex UI. Instead of worrying about the entire web app, it makes it possible to break the complex UI/UX development into simpler components. This is crucial in making every component more intuitive.

ReactJS has a massive active community and its GitHub Repository has over 164k Stars. It is one of the Top 5 Repositories on GitHub. Popularity is one of the important factors when it comes to choosing a frontend library for development as well due to community support.

Backend (Microservice Architecture)

The microservice architecture breaks down software development into smaller, independent “chunks,” where each chunk executes a particular service or function.

Some of the benefits of this architecture include:

- Scalability
    - Individual services can also be scaled independently, and new components can be added without requiring downtime and redeployment of the entire system.
    - Services can also be deployed in multiple servers which reduces the performance impact of more demanding components.
- Faster development
    - Breaking an application down into smaller fragments makes it easier to build and maintain. Each service can be developed, deployed, and managed independently.
- Resilience
    - Implementing microservice-based architecture adds ease to the process of identifying and resolving the root cause of performance issues.
- Optimize business functionality
    - When the focus is on a specific service versus the entire application, it's easier to customize the needs of each component to improve business functionality.

Database

A comparison between SQL and NoSQL databases below highlights the usefulness of a NoSQL database for this project.

	SQL databases	NoSQL database
Schema	Predefined	Dynamic
Scalability	Harder to scale horizontally	Easy to scale horizontally
Data size	Better for smaller data sets	Scales to handle large data sets
Properties	ACID compliant	Adheres to the CAP theorem, in this case, Mongo is CP

SQL database requires careful planning of schema given that changes in the future would require costly changes to the codebase. The flexibility of NoSQL's dynamic schema gives room for change. NoSQL databases are also easy to scale to handle large amounts of data compared to SQL databases as well.

For our NoSQL database, we selected MongoDB as it has all the benefits of NoSQL, and its document model is a simple and powerful way to store and retrieve data. Deployment options are also easy and its horizontal, scale-out architecture can support huge volumes of traffic and data. Another big plus is MongoDB's large and mature platform ecosystem, making it easy to find resources and help online.

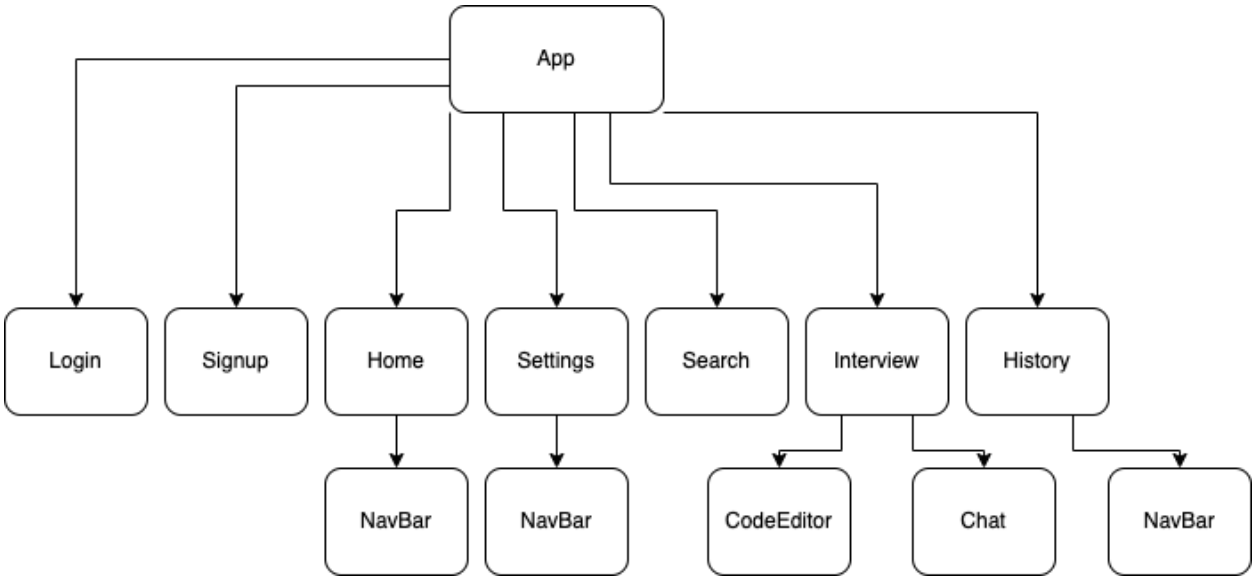
Frontend

Tech Stack

Component	Library/framework used
Frontend	React
UI Styling	React Bootstrap
Countdown Timer	React Countdown Circle Timer
Code Editor	CodeMirror
Question	React Markdown

Architecture

The frontend consists of the different components as shown below. The first row of components represents the different pages accessible in the application (eg. login, signup, interview etc.). The CodeEditor and Chat components are abstracted and used inside the Interview component. We will show more examples of how each component looks in the next section



React Components

Documentation of React components, how they are related to other components and their purposes. Each colour in the table below represents components belonging to a particular page.

Page	Component	Child of	Purpose
All	App	-	Handles routing.
Login	LoginPage	App	The LoginPage component allows the user to log in to their account. Upon successful login, they would be redirected to the home component. Clicking the signup button will redirect the user to the Sign-Up page.
	AlertMessage	Login	Display error messages.
Sign-Up	SignupPage	App	The SignupPage component allows the user to create an account and upon successful creation of an account, the user would be redirected to the home page. The user can also be redirected back to the login page by clicking the sign-in button.
	AlertMessage	SignupPage	Display error messages.
Home	HomePage	App	The HomePage component allows users to choose a question difficulty and search for a match by clicking the match button. The user will then be redirected to the search page to be matched with another user.
	NavBar	HomePage	Navigation Bar for users to access home, history and settings pages. The user can also click the logout button to logout.
	AlertMessage	HomePage	Display error messages.
Search	SearchPage	App	The SearchPage component is where the user will search for a match. A countdown timer of 30 seconds will be shown, if the timer expires and a match is not found, the user will be redirected back to the home page. The user can also cancel finding a match manually by clicking the cancel find match button and a popup would then be

			shown to confirm if the user wants to cancel finding a match. Upon cancellation, the user will also be redirected back to the home page.
Interview	InterviewPage	App	The InterviewPage component contains the CodeEditor and Chat components and allows users to end the interview by clicking the end interview button. The component also shows the question difficulty and usernames of the matched pair.
	CodeEditor	InterviewPage	The CodeEditor component allows the user to type his/her code in collaboration with the other matched user. The CodeEditor component will update in real-time the inputs of both users.
	Chat	InterviewPage	The Chat component allows the user to communicate with the other matched user. A notification message will be sent when a user joins or leaves the interview.
Settings	SettingsPage	App	The settings component allows the user to change his/her password or delete his/her account. When the user changes or deletes his/her account, they will be redirected to the login page.
	NavBar	SettingsPage	Same as above.
History	HistoryPage	App	The HistoryPage component shows all previous interviews that the user has done before in a table. The table consists of details such as the other matched user, the difficulty of the question attempted and the question name with a hyperlink to the leetcode page of the question.
	NavBar	HistoryPage	Same as above.

Microservices

User Service

The user service manages account-related information, from registration to login and authentication with json web tokens (JWT). It also supports additional features such as logging out, changing password and deleting user accounts. User-service includes an authentication middleware for endpoints that require authentication by validating the JWT token. Passwords are hashed before being stored in the database for security reasons.

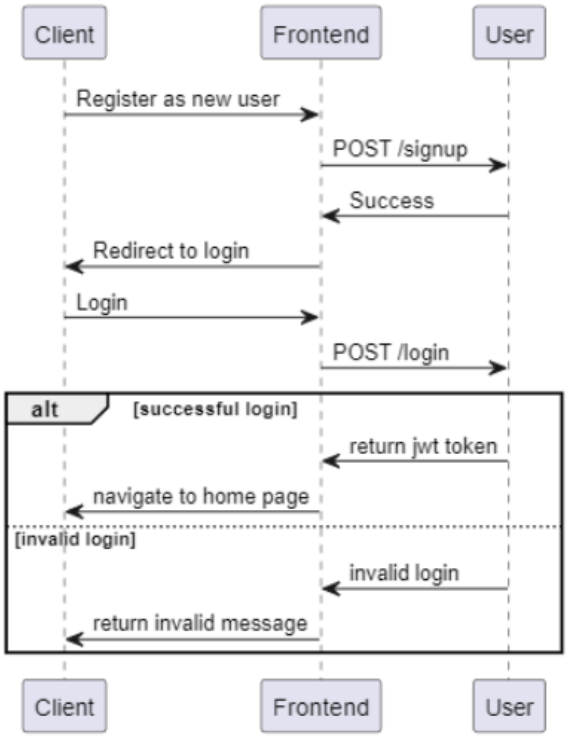


Figure A: Signup, Login and Logout

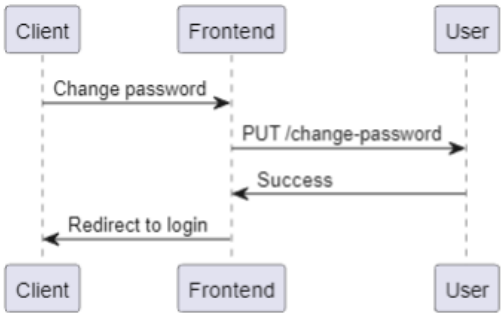


Figure B: Change password (User logged in)

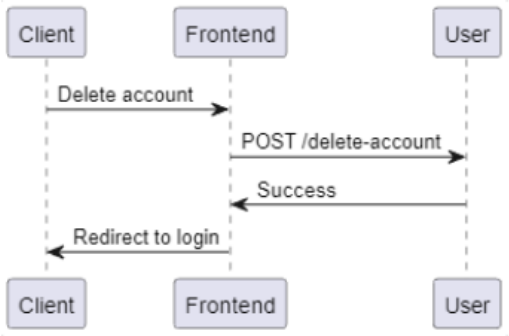


Figure C: Delete account (User logged in)

Figure A shows a successful signup which will redirect the user to the login page. The user will then login using his/her username and password. Upon successful login, a JWT token will be returned in the response message and will be used in the authorization header to authenticate the user for the other endpoints. Figure B and C show the sequence diagrams for changing password and deleting account respectively.

User APIs

`GET /`

Status Code	Response / Error
200	`{ "user": "CS2101" }`
500	`{ "message": "Unable to get user" }`

`POST /signup`

Status Code	Response / Error
200	`{ "message": "Created new user CS2101 successfully!" }`
400	`{ "message": "Username and/or Password are missing!" }`
400	`{ "message": "Username has been taken!" }`
500	`{ "message": "Could not create a new user!" }`
500	`{ "message": "Could not hash password" }`
500	`{ "message": "Database failure when creating new user!" }`

`POST /login`

Status Code	Response / Error

200	<pre>{
	  "message": "CS2101 logged in successfully"
	}</pre>

400	<pre>{
	  "message": "Username and/or Password are missing!"
	}</pre>

401	<pre>{
	  "message": "Invalid username or password!"
	}</pre>

500	<pre>{
	  "message": "Could not generate token"
	}</pre>

500	<pre>{
	  "message": "Database failure when logging in!"
	}</pre>

`POST /logout`

Status Code	Response / Error

200	<pre>{
	  "message": "CS2101 logged out successfully"
	}</pre>

500	<pre>{
	  "message": "Could not blacklist token"
	}</pre>

500	<pre>{
	  "message": "Database failure when logging out!"
	}</pre>

`PUT /change-password`

Status Code	Response / Error

200	<pre>{
	  "message": "Changed password successfully"
	}</pre>

400	<pre>{</pre>

	<pre>{
	"message": "Missing fields!"
	}</pre>

400	<pre>{
	"message": "New password is the same as current password!"
	}</pre>

401	<pre>{
	"message": "Invalid password!"
	}</pre>

500	<pre>{
	"message": "Could not hash new password"
	}</pre>

500	<pre>{
	"message": "Could not update new password"
	}</pre>

500	<pre>{
	"message": "Database failure when changing password!"
	}</pre>

`POST /delete-account`

Status Code	Response / Error

200	<pre>{
	"message": "Deleted account successfully"
	}</pre>

400	<pre>{
	"message": "Password is missing!"
	}</pre>

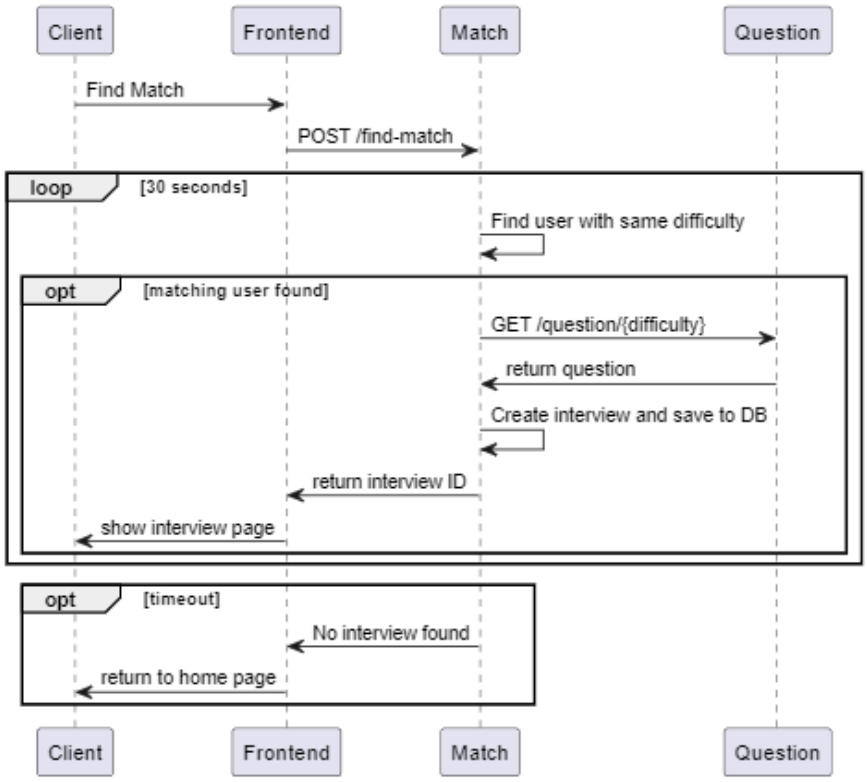
401	<pre>{
	"message": "Invalid password!"
	}</pre>

500	<pre>{
	"message": "Could not delete account"
	}</pre>

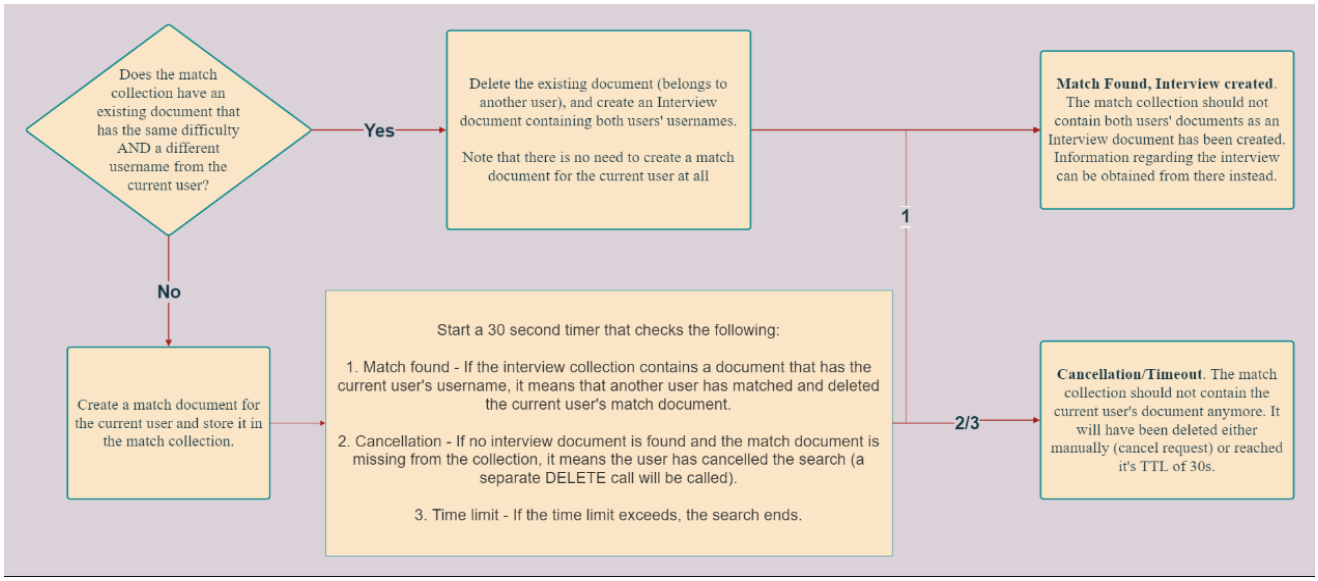
500	<pre>{
	"message": "Database failure when deleting account!"
	}</pre>

Matching Service

A client will first search for a match through the find-match API for up to 30 seconds. If another user is found (having requested the same difficulty), a question will be selected from the Question microservice and an interview will be created.



Matching Service sequence diagram



Matching Service Flowchart

Match APIs

`POST /find-match`

Status Code	Response / Error

200	<pre>{
	  "message": "INTERVIEW FOUND",
	  "interviewId": "63687930bbd5755c4af010f4"
	}</pre>

200	<pre>{
	  "message": "NO INTERVIEW FOUND",
	}</pre>

500	Server Error

`DELETE /cancel-find-match`

Status Code	Response / Error

200	<pre>{
	  "_id": "6368adee01866c4551e24378",
	  "username": "user1",
	  "difficulty": "hard",
	  "createdAt": "2022-11-07T07:04:14.508Z",
	  "__v": 0
	}</pre>

500	Server Error

`GET /interview-id/:id`

Status Code	Response / Error

200	<pre>{
	  "_id": "6368ae6301866c4551e2438a",
	  "difficulty": "hard",
	  "question": {
	  "_id": "6340e6b9e710a649f0e402ce",
	  "difficulty": "hard",
	  "title": "Wildcard Matching",
	  "description": "Given an input string (s) ...",
	  "link": "https://leetcode.com/problems/wildcard-matching/",
	  "__v": 0
	  },
	  "firstUsername": "user3",
	  "secondUsername": "user1",
	  "createdAt": "2022-11-07T07:06:11.943Z",</pre>

	"__v": 0 }
500	Server Error

`DELETE /end-interview/:id`

Status Code	Response / Error

200	{
	"_id": "6368ae6301866c4551e2438a",
	"difficulty": "hard",
	"question": {
	"_id": "6340e6b9e710a649f0e402ce",
	"difficulty": "hard",
	"title": "Wildcard Matching",
	"description": "Given an input string (s) ...",
	"link": "https://leetcode.com/problems/wildcard-matching/",
	"__v": 0
	},
	"firstUsername": "user3",
	"secondUsername": "user1",
	"createdAt": "2022-11-07T07:06:11.943Z",
	"__v": 0
	}

500	Server Error

GET /interview-username/:username

Status Code	Response / Error

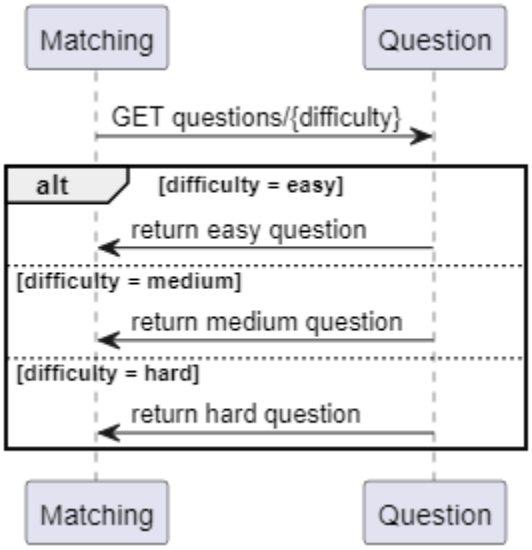
200	{
	"_id": "6368ae6301866c4551e2438a",
	"difficulty": "hard",
	"question": {
	"_id": "6340e6b9e710a649f0e402ce",
	"difficulty": "hard",
	"title": "Wildcard Matching",
	"description": "Given an input string (s) ...",
	"link": "https://leetcode.com/problems/wildcard-matching/",
	"__v": 0
	},
	"firstUsername": "user3",
	"secondUsername": "user1",
	"createdAt": "2022-11-07T07:06:11.943Z",
	"__v": 0
	}

200	{
	"message": "NO INTERVIEW FOUND",
	}

500	Server Error

Question Service

The question service allows the matching service to fetch various types of questions for the interview based on difficulty. Depending on which difficulty is selected, a GET request will be sent to retrieve a random question of that difficulty.



Sequence diagram of question service scenarios

Question APIs

`GET /questions/{difficulty}`

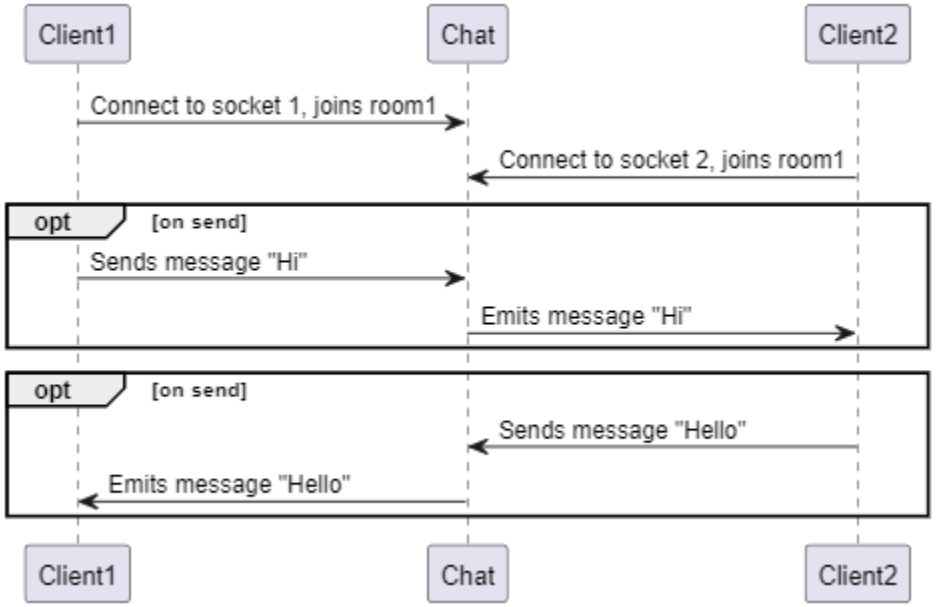
Status Code	Response / Error

200	<pre>{
	  "difficulty": "easy",
	  "title": "Two Sum",
	  "description": "Given an array of integers nums and an integer target, return indices of the two numbers..."
	  "link": "http://leetcode.com/problems/two-sum/"
	}</pre>

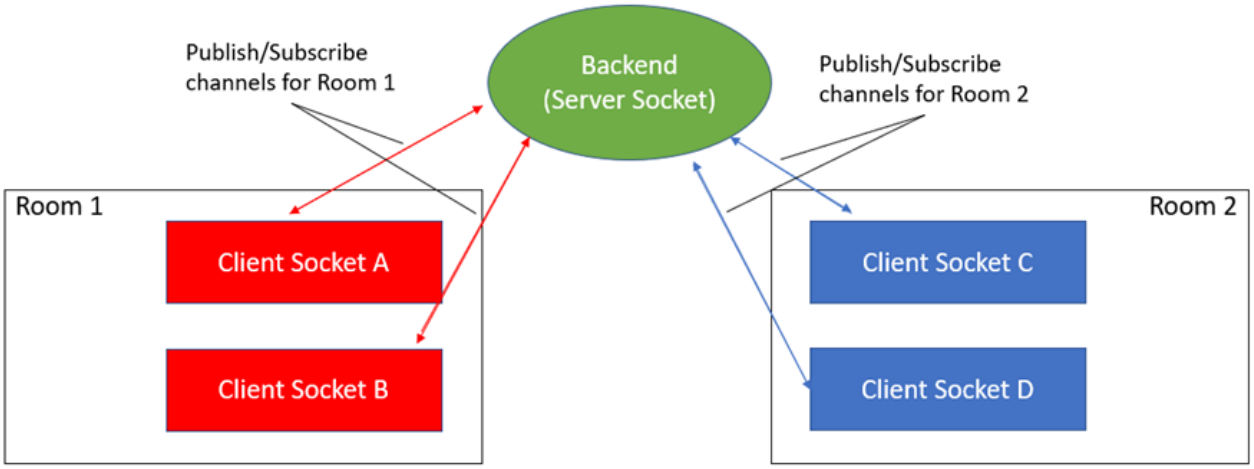
500	Server error

Chat service

Chat service provides a real-time chat for both users to communicate with each other in a chat box. Both users will connect to a socket, and join a common room based on the interview id.



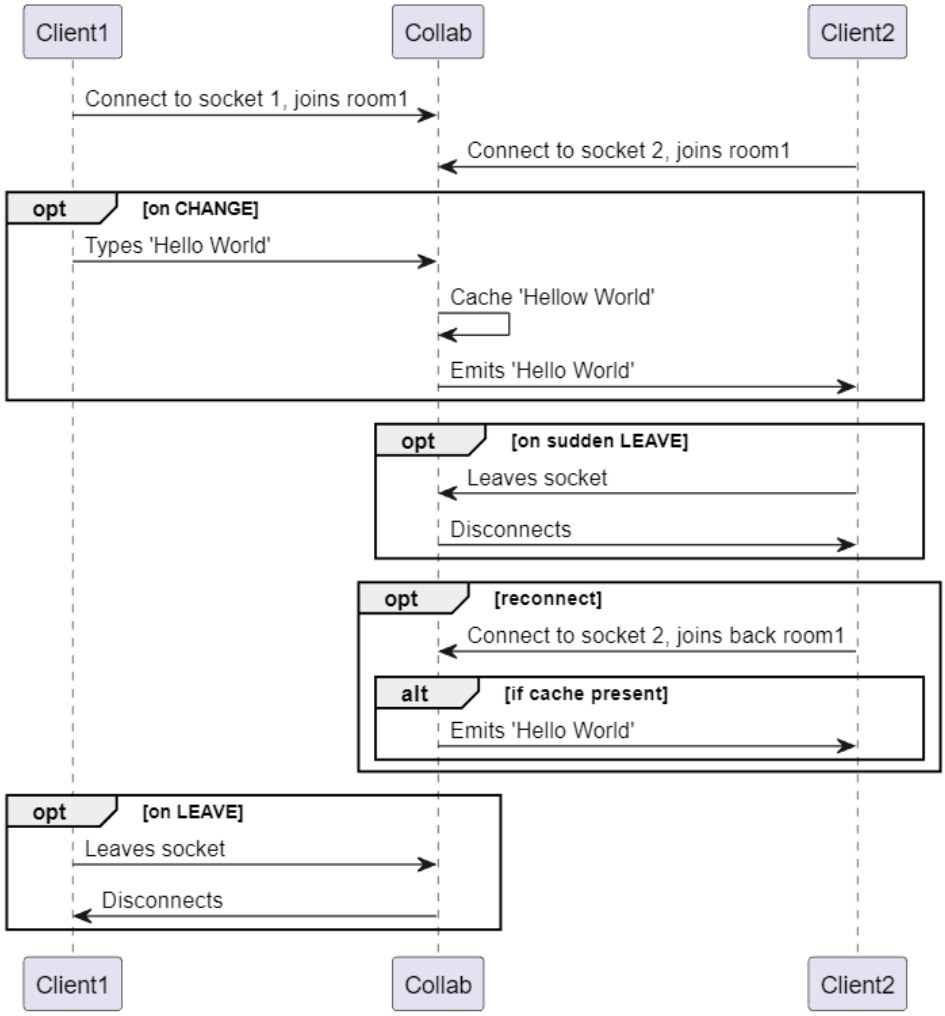
Publish/Subscribe Framework



Whenever a client types a message in the chat box and press send message, the message will be emitted to the server socket which then broadcasts to the other client socket in the room. Both users will be able to see their messages update in real time.

Collab Service

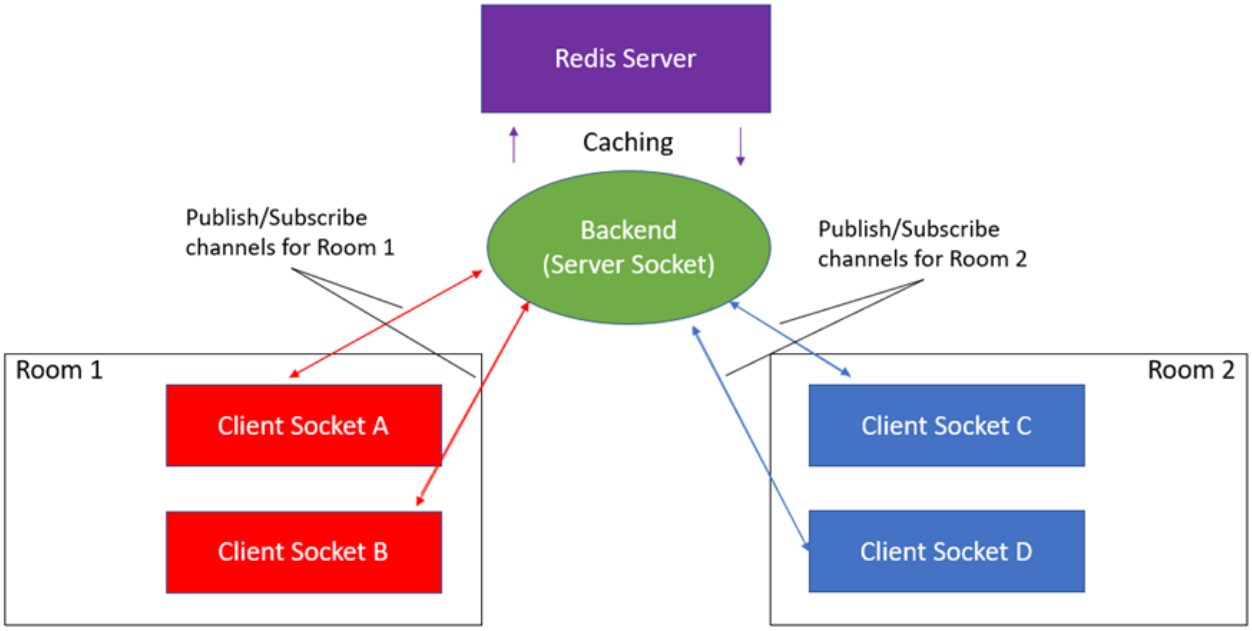
Collaboration (Collab) Service provides a real-time code editor, allowing both users within the interview to share their ideas and work together to solve the given question.



Sequence diagram of collab-service scenarios, both including the use of a cache or not.

Each change emitted by one client will be cached and emitted to the other client connected and subscribed to the same room. If one client disconnects abruptly and reconnects after changes are present, the cache will be emitted to them updating their code editor to where they left off.

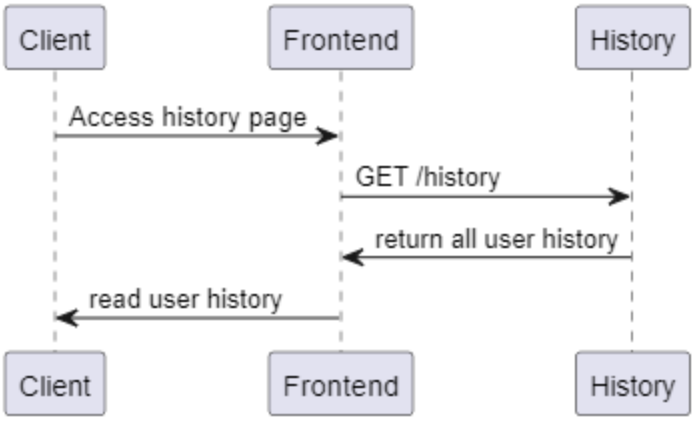
Publish/Subscribe & Cache Framework



Whenever a client edits the code within the code editor, the change will be emitted to the server socket which then broadcasts this change to the other client socket in the room. In the meantime, the change will be cached in the Redis server. When a client connects late or reconnects due to unforeseen circumstances, the backend server will emit the current cache back to the client, updating his code to the latest one.

History service

The history service allows for the tracking of past questions done by the user and also includes which other user he/she attempted the question with.



Whenever a user joins an interview, a create history request will be sent including both the interview question and the username of the other interviewee. This data will then be stored in the database and can be retrieved when the user enters the history page.

History APIs

`POST /create-history`

Status Code	Response / Error
200	`{ "message": "history already created previously for test", }`
200	`{ "message": "interview history saved for test", }`
500	Server error

`GET /get-history/:username`

Status Code	Response / Error
200	`{ "username": "CS2105", "matchUsername": "CS2106",`

	"difficulty": "easy",
	"question": {
	"_id": "6340f3479c538ccde4f8460b",
	"difficulty": "easy",
	"title": "Remove Element",
	"description": "question description (truncated due to length)",
	"link": "https://leetcode.com/problems/remove-element/",
	"__v": 0
	},
	"interviewId": "test"
	}

500	Server error

`DELETE /delete-all-history/`

Status Code	Response / Error

200	{
	"acknowledged": true,
	"deletedCount": 1,
	}

500	Server error

Testing

We used Mocha as our testing framework for all backend services and Chai as our assertion library, which can be easily paired with Mocha (or any other javascript testing framework).

```
> user-service@1.0.0 test
> cross-env NODE_ENV=test mocha --require mocha-suppress-logs --exit

User service listening on port 8000


  POST /signup
    Checks if new user is created successfully
      ✓ should return creation success message (487ms)

  POST /login
    Checks if user logged in successfully
Created new user CS2105 successfully!
      ✓ should return login success message and JWT token (94ms)

  GET /logout
    Checks if user logged out successfully
Created new user CS2105 successfully!
CS2105 logged in successfully!
      ✓ should return logout success message

  PUT /change_password
    Checks if user changed password successfully
Created new user CS2105 successfully!
CS2105 logged in successfully!
      ✓ should return change password success message (207ms)

  POST /delete_account
    Checks if user deleted account successfully
Created new user CS2105 successfully!
CS2105 logged in successfully!
      ✓ should return deletion success message (126ms)

  GET /
    Checks if get username function is successful
Created new user CS2105 successfully!
CS2105 logged in successfully!
      ✓ should return get username success message


  6 passing (2s)
```

Unit Testing for User microservice

```
> matching-service@1.0.0 test
> mocha --require mocha-suppress-logs --exit

Matching service listening on port 8001


  POST /find-match
    Checks interview database and finds existing interview.
Matching service - connected to MongoDB! Database name: "cs3219-matching-service"
      ✓ should return interviewId since existing interview found. (1104ms)
    Unable to find a matching partner
      ✓ should timeout after 30s. (Match document will expire after 30s also). (31289ms)

  DELETE /cancel-find-match
    ✓ should remove match document from collection.

  GET /interview-id/:id
    ✓ should return the interview info, including id

  DELETE /end-interview/:id
    ✓ should delete and return the interview info


  5 passing (33s)
```

Unit Testing for Matching microservice

```
> question-service@1.0.0 test
> mocha --require mocha-suppress-logs --exit

Question service listening on port 8002


  GET /easy
    ✓ should return an easy question (403ms)

  GET /medium
    ✓ should return a medium question

  GET /hard
    ✓ should return a hard question


  3 passing (471ms)
```

Unit Testing for Question microservice

```
> collaboration-service@1.0.0 test
> mocha --exit

Connected to redis.
collab-service listening on port 8003


  Code change
A user connected
A user connected
Joined room 123456789!
Joined room 123456789!
    ✓ client2 should receive the change in code (195ms)
```

Unit Testing for Collaboration microservice

```
> chat-service@1.0.0 test
> mocha --exit

Chat-service listening on port 8005


  One user joins room
User connected to socket VOXVKptdjkEF3zynAAAB
    ✓ should return user joined interview (111ms)

  Two users join room
User connected to socket XbOY5H1FiWwlTd4KAAAE
User connected to socket 4FvjS2c2lysaNLKsAAAF
    ✓ should return second user joined interview (46ms)

  Send message
User connected to socket g21mfonyzOskzzSVAAAI
User connected to socket TTtN66nQNzGu98-JAAAJ
    ✓ should return message (55ms)


  3 passing (232ms)
```

Unit Testing for Chat microservice

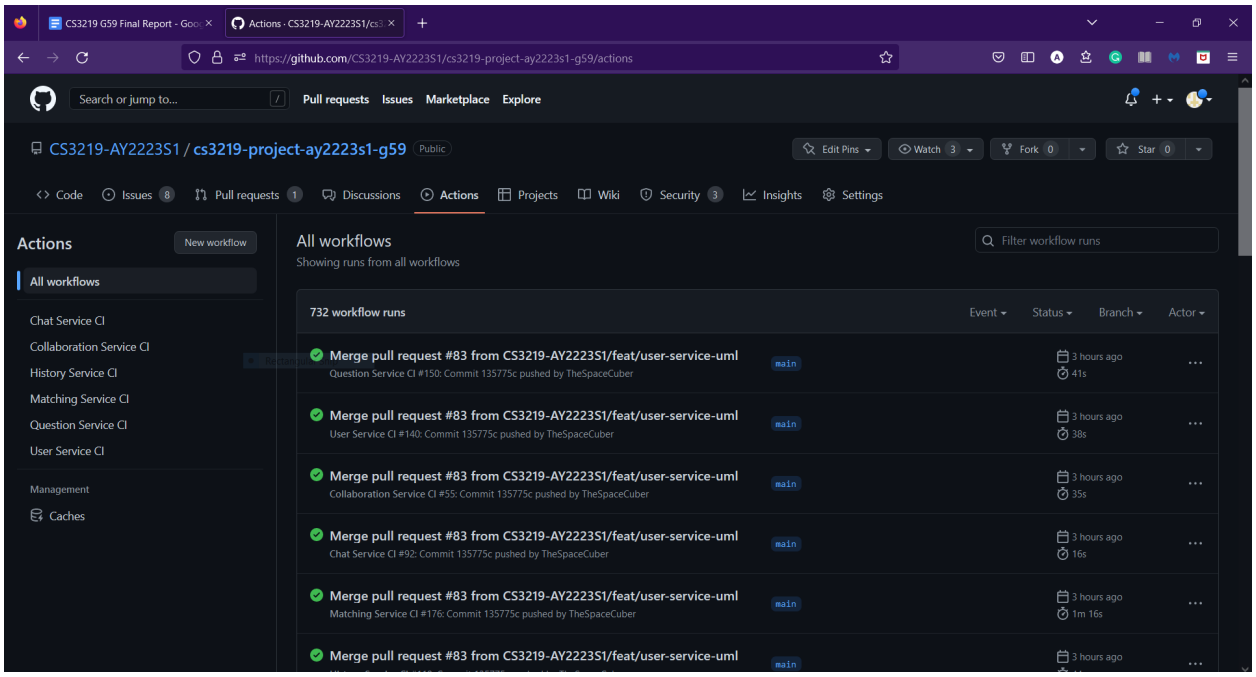
```
> history-service@1.0.0 test
> mocha --require mocha-suppress-logs --exit

History service listening on port 8004


  history-service tests
    POST/create-history
      ✓ Create history successful (384ms)
    GET/get-history/:username
      ✓ Get history succeessful
    DELETE/delete-history/:username
      ✓ Delete history successful


  3 passing (452ms)
```

Unit Testing for History microservice

Additionally, user testing was done to simulate end-to-end testing. More on improving testing in the ‘Possible improvements’ section down below.

Development Processes

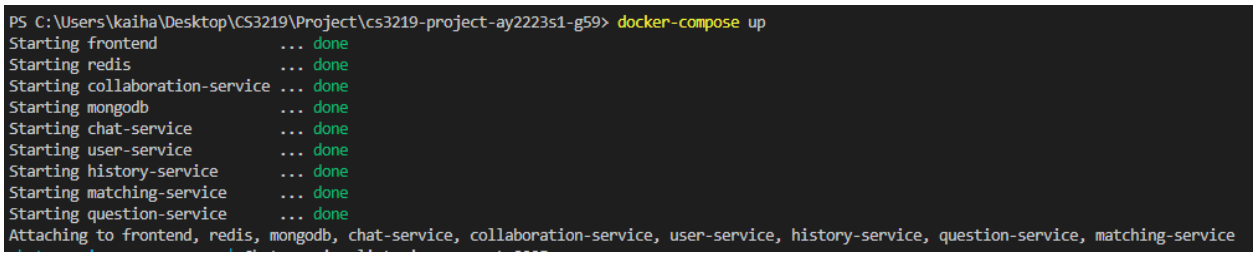
In our project, we used GitHub Actions to manage continuous integration (CI). Tests were incorporated into GitHub Actions to ensure continuous integration upon every merge into the main branch. More info can be found in our repository here.



Example of GitHub actions being used to handle CI

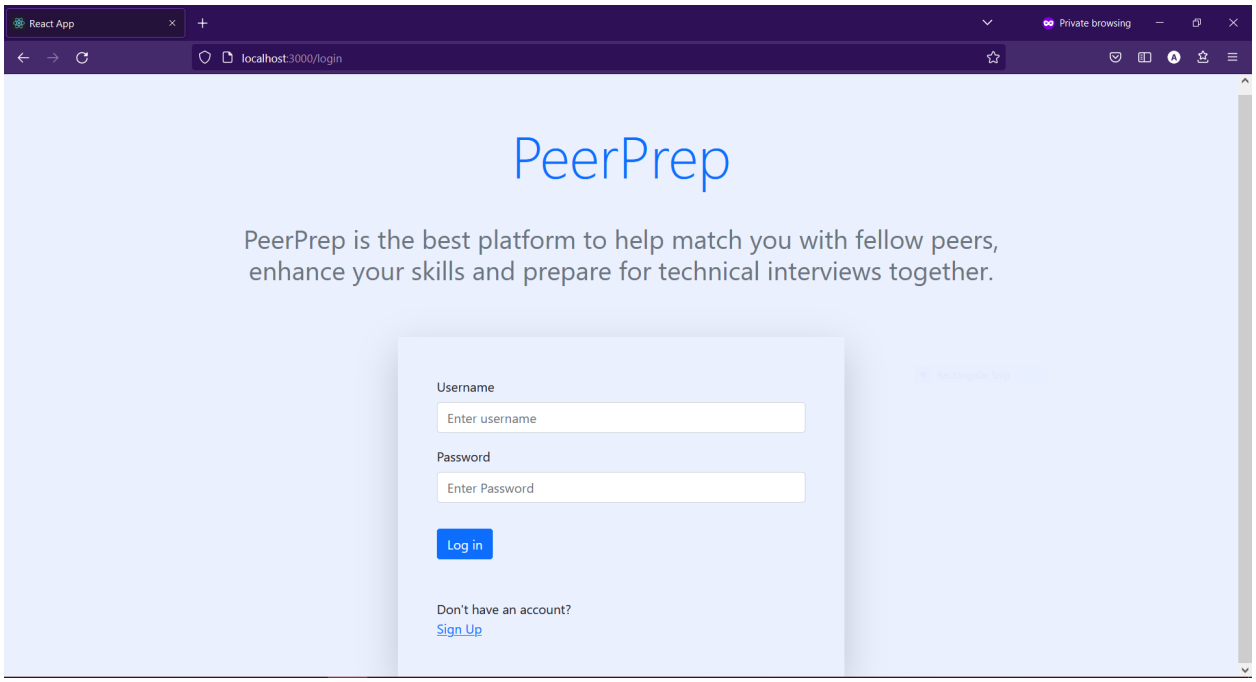
Docker

We also used docker-compose to easily run multiple containers as a single service. More info on how to use docker to start up our project can be found at our repository here.

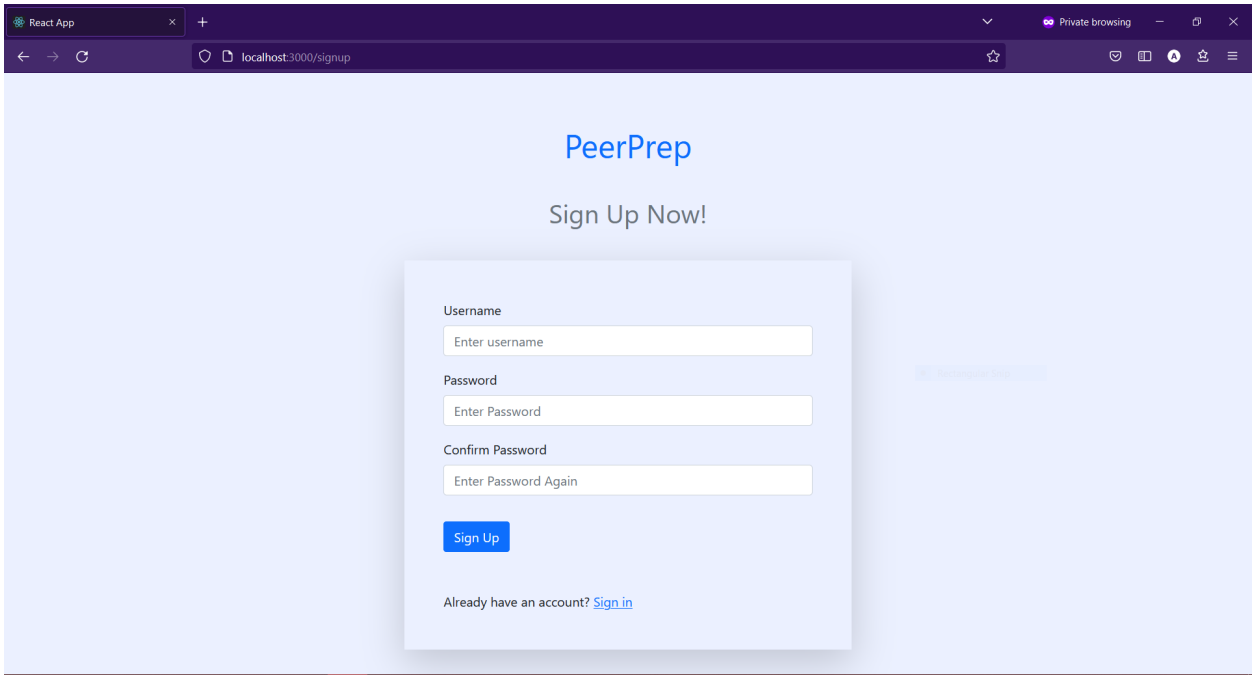


Example of Docker Compose

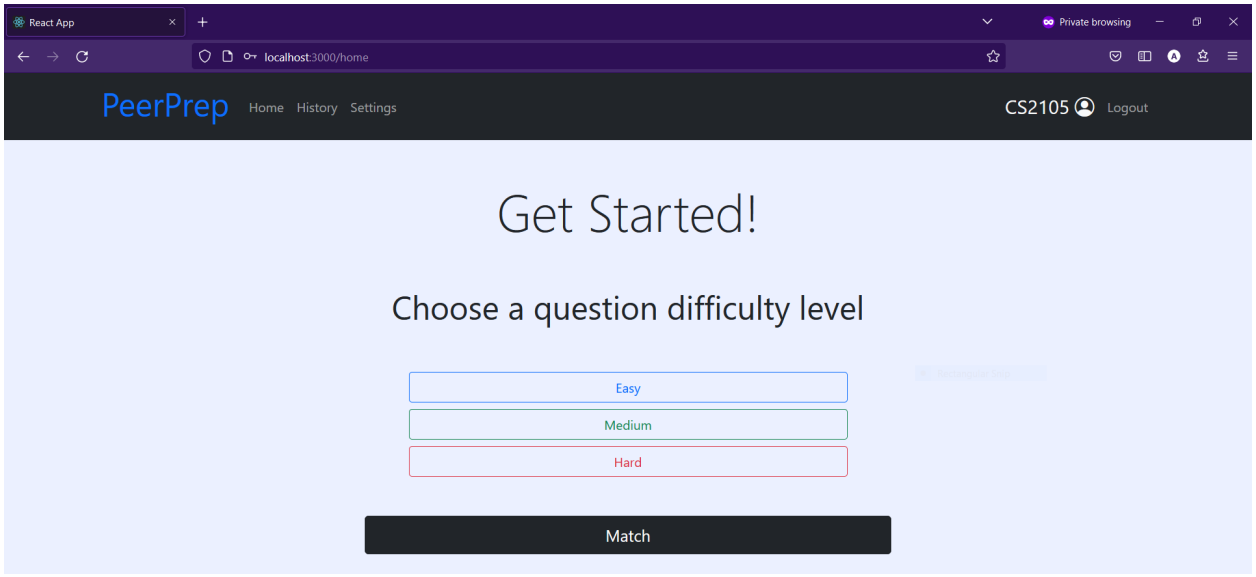
Application screenshots



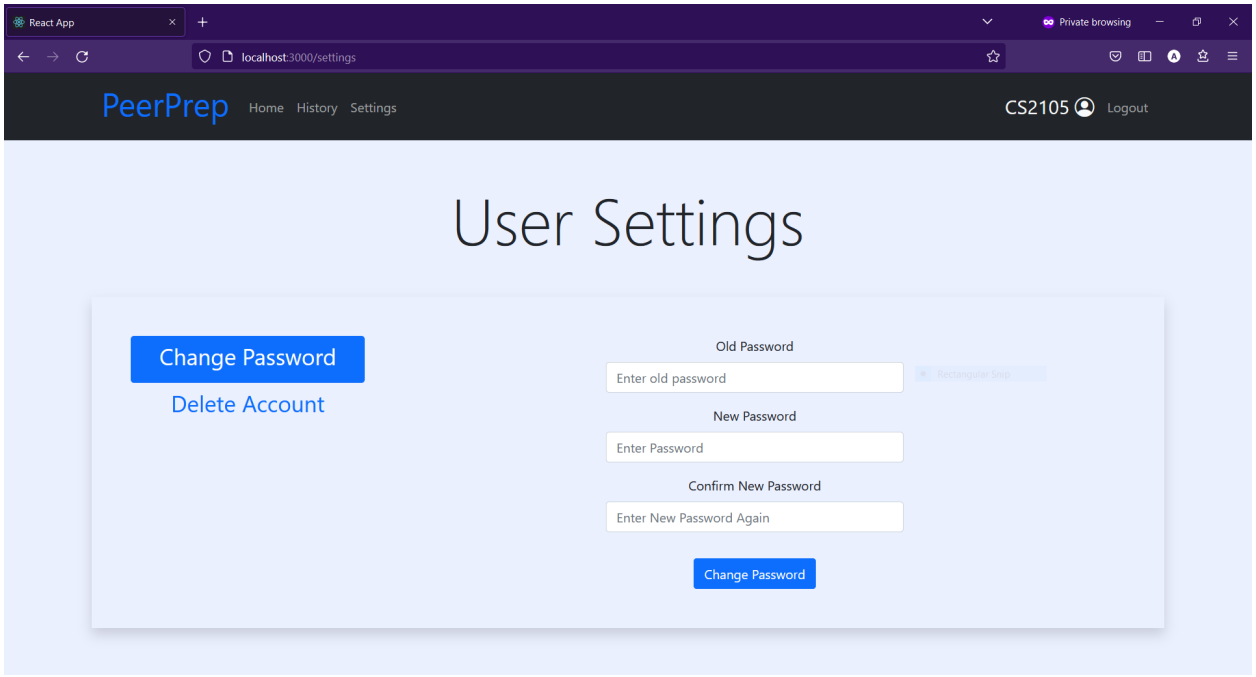
Login Page.



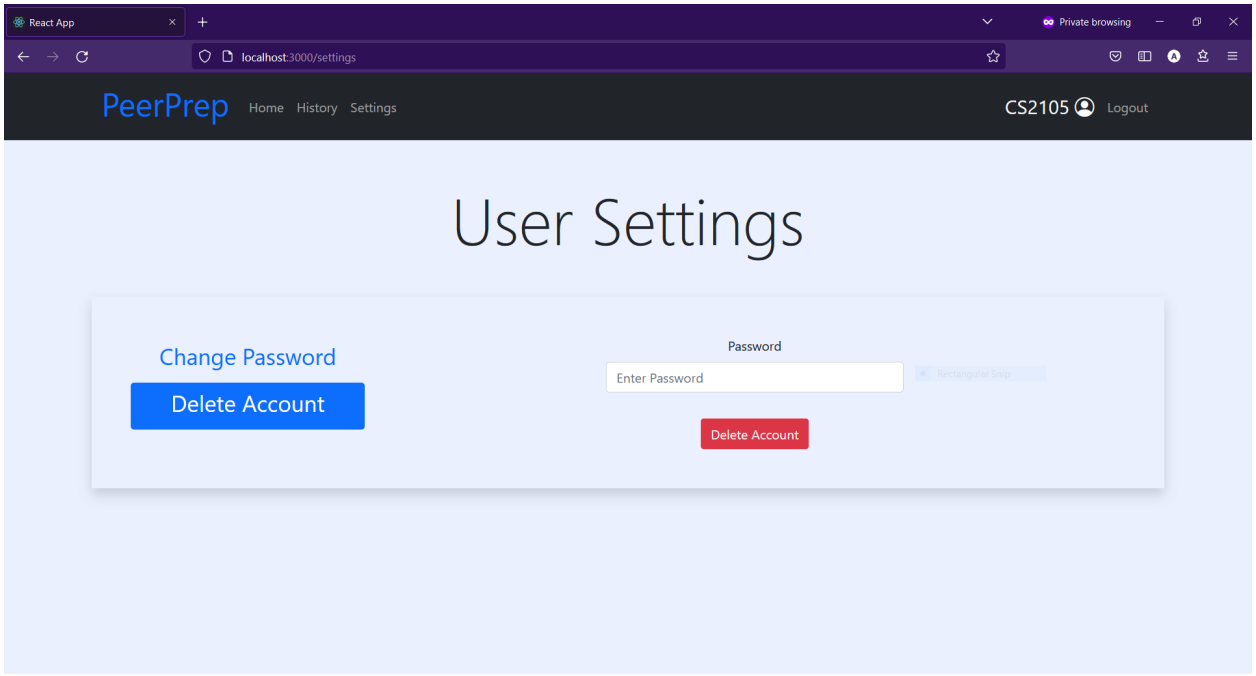
Register Page.



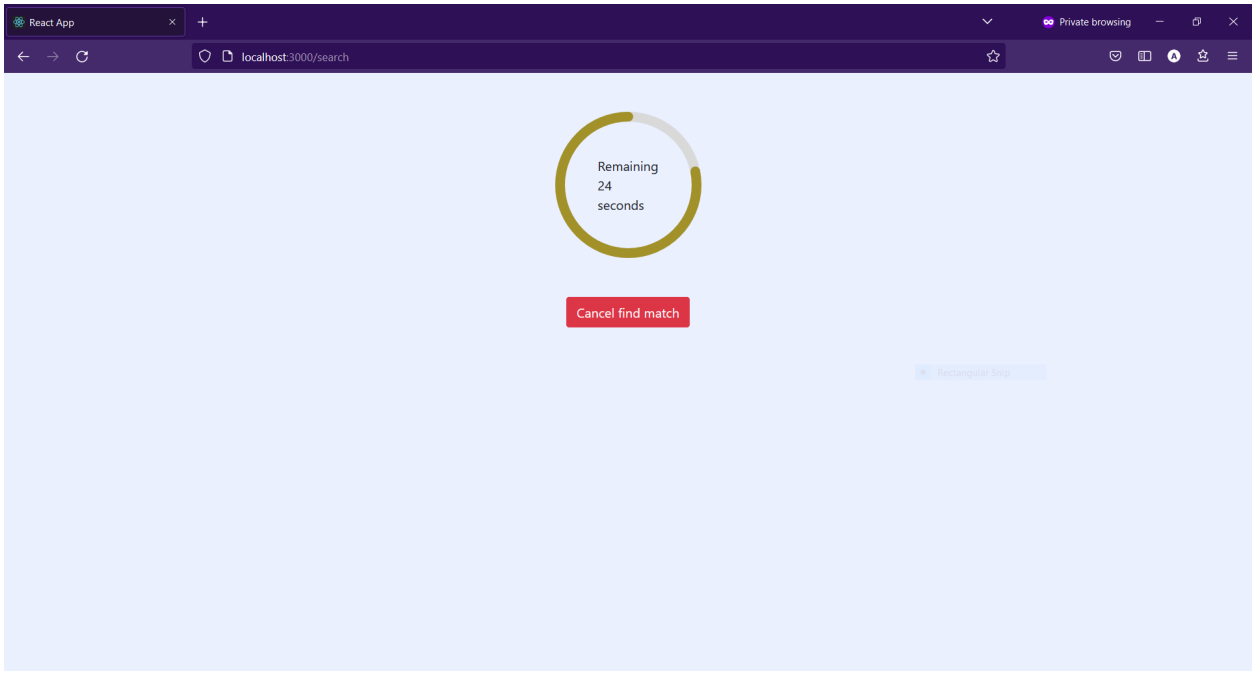
Home Page. Contain difficulties, option to logout, check history, or enter settings.



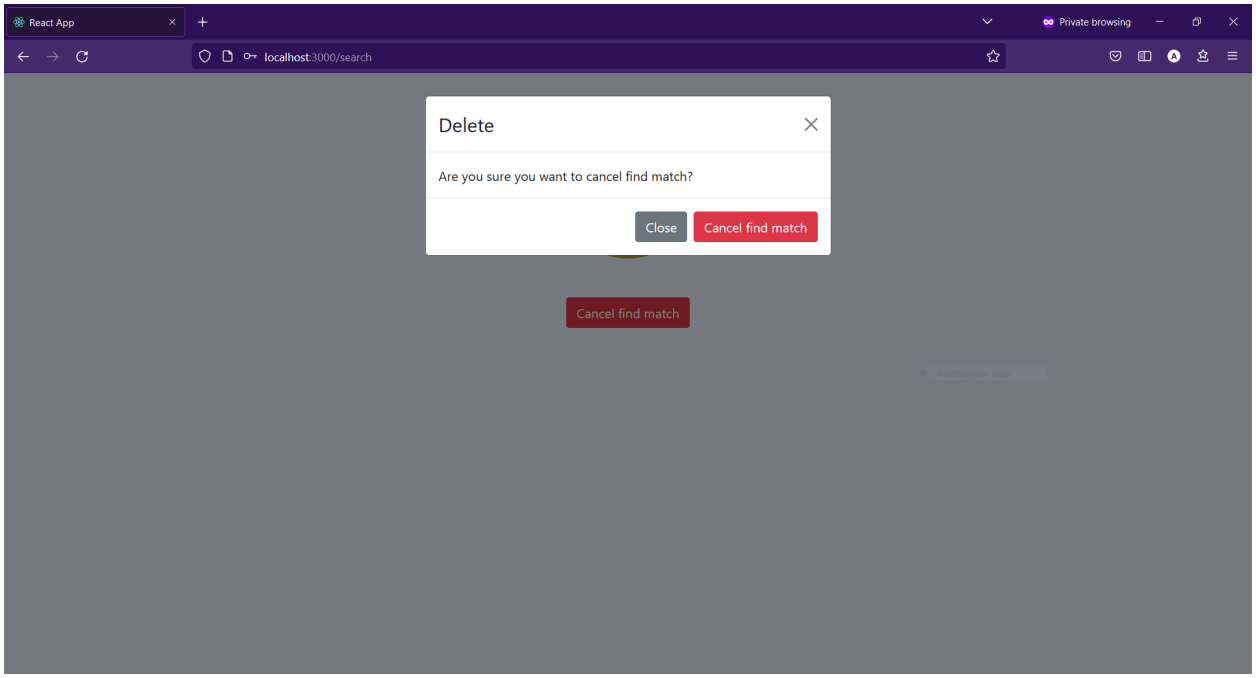
Settings Page: Change password tab.



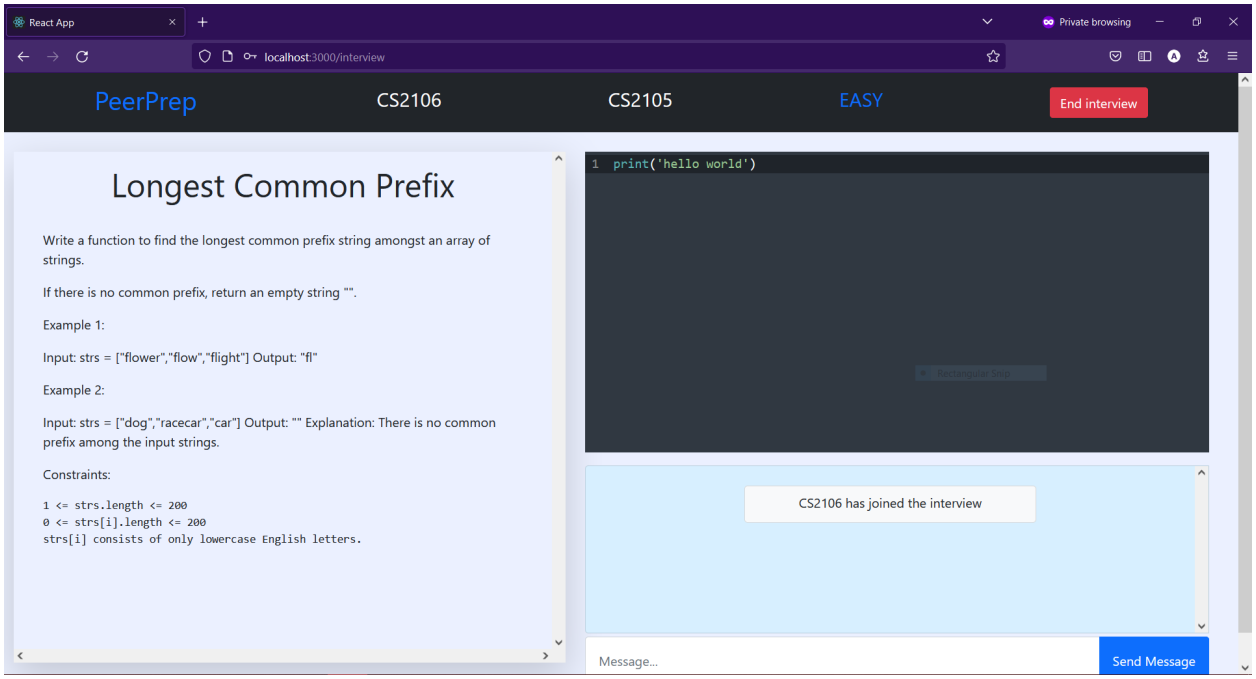
Settings Page: Delete account tab.



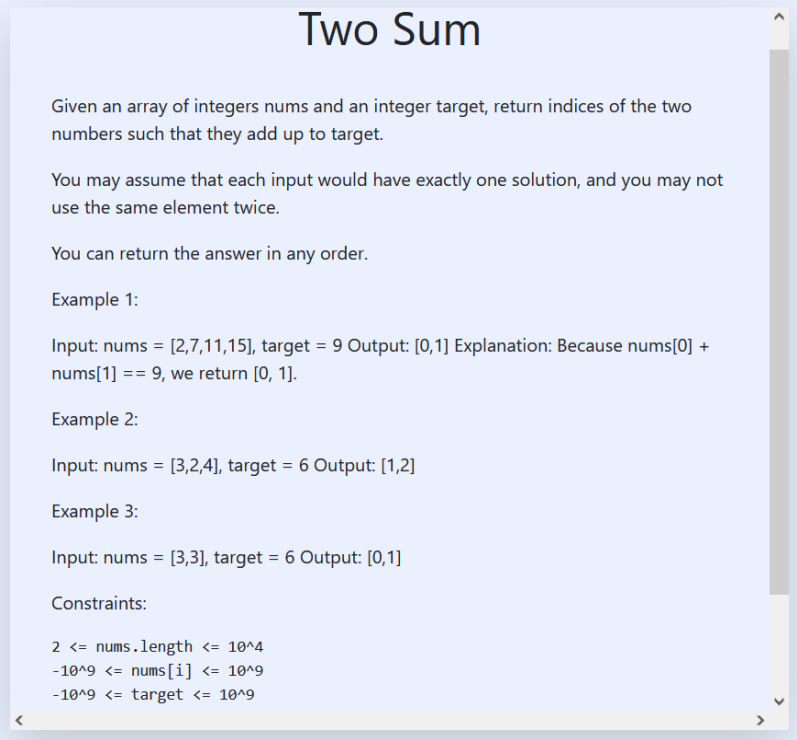
Matching Page.



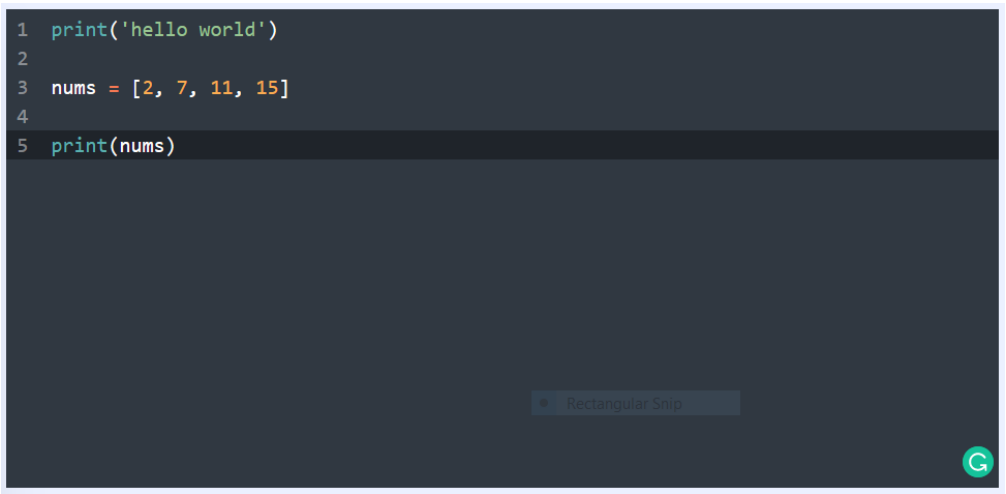
Stop Search Message



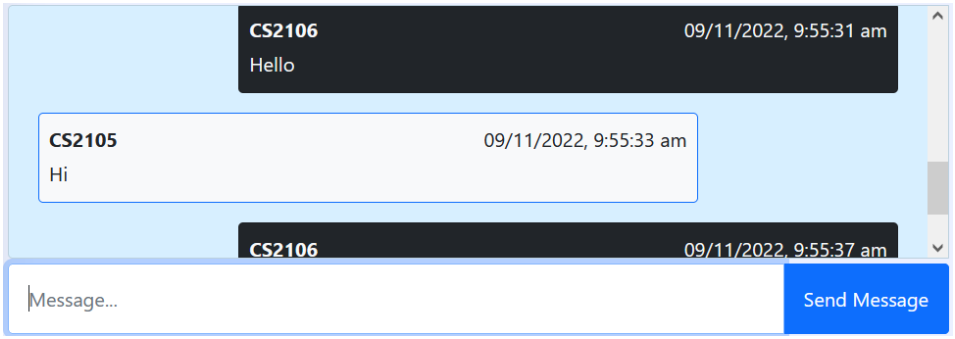
Interview Page. Contains interview question, code editor and chat box.



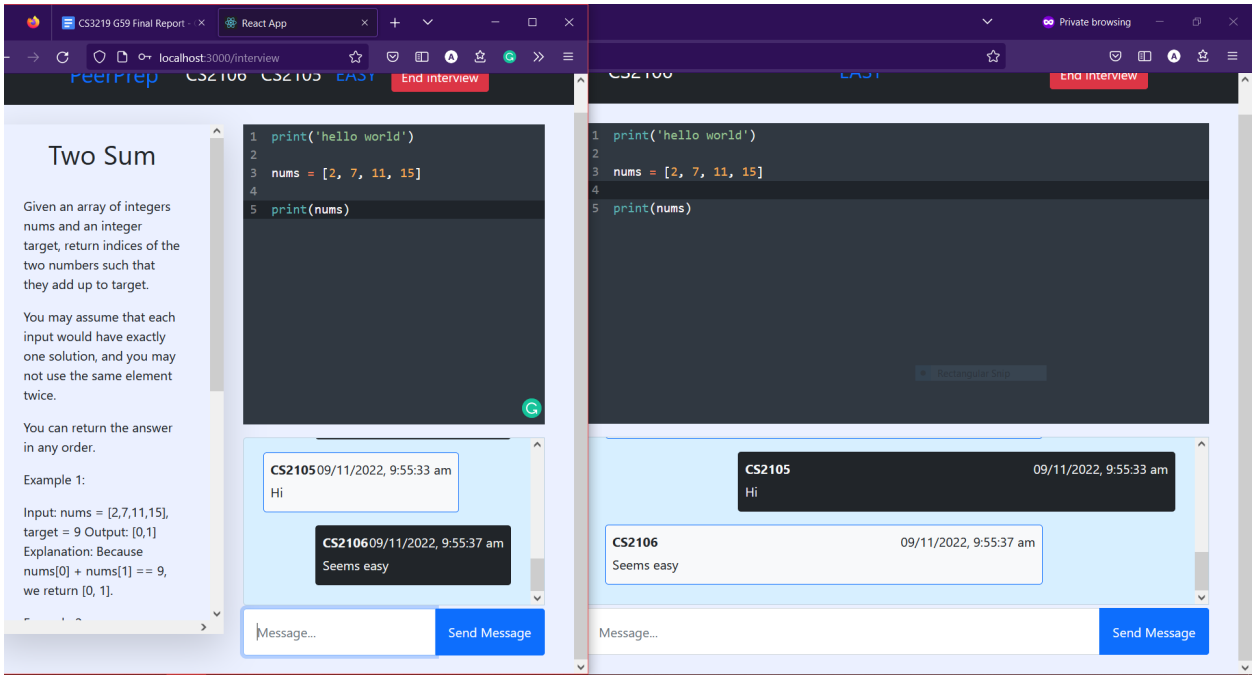
Interview Page: Question Component.



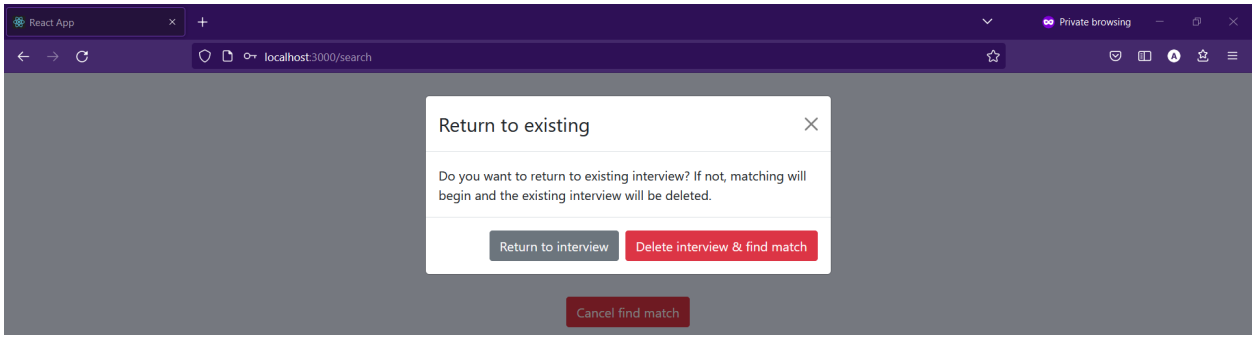
Interview Page: Code Editor Component



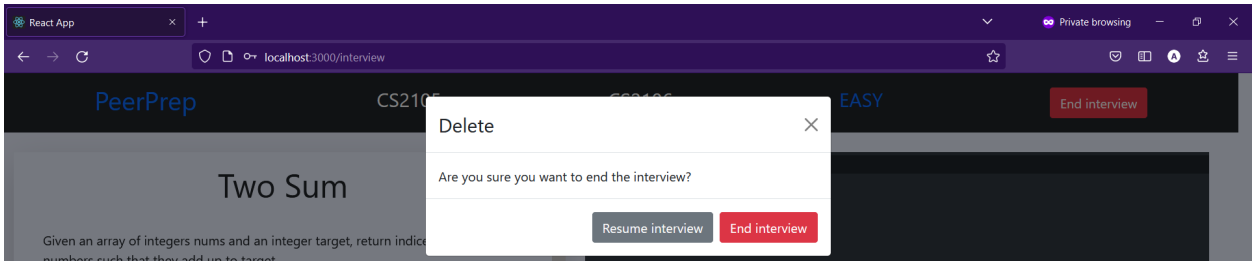
Interview Page: Chat Component



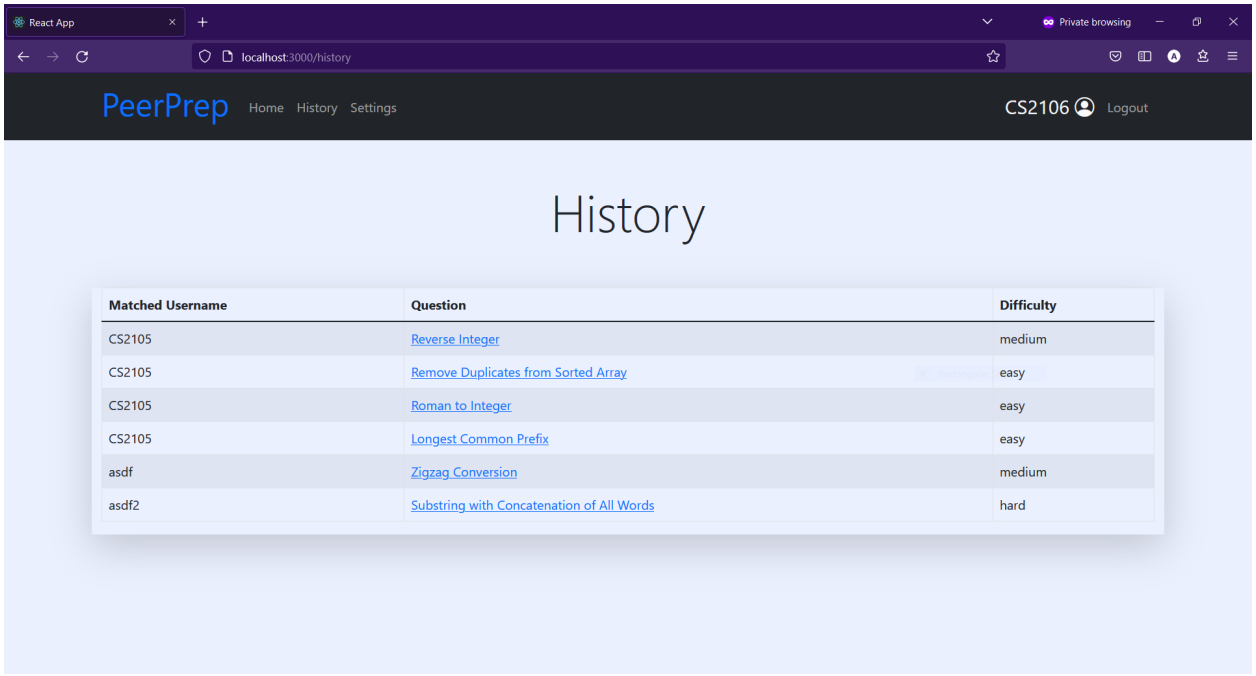
Example of communication between two users via the interview page.



Option to rejoin message. Confirmation whether to find a new match or return to interview.



Ending interview message. Confirmation whether to end the interview.



History Page. Can check previous question, difficulty and who the question was attempted with.

Possible improvements/enhancements

Here are some possible improvements that can be made to the web application:

Domain	Improvements
Testing	Add integration and end-to-end testing. For example, Cypress could be used for Front-end end-to-end tests.
Documentation	Create formal API documentation adhering to OpenAPI standards. Possible tools include Swagger.
Deployment	Deploy to a cloud provider for ease of access for users
Security	Rate limiting to prevent DDOS attacks Web application firewall add-on to filter layer 7 traffic 2 Factor Authentication and Captcha for login
Availability	Multi-zone deployment with load balancing to ensure high availability and fast recovery time objective (RTO)
Backup	Scheduled backups for the database to achieve shorter recovery point objective (RPO)
Scalability	Deployment with Kubernetes (e.g. GKE) for horizontal autoscaling to meet flexible and higher demands Add read replicas for the database to meet high queries per second (QPS)
Performance	Use of content delivery network (CDN) such as may be used to cache static assets for users around the world for faster performance Database sharding for users can be considered if there are many users, which will lessen read load on each database.
Monitoring	Prometheus and Grafana could be used to monitor performance over time. Alternatively, cloud-native services could do that job as well (e.g. AWS Cloudwatch / Google Cloud Operations).
Logging and Analytics	User data and behaviour could be captured, stored, processed and analyzed to make decisions for future updates.

Reflections

Project Management

Throughout the whole project, management and planning for each SDLC was done by us. Using the milestones provided as guidelines, we had to decide which services or upgrades to implement, manage the division of work, and the deadlines required to complete them. Although each member has done so in past modules before (CS2103/CS2113), doing so in this module provided challenges due to its learning curve. To achieve our desired results for each microservices, we had to do our own research and implement softwares and technologies unfamiliar to us.

Technical Skills

MERN stack (MongoDB, Express, Node, React)

The MERN stack allowed us to better understand how a standard 3-tier web architecture works from the frontend to the backend and database layer. We also gained a better understanding of the Model View Controller pattern.

Web security - JWT

JSON Web Tokens are an open, industry-standard RFC 7519 method for representing claims securely between two parties. We learn how JWT tokens are implemented and used in web applications and the problems that they solve.

Real-time technologies - Socket.io

We learnt how to implement real-time technologies such as the ones required in both collaboration and chat microservice. In this case, we used socket-io to manage the transfer of information in real-time in the form of a pub-sub mechanism.

Caching - Redis

The use of redis as a cache allowed us to make our web application more robust. We learnt how to use it as a middleware and improve our application's performance. Thankfully, the OTOT assignments provided us with the background we needed to start implementing it.

Containerization - Docker

Docker sets the new standard for containers today, and through the incorporation of docker we learnt how containerization benefits development, through the use of standardized environments for testing and the portability of containers.

Testing - Mocha/Chai

Test-driven development is essential and taught us the importance of how testing is used to prevent bugs and allow continuous integration.

Microservices Architecture

Although we were taught about it in both lecture and tutorial, at times we found it difficult to implement the microservices architecture into our application due to the need to decouple services and the increased complexity in communication between them. However, we learned and experienced its perks, including its fault tolerance and the ability to deploy services independently for testing.

Conclusion

This module has provided us with valuable knowledge and experience in developing web applications. We have managed to learn about the different technologies and how designing and implementing a web application requires meticulous planning and decision-making. We will be able to take what we have learnt in this module for our future studies and career in the future.