NUS

National University
of Singapore

CS3219 AY22/23 Semester 1
Project: Peer Prep
Project Group 18

Name	Matriculation Number
Ng Hong Ming	A0206226N
Tang Zhi You	A0199947Y
Theodore Pinto	A0199277H
Chan Sin Teng, Tiffany	A0187858A

Table Of Contents

1 The Team

Member	Main Roles	Contributions
Tang Zhi You	● Backend Development ● Websocket Implementation	**Technical contributions:** ● Matching Service APIs ● Websocket implementation for matchmaking ● Scraping Leetcode questions for question bank ● Question Service APIs ● User History Service APIs ● Stress testing
		Non-technical contributions: ● Requirements documentation ● Final Report
Theodore Pinto	● Backend Development ● DevOps	**Technical contributions:** ● User Service APIs ● Authentication Service API's ● Gateway Service Implementation ● CI/CD Framework ● Deployment of microservices on Cloud Run
		Non-technical contributions: ● Requirements documentation ● Final Report
Ng Hong Ming	● Backend Development ● Websocket Implementation	**Technical contributions:** ● Collaboration Service APIs ● Video Conferencing functionality ● Websocket implementation for matchmaking
		Non-technical contributions: ● Requirements documentation ● Final Report
Chan Sin Teng, Tiffany	● UI/UX Design ● Frontend Development ● Websocket Implementation	**Technical contributions:** ● Frontend pages, UI and architecture ● Video Conferencing functionality ● Websocket implementation for video conferencing, chatting and code collaboration
		Non-technical contributions: ● Figma Prototype ● Requirements documentation ● Final Report

2 Technology Stack

Deployment Service	Technology
Cloud Provider	Google Cloud Platform
Deployment	Google Cloud Registry, Google Cloud Run
Local Orchestration Service	Docker-Compose
CI/CD	GitHub Actions
Project Management Tools	Trello, Microsoft Teams
Repository Management	GitHub
Database	MongoDB

Microservice Implementation	Technology
Frontend	React.js, Axios, Bootstrap
User	NodeJS, Redis
Collaboration	NodeJS, SocketIO, PeerJS
Question	NodeJS, Selenium, Beautiful Soup
User History	NodeJS
Testing	Mocha, Chai (Unit/Integration testing), jMeter (stress testing) and PowerBI (to generate report for insightful analysis of test results)

3 Motivation

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.

PeerPrep offers a platform for users to practice whiteboard-style interview questions with other users on the platform, through real time code collaboration along with other communication features, such as video-calling and chat messaging.

4 Understanding the problem

The diagram shown below refers to the bounded context of PeerPrep.

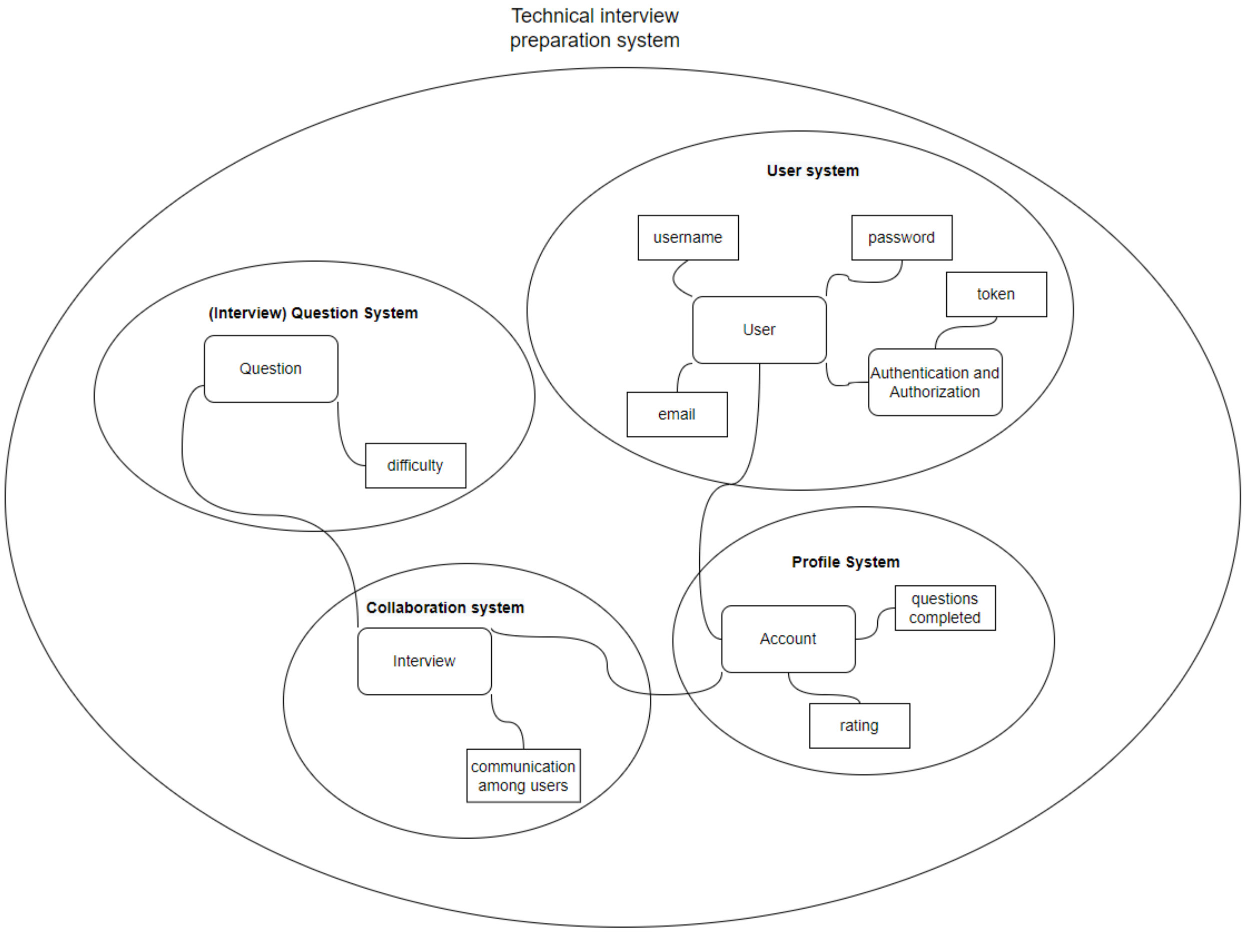


Figure 1: Bounded Context for PeerPrep

Bounded context interpretations:

- User system – a system for users to register an account for this application, with authorization and authentication systems in place.
- Profile system – maintains the account statistics for users to keep track of their progress and history of completed questions.
- Collaboration system[1] - matches users together and allows matched users to collaborate to solve the same question.
- Question system - maintains a question database and retrieves the desired questions.

[1] The collaboration system contains the matching service. This design is more effective due to both services sharing the same technology, SocketIO. If the matching service is implemented separately, it will be difficult to reduce tight coupling between both microservices.

5 Overall Architecture and Design Patterns

A software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. These design patterns are designed based on many great software engineering principles and following them allow us to significantly improve code quality and maintainability. This section describes the architectural design patterns used in PeerPrep, in the form of Microservice architecture and Model View Controller (MVC), while briefly mentioning how the Mediator pattern and Publish-Subscribe pattern have been incorporated, as they will be further elaborated in later sections.

In the beginning, we considered implementing PeerPrep with a **Monolithic** architecture. However, we ultimately decided to go with using the Microservice architecture. This was after consideration of our team's working style of working more independently: since each microservice is identified within an area of a bounded context, this allowed us to isolate the different functionalities to each microservice, thus following the **single responsibility principle**.

Each microservice is owned by a member (or two) of the team, which divides the work and allows for **parallel development**, while allowing us to be very familiar with the service we owned. This design also provides **separation of concerns** across the different services required and **facilitates extensibility**. Each service is independently developed, deployed, and has its own collection schema.

The architecture diagram of PeerPrep is shown in Figure 2 below.

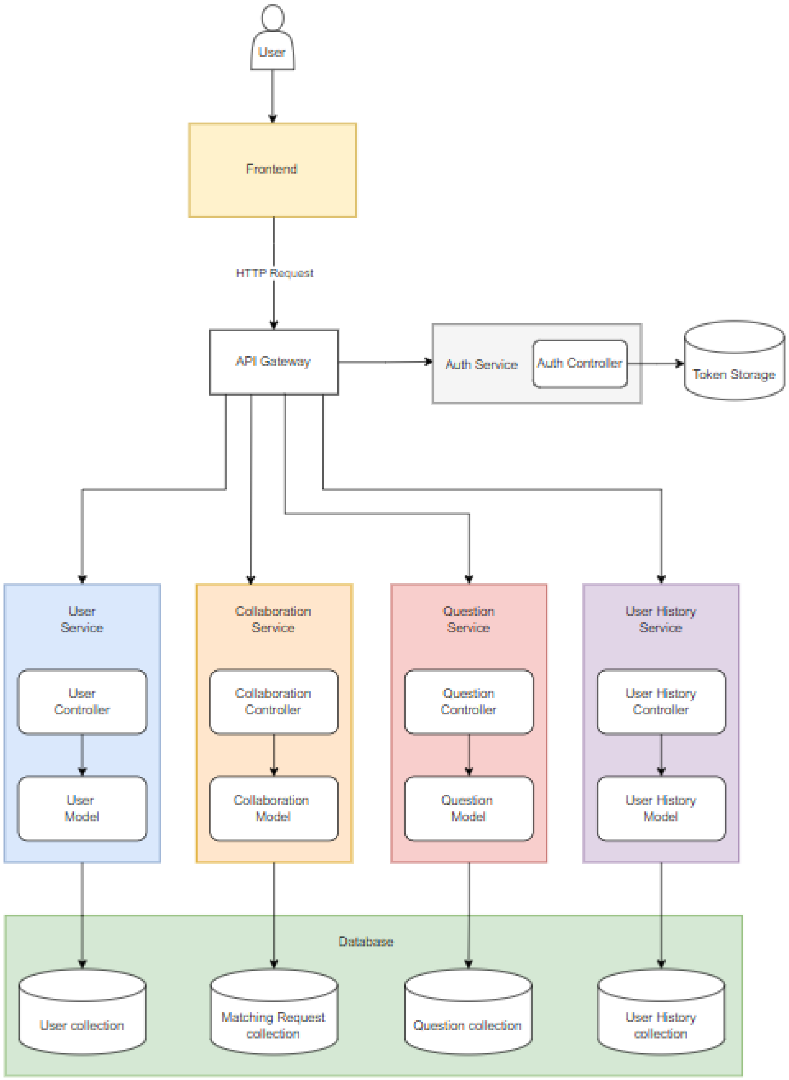


Figure 2: PeerPrep's Architecture Diagram

As shown in Figure 2 above, the user interacts with PeerPrep through the frontend service, making HTTP-based API calls through the various interactions captured by it. Such requests are sent to the **API gateway**, a **nginx reverse proxy**, which **forwards the requests** directly to the respective microservices if no token verification is required. On the other hand, API calls that require validated tokens will be passed to the authentication microservice for token validation before being routed to the intended microservice. As it not only routes the requests directly to the microservices, but also handles some logic and helps to communicate with the authentication microservice, our implementation of the API gateway acts as a **Mediator** in our architecture. More about the API gateway will be explored in section 7.5.

Within each microservice, an **MVC architecture** is adopted. We tweaked the MVC design by abstracting out the View component of each microservice to the User Interface Microservice. This reason for this decision was to ensure that we have clean and consistent code across all components on the front-end.

For each microservice, there is a controller which is responsible for **making a call to the intended MongoDB database** (highlighted in green) or the **Redis Instance** (for the authentication microservice). For the microservices making a call to the MongoDB database, they will craft their request and make it to a collection of the database dedicated to its microservice using a model consisting of a specialized schema.

This design allows us to effectively have **separation of concerns** even within the microservices, whereby different components would handle business logic, application flow and returning API results to users.

Another design pattern that we have implemented is the **Publish-Subscribe messaging pattern** within the Collaboration service. In this pattern, we have subscribers that are listening for any messages published by the publisher. Our usage of this pattern will be further explained in section 6.3.4.2 on messaging architecture.

6 Microservices

This section is devoted to describing the microservices we have developed for PeerPrep. For each of them, we will give a brief description of the microservice itself, its capabilities in the form of the functional requirements it fulfills, the non-functional requirements it meets, the APIs that it provides, how we have implemented it and its collection schema to showcase what data the microservice deals with.

6.1 User Service

The User Service is responsible for maintaining user details of PeerPrep's users. In addition, it is also responsible for creating various types of JWT tokens identifying the user by certain attributes. With such tokens, users will be able to make API Requests which require authorization, which is only approved through successful token validation.

6.1.1 Functional Requirements

S/N	Functional Requirements	Priority
FR1.1	The system should allow users to create an account with username and password.	High
FR1.2	The system should ensure that every account created has a unique username.	High
FR1.3	The system should allow users to log into their accounts by entering their username and password.	High
FR1.4	The system should allow users to delete their account.	High
FR1.5	The system should allow users to change their password.	Medium

6.1.2 Non-Functional Requirements

S/N	Category	Requirements	Priority
NFR1.1	Security	The system should only provide access and refresh tokens when the given credentials are valid	Medium

NFR1.2	Security	Users' passwords should be hashed and salted before storing in the DB.	Medium
NFR1.3	Security	The system should verify the user’s identity through email verification when a request to reset password without prior authentication is made.	Low
NFR1.4	Security	The system should verify the user’s identity through email verification when a request to create an account is made.	Low
NFR1.5	Security	The system should allow users to automatically renew their access to it for a set period with refresh tokens	Low

6.1.3 API Routes

The table below states the various routes supported by the User Service

Request	API Route	Description
GET	/api/user/health	Checks the operational status of the service
GET	/api/user/signup-verify	Creates a new user based on the decoded details of the verification token
POST	/api/user/password-reset	Sends an email with a verification token to given email address provided it is of a registered person
PATCH	/api/user/password-reset-verify	Updates password of decoded user with newly provided password, providing a new refresh and access token if successful
GET	/api/user/get-access	Gets a new access token bearing the same user’s credentials
GET	/api/user/accounts	Gets a list of all accounts registered in PeerPrep

GET	/api/user/accounts/:username	Gets the account of the specified user based on username
PATCH	/api/user/accounts/:username	Updates the password attribute of a given user based on given username
DELETE	/api/user/accounts/:username	Deletes a user based on given username

6.1.4 Services

6.1.4.1 CRUD Operations of User Accounts

This is the main functionality of the User Service. It is responsible for maintaining the data of all users registered with PeerPrep. Using this functionality, the user can perform the ability to make an account on PeerPrep, retrieve or update its details or even delete it if they wish to discontinue the usage of PeerPrep.

The sequence diagram below shows one of the supported routes that deletes a user by username from PeerPrep.

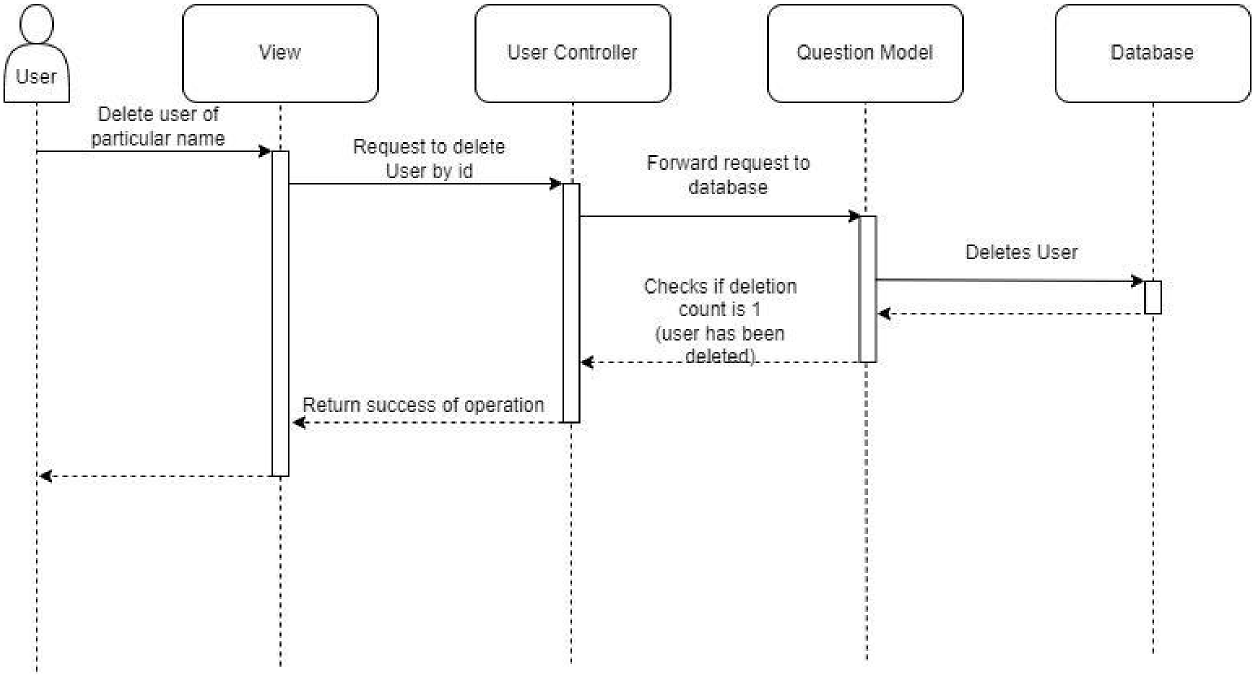

User
View
User Controller
Question Model
Database
Delete user of particular name
Request to delete User by id
Forward request to database
Deletes User
Checks if deletion count is 1 (user has been deleted)
Return success of operation

Figure 3: Sequence Diagram of User Deletion from PeerPrep

6.1.4.2 Email Verification

This functionality was created with the aim to add an **additional layer of security** for API requests that require altering database data without the need for a user to be authenticated in PeerPrep. There are two of such requests that require email verification – resetting a forgotten password, and the creation of a new user on PeerPrep. The diagram below shows the overall flow of such a feature.

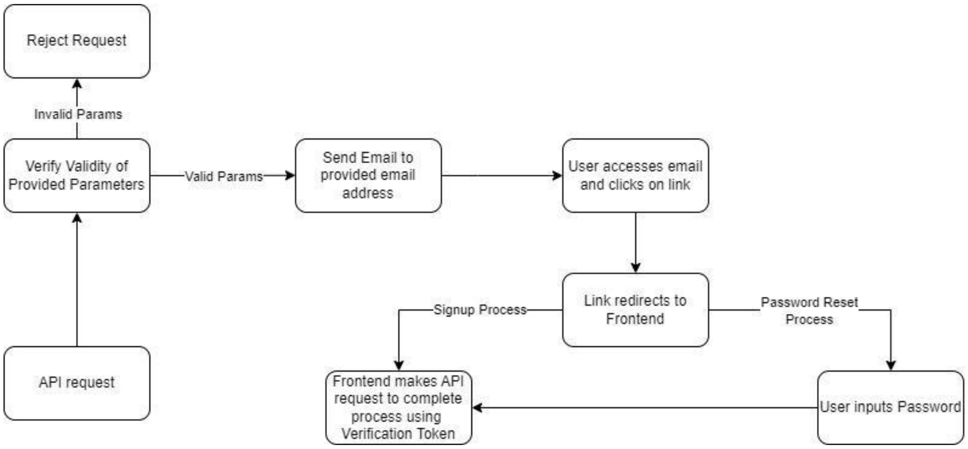
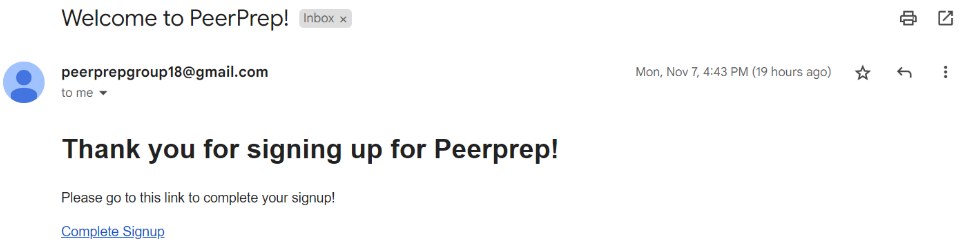

Reject Request
Invalid Params
Verify Validity of Provided Parameters
Valid Params
Send Email to provided email address
User accesses email and clicks on link
API request
Signup Process
Link redirects to Frontend
Password Reset Process
Frontend makes API request to complete process using Verification Token
User inputs Password

Figure 4: Flow of Email Verification

As shown in the diagram above, this process starts off by verifying the given parameters with the database – ensuring that for signups, provided email and/or username is not existent and for password reset, provided email is existent. When such a request is complete, a signed verification token containing the provided details will be embedded in an URL and sent via a pre-composed e-mail to the provided email account and notify the user to access it. An example of such an email is shown in the figures below.



Welcome to PeerPrep! Inbox ×

peerprepgroup18@gmail.com
to me

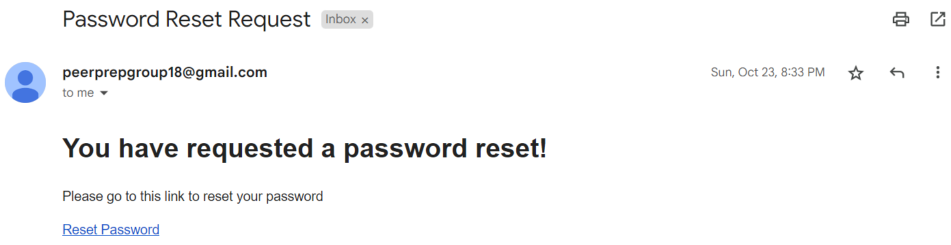
Mon, Nov 7, 4:43 PM (19 hours ago)

Thank you for signing up for Peerprep!

Please go to this link to complete your signup!

Complete Signup

Figure 5: Signup Completion email with embedded URL link



Password Reset Request

peerprepgroup18@gmail.com

to me

Sun, Oct 23, 8:33 PM

You have requested a password reset!

Please go to this link to reset your password

Reset Password

Figure 6: Password Request Completion email with embedded URL link

After the user manages to access this email message and clicks on this link, he will be redirected to a page on the frontend service. If the user was attempting to signup, the frontend will automatically make a request to the user service to create a user using the embedded verification token as a form of authentication to complete the request.
On the other hand, if a user was attempting to reset his password, the link will take him to a page where the user will be prompted to type and send his updated password. This will make a request to update a user's password, also using the embedded verification token as a form of authentication to complete the request.

6.1.4.3 Token Generation

To ensure that protected API Calls can run **securely**, we have used JWT tokens as the mode of authorization. For such calls to run successfully, these tokens will need to be passed through the authorization header of the request (eg. Bearer <token_value>). These tokens are signed by the server, making it difficult for a user who has illegally obtained such a token to modify it.

Since an authentication token should bear necessary information about the user, we have decided to implement the token generation in the User service. When requested, a token is created which includes the required user information, the duration of the token validity and signed with a particular signature depending on token type (shown in the table below).

Token Type	**API Generated In**	**Purpose**
Verification	● Signup Request (After provided credentials are verified in database) ● Password Reset Request (After email	For API Calls where the user is not authenticated but makes certain types of API calls requiring some form of authorization in the future, such as email based verification requests. Currently set to expire 15 minutes after generation to give

	is verified in database)	sufficient time to make one email-based verification request.
Refresh	- Authentication (After successful verification of credentials)	To act as an identifier for an authenticated user when a request for an access token is requested. Currently set to expire 12 hours after generation to provide a balance of sufficient **availability** for the user to utilize PeerPrep entirely and ensuring **security** by ensuring that the user needs to re-authenticate after the expiry of this duration so that the token will be at less risk of being abused if fallen into the hands of malicious users.
Access	- Authentication (After successful verification of credentials) - Get Access (After successful validation of provided Bearer Token)	To act as an identifier for an authenticated user when making an API Call that requires the user to be authorized. Currently set to expire 30 minutes after generation to prevent prolonged abuse of token if it has landed in the possession of unauthorized users.

6.1.4.4 Collection Schema

The following table depicts the collection schema for the User microservice.

Field Name	**Field Property**
- email	- Type: String - Required - Unique
- username	- Type: String - Required - Unique
- password	- Type: String - Required

6.2 Collaboration Service

The responsibility of the collaboration service is to match users together and allow matched users to work together on the same question. It provides real time code collaboration, chatting and video calling features between users.

6.2.1 Functional Requirements

S/N	Requirements	Priority
FR2.1	The system should allow users to select the difficulty level of the questions they wish to attempt	High
FR2.2	The system should be able to match two waiting users with similar difficulty levels and put them in the same room.	High
FR2.3	If there is a valid match, the system should match the users within 30s.	High
FR2.4	The system should inform the users that no match is available if a match cannot be found within 30 seconds.	High
FR2.5	The system should allow the user to end the session.	Medium
FR2.6	The system should provide a mechanism for real-time collaboration (e.g., concurrent code editing) between the participants in the room.	High
FR2.7	Provides chatting among the participants in the room, once the users have been matched.	Medium
FR2.8	Provide voice and video calling among participants in the room	Medium

6.2.2 Non-Functional Requirements

S/N	Category	Requirements	Priority
NFR2.1	Performance	Concurrent code editing should have a response time of less than 1 second	High
NFR2.2	Performance	Voice and video calling should have a response time of less than 1 second	Medium

6.2.3 Services

Interface	Service description
1. Match Request	If there is a match request of the same difficulty, put the two users in the same room. If there is no match requested now, wait for at most 30s for another user to request a match.
2. Get Host Peer ID	Emits the host peer ID used for video chatting.
3. Chat Message	Emits the message to the partner.
4. Code Editor	Emits the code to the partner.
5. Partner Rating	Emits the rating to the partner.
6. Check room existence	Check if the user still belongs to the room.
7. End Session	Emits all both partner's socket ID and informs both partners to end their session.
8. Disconnect	Emit a disconnect message to the partner.

6.2.4 Implementation

This section describes how the Collaboration microservice is implemented.

6.2.4.1 Matching Feature Implementation

The diagram below shows the sequence diagram of the matching feature. The user first emits a match request. The user either waits for another user to match him/her or joins a waiting user.

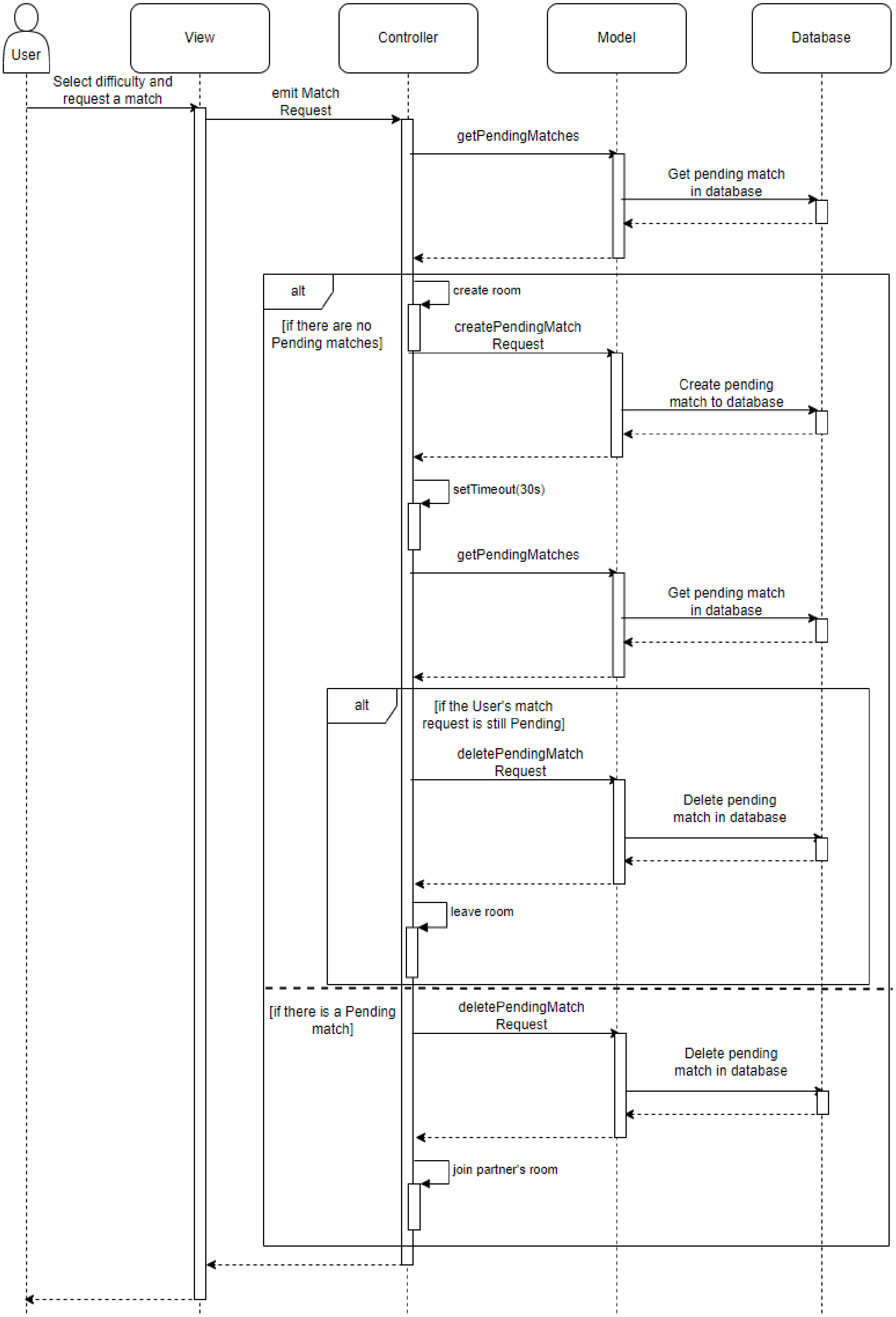

User
View
Controller
Model
Database
Select difficulty and request a match
emit Match Request
getPendingMatches
Get pending match in database
alt
create room
[if there are no Pending matches]
createPendingMatch Request
Create pending match to database
setTimeout(30s)
getPendingMatches
Get pending match in database
alt
[if the User's match request is still Pending]
deletePendingMatch Request
Delete pending match in database
leave room
[if there is a Pending match]
deletePendingMatch Request
Delete pending match in database
join partner's room

Figure 7: Sequence Diagram of the matching feature

6.2.4.2 Messaging Architecture

This service uses the **Publish-Subscribe messaging pattern** with SocketIO. It is used for the entire collaboration system, including chatting, video chatting and code collaboration. This design choice was due to the nature of the interview session, where the server must broadcast the same messages to the clients that are collaborating.

The diagram below shows the architecture diagram of the message pattern.

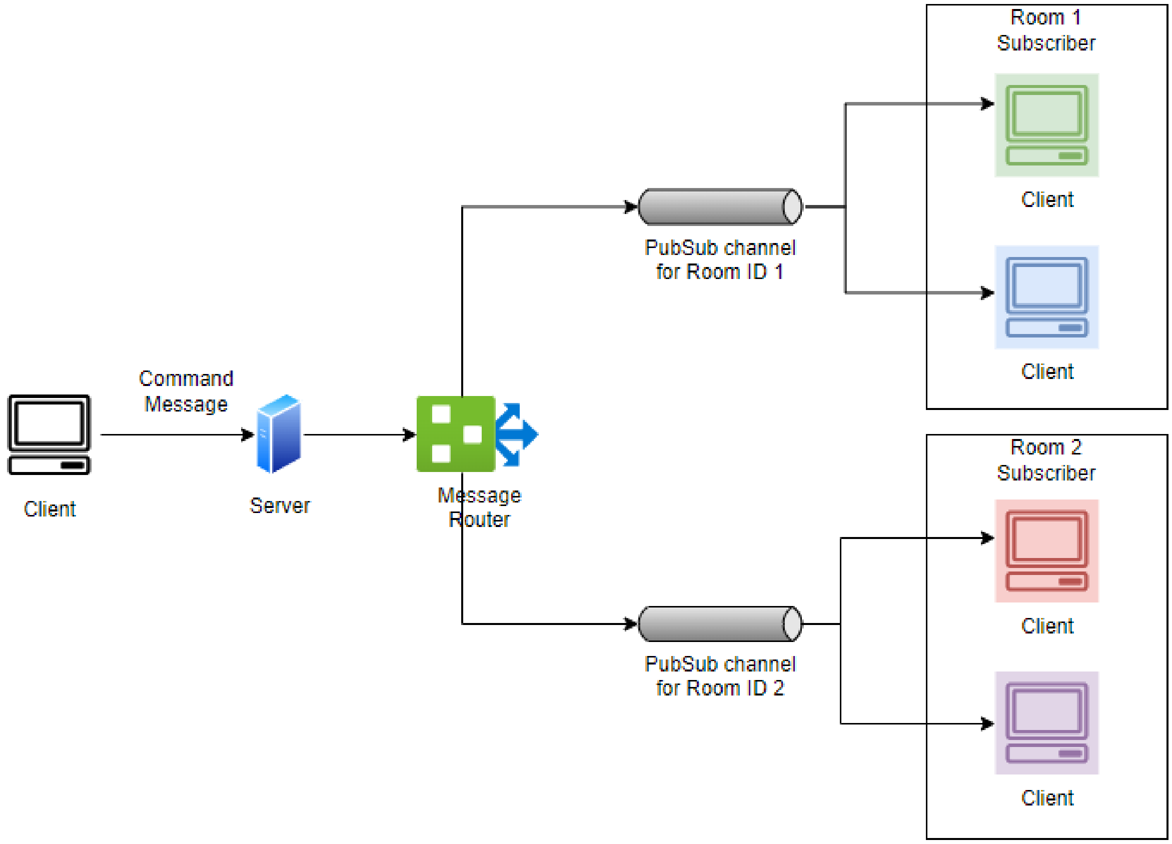


Figure 8: Messaging pattern Architecture Diagram

An example of the message pattern:
A Client emits a Command Message to the server, eg. Chat Message (refer to Services). The Server parses the Command Message, and emits the message to the respective room's Message Channel that the Client belongs in. Both partners are on the same Message Channel and is subscribed to the "Chat Message" topic. Both partner receives the message.

6.2.4.3 Collection Schema

The following table depicts the collection schema for the Collaboration microservice.

Field Name	Field Property
● Name	● Type: String ● Required ● Unique
● Difficulty	● Type: String ● Required
● SocketId	● Type: String ● Required ● Unique

6.3 Question Service

The responsibility of the Question Service is to supply the questions needed for PeerPrep main functionality of allowing users to collaborate together on coding questions. The question service play this part by first filling up a question bank by scraping questions from Leetcode, and then providing the necessary APIs to retrieve questions from the question bank.

6.3.1 Functional Requirements

S/N	Requirements	Priority
FR3.1	The system should store questions of various types (eg. difficulty, topic)	High
FR3.2	The system should retrieve an appropriate and random question for the users in a particular room	High
FR3.3	The system should allow the user to select questions based on difficulty	High
FR3.4	The system should allow the user to select questions based on topics	Very low

6.3.2 Non-Functional Requirements

S/N	Category	Requirements	Priority
NFR3.1	Usability	Question retrieved should be displayed in rich text format for better comprehension of the question by the user	Medium
NFR3.2	Performance	Question fetching from the database when a match between 2 users is made should be completed in less than 2 seconds	High

6.3.3 APIs

Request	API Route	Description
GET	/api/questions/	Gets a random question of a specified difficulty (passed via a query parameter) from our question bank
GET	/api/questions/:question_id	Gets a specific question using its ID
GET	/api/questions/health	Checks the operational status of the service efficiently

6.3.4 Implementation

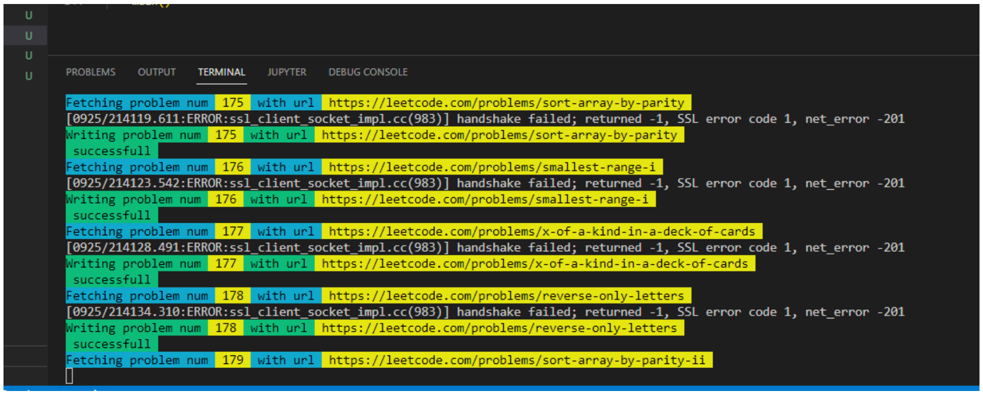
This section describes how the Question microservice is implemented.

6.3.4.1 Scraping Questions from Leetcode

Firstly, to populate questions into our question bank, which is a MongoDB collection, we have written a python program that does the following steps sequentially:

1) Grabs a minimum set of important fields of all the questions, such as question title, from an internal API of Leetcode
2) Using the question title information extracted from step 1, we generated the URL for each question
3) Using Selenium WebDriver and Chromedriver, we accessed each question one by one automatically
4) For every question accessed, we used Beautiful Soup (a python package) to scrape the necessary information according to our question model
5) Every time relevant information for a particular question is scraped, the data will be inserted into our question bank

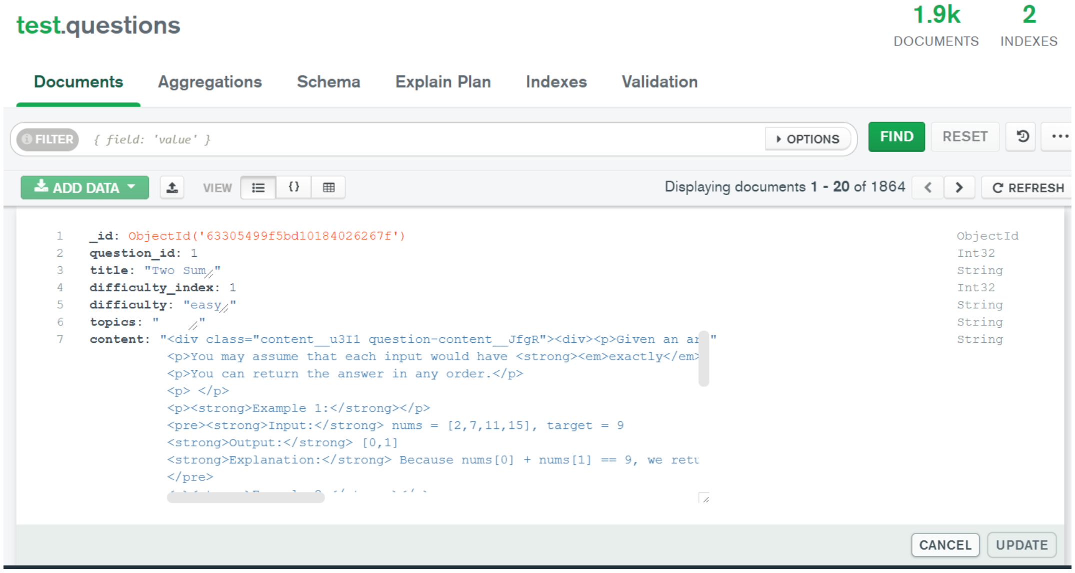
Figure 9 below depicts how the python program looks like when it is in action:



```
Fetching problem num  175  with url  https://leetcode.com/problems/sort-array-by-parity
[0925/214119.611:ERROR:ssl_client_socket_impl.cc(983)] handshake failed; returned -1, SSL error code 1, net_error -201
Writing problem num  175  with url  https://leetcode.com/problems/sort-array-by-parity
 successfull
Fetching problem num  176  with url  https://leetcode.com/problems/smallest-range-i
[0925/214123.542:ERROR:ssl_client_socket_impl.cc(983)] handshake failed; returned -1, SSL error code 1, net_error -201
Writing problem num  176  with url  https://leetcode.com/problems/smallest-range-i
 successfull
Fetching problem num  177  with url  https://leetcode.com/problems/x-of-a-kind-in-a-deck-of-cards
[0925/214128.491:ERROR:ssl_client_socket_impl.cc(983)] handshake failed; returned -1, SSL error code 1, net_error -201
Writing problem num  177  with url  https://leetcode.com/problems/x-of-a-kind-in-a-deck-of-cards
 successfull
Fetching problem num  178  with url  https://leetcode.com/problems/reverse-only-letters
[0925/214134.310:ERROR:ssl_client_socket_impl.cc(983)] handshake failed; returned -1, SSL error code 1, net_error -201
Writing problem num  178  with url  https://leetcode.com/problems/reverse-only-letters
 successfull
Fetching problem num  179  with url  https://leetcode.com/problems/sort-array-by-parity-ii
```

Figure 9: Python web scraping program in action

Using this method, a total of 1864 questions consisting of easy, medium and hard questions have been scraped and inserted into our question bank.

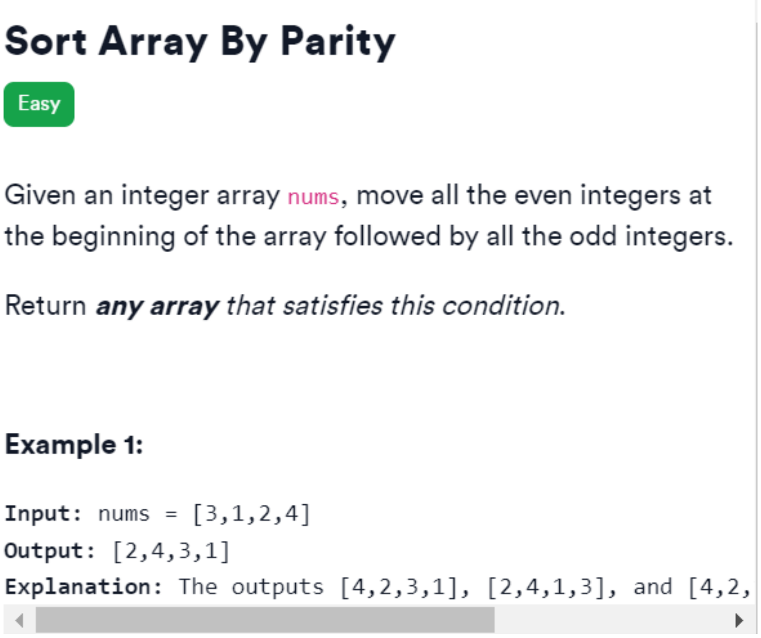


Displaying documents 1 - 20 of 1864

```
_id: ObjectId('63305499f5bd10184026267f')                                    ObjectId
question_id: 1                                                               Int32
title: "Two Sum"                                                             String
difficulty_index: 1                                                          Int32
difficulty: "easy"                                                           String
topics: "    "                                                               String
content: "<div class="content__u3I1 question-content__JfgR"><div><p>Given an ar "  String
          <p>You may assume that each input would have <strong><em>exactly</em>
          <p>You can return the answer in any order.</p>
          <p> </p>
          <p><strong>Example 1:</strong></p>
          <pre><strong>Input:</strong> nums = [2,7,11,15], target = 9
          <strong>Output:</strong> [0,1]
          <strong>Explanation:</strong> Because nums[0] + nums[1] == 9, we retu
          </pre>
```

Figure 10: A question in our question bank

As seen from the figure above, we have kept the html formatting of the content of the question when we inserted it into our question bank. Using this formatting, our frontend could easily display the questions in rich text format for better comprehension of the question by the user, as seen figure 11 below.



Sort Array By Parity

Easy

Given an integer array `nums`, move all the even integers at the beginning of the array followed by all the odd integers.

Return ***any array*** *that satisfies this condition.*

Example 1:

```
Input: nums = [3,1,2,4]
Output: [2,4,3,1]
Explanation: The outputs [4,2,3,1], [2,4,1,3], and [4,2,
```

Figure 11: Example of a question displayed to a user

6.3.4.2 Retrieving questions from our question bank

The diagram below shows the sequence diagram of the question feature whenever a user wants to attempt a question in PeerPrep. To demonstrate how a question is retrieved from the question bank (called Database in the diagram), in this sequence diagram we assume the case of there being no pending match request and the user is initiating one.

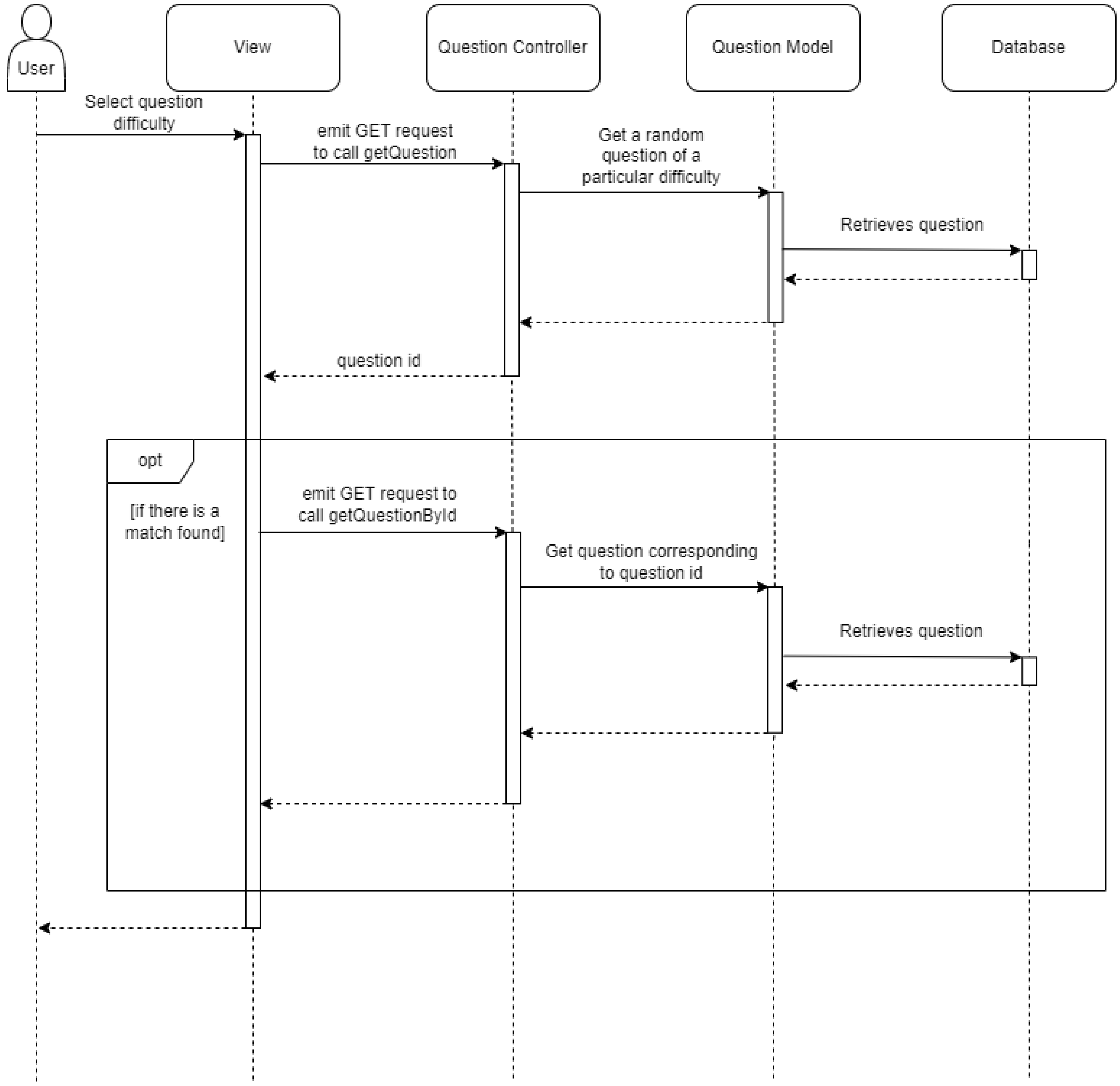

User
View
Question Controller
Question Model
Database
Select question difficulty
emit GET request to call getQuestion
Get a random question of a particular difficulty
Retrieves question
question id
opt
[if there is a match found]
emit GET request to call getQuestionById
Get question corresponding to question id
Retrieves question

Figure 12: Sequence Diagram of the question feature

The reason why the Question Controller returns a question id when the user initiated a match is so that the other matched user can then also get a reference of this question id and retrieve the same question from the Database. This is possible as we used the socket in the Collaboration Service to emit the same question id to both users when they have matched. As such, this synchronization of question id allows both users to retrieve and display the same question as they collaborate with one another in the interview room.
On the other hand, from the perspective of a user matching with another user who is already waiting for a match, this user will only have to perform the operations as seen in the optional path of the sequence diagram in Figure 8, whereby only a specific problem of appropriate difficulty is retrieved and displayed to the user in the interview room.

6.3.4.3 Collection Schema

The following table depicts the collection schema for the Question microservice.

Field Name	**Field Property**
● QuestionId	● Type: Number ● Required ● Unique
● Title	● Type: String ● Required
● DifficultyIndex	● Type: Number ● Required
● Difficulty	● Type: String ● Required
● Content	● Type: String

6.4 User Interface

The responsibility of the User Interface service is to display the application's functionalities to the user in a user-friendly and aesthetically pleasing way.

6.4.1 Functional Requirements

S/N	Requirements	Priority
FR4.1	Functional user interface that guides the user to using the features of the app	High
FR4.2	Responsive user interface that enables users to practice from their mobile phone.	Medium
FR4.3	The system can allow users to craft a solution in a supported language of their choice.	Medium-Low

6.4.2 Non-Functional Requirements

S/N	Category	Requirements	Priority
NFR4.1	Usability	Optimize for usability through well-designed and user-friendly user interface	Medium
NFR4.2	Usability / Compatibility	Enable system to be usable from a user's mobile phone, such that they are able to use the app on the go	Medium
NFR4.3	Security	The system should encrypt stored cookies to prevent them from being sniffed by unauthorized users.	Low
NFR4.4	Security	The system should validate a given token's expiry to determine a user's ability to access its features	Medium
NFR4.5	Usability	The system should have a 404 page in the case where user tries to access an invalid url	Low

6.4.3 Implementation

The frontend consists of the presentation layer. ReactJS was the framework of choice for developing the user interface for its modularity, simplicity and performance. We also used **customized Bootstrap components** as the CSS framework and to ease implementation of the responsiveness of the interface.

Each component and page have their own functionality and may call HTTP API requests to fetch or manipulate data in the various microservices. These API requests are made to the application layer.

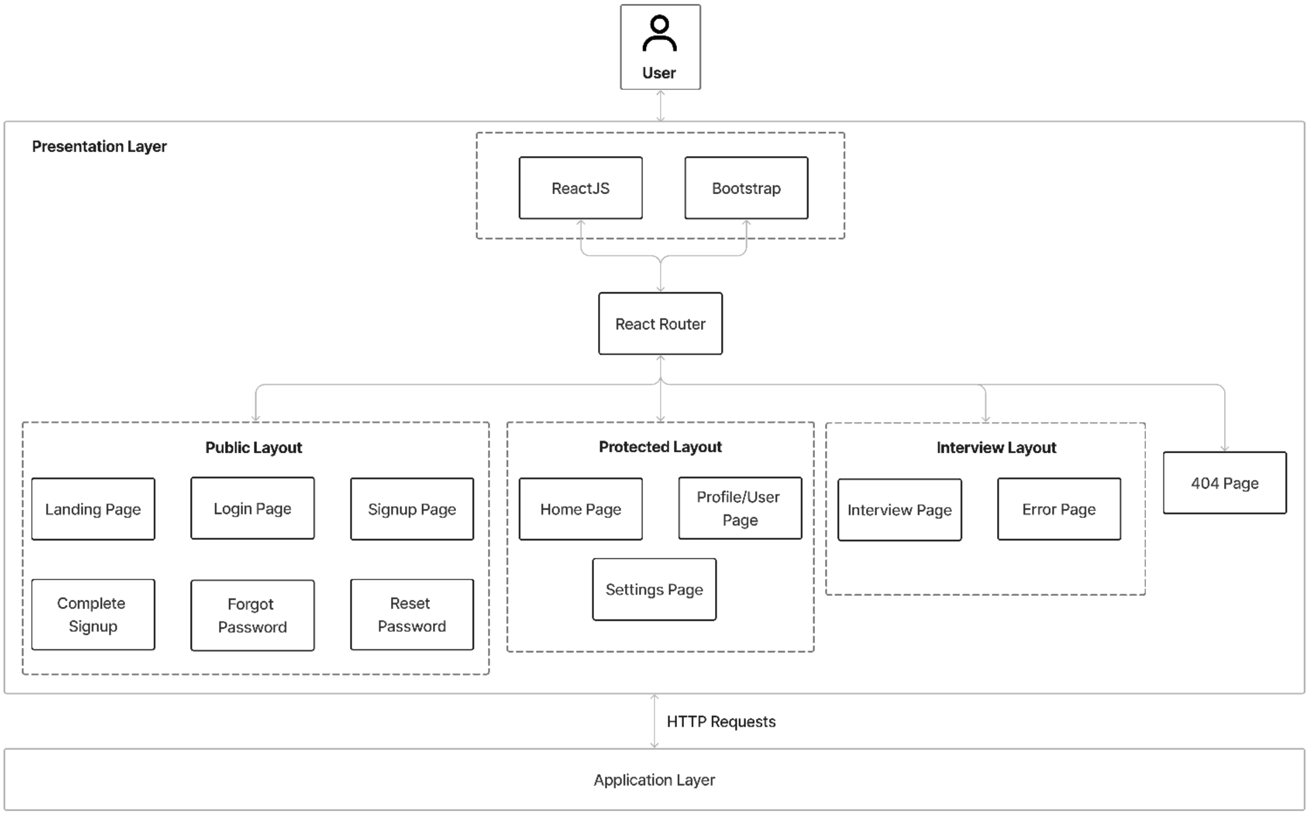
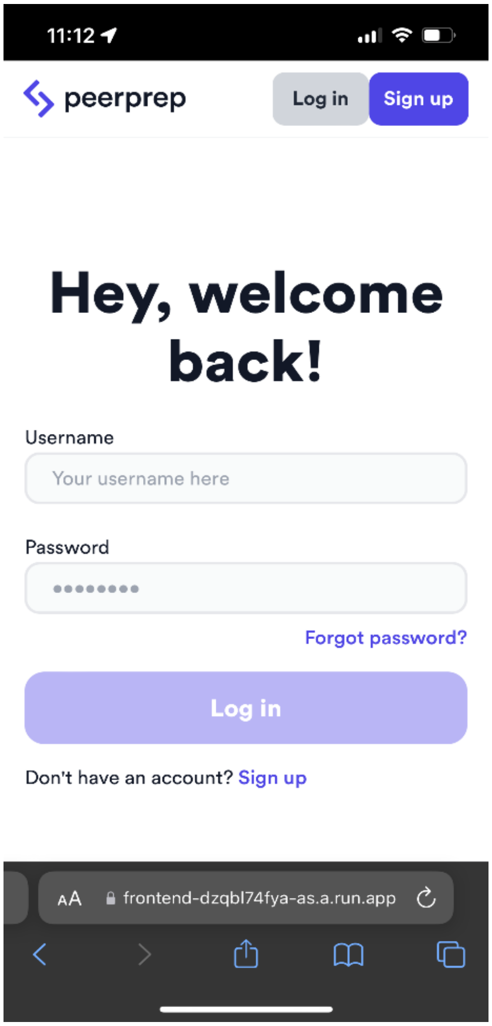


Figure 13: Frontend architecture

In addition, to satisfy the NFR of **Usability** and **Compatibility**, we also ensured that the **entire application is also mobile responsive**. Due to the ubiquity of mobile devices, we thought it would be beneficial for users to be able to practice with their friends even when on the go.



Hey, welcome back!

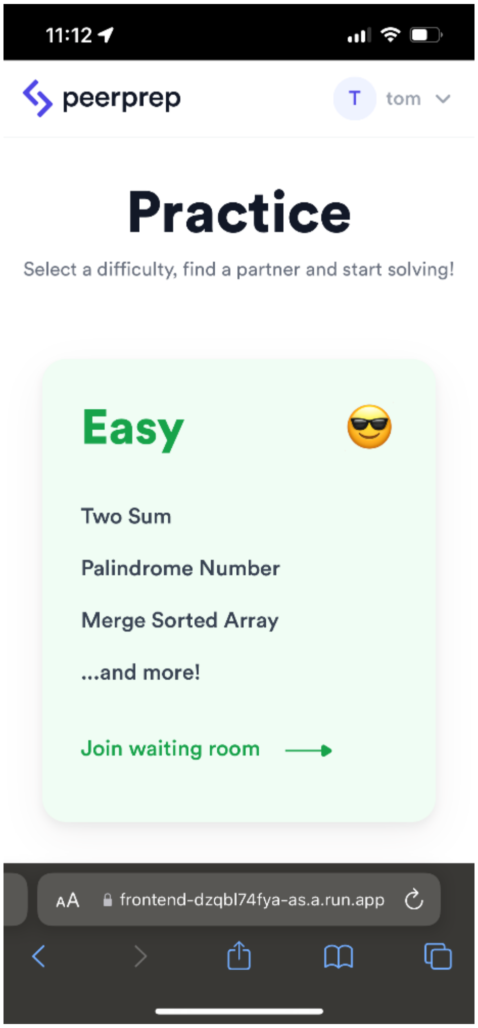
Username

Your username here

Password

••••••••

Don't have an account? Sign up



Practice

Select a difficulty, find a partner and start solving!

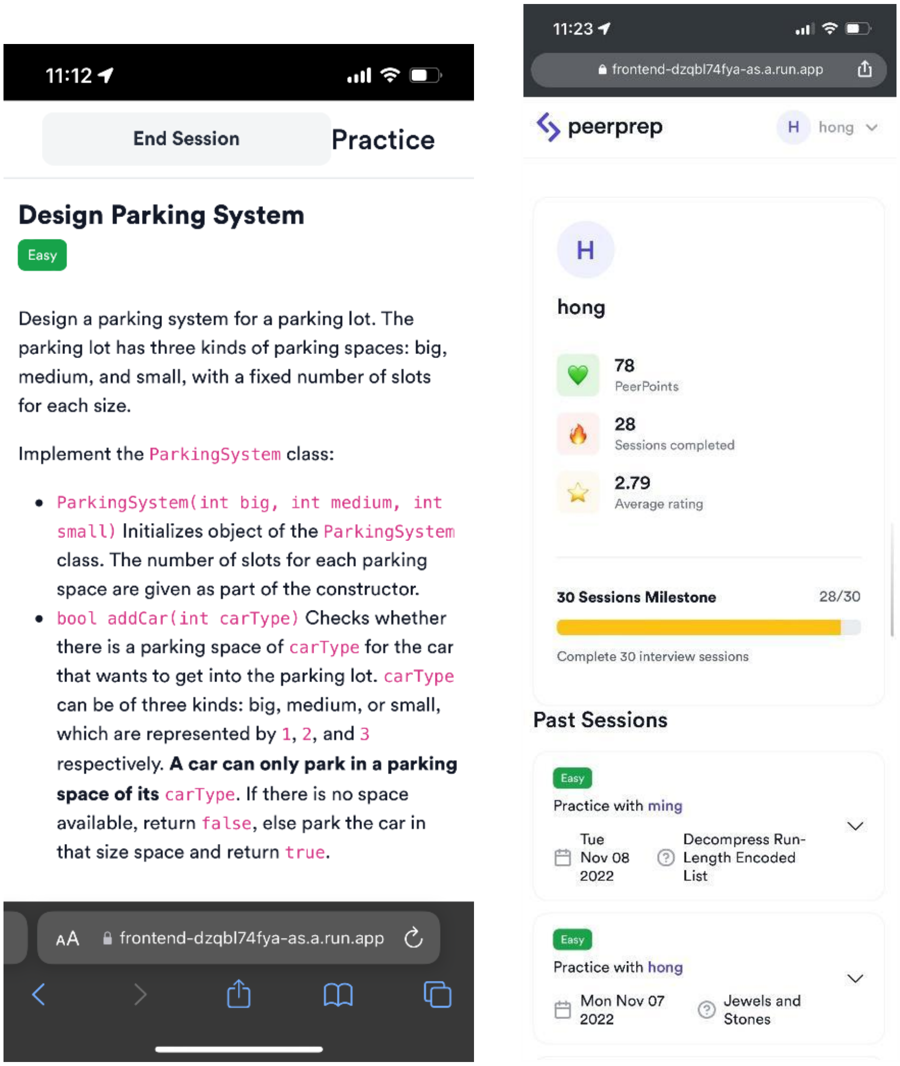
Easy 😎

Two Sum

Palindrome Number

Merge Sorted Array

...and more!



11:12

End Session

Practice

Design Parking System

Easy

Design a parking system for a parking lot. The parking lot has three kinds of parking spaces: big, medium, and small, with a fixed number of slots for each size.

Implement the `ParkingSystem` class:

- `ParkingSystem(int big, int medium, int small)` Initializes object of the `ParkingSystem` class. The number of slots for each parking space are given as part of the constructor.
- `bool addCar(int carType)` Checks whether there is a parking space of `carType` for the car that wants to get into the parking lot. `carType` can be of three kinds: big, medium, or small, which are represented by `1`, `2`, and `3` respectively. **A car can only park in a parking space of its** `carType`. If there is no space available, return `false`, else park the car in that size space and return `true`.

AA frontend-dzqbl74fya-as.a.run.app

11:23

frontend-dzqbl74fya-as.a.run.app

peerprep H hong

H

hong

78 PeerPoints

28 Sessions completed

2.79 Average rating

30 Sessions Milestone 28/30

Complete 30 interview sessions

Past Sessions

Easy

Practice with ming

Tue Nov 08 2022 Decompress Run-Length Encoded List

Easy

Practice with hong

Mon Nov 07 2022 Jewels and Stones

Figure 14: Screenshots of our app working on mobile

6.4.3.1 Layouts and protected routes

We have also implemented different layouts within the app to ensure security of protected pages. For example, pages within the Protected Layout are not accessible to unauthenticated users, and users that are authenticated should not have to see pages within the Public Layout and have to log in every single time if their refresh token has not expired.

Below is a summary of the various pages within the app and the layouts they are nested within:

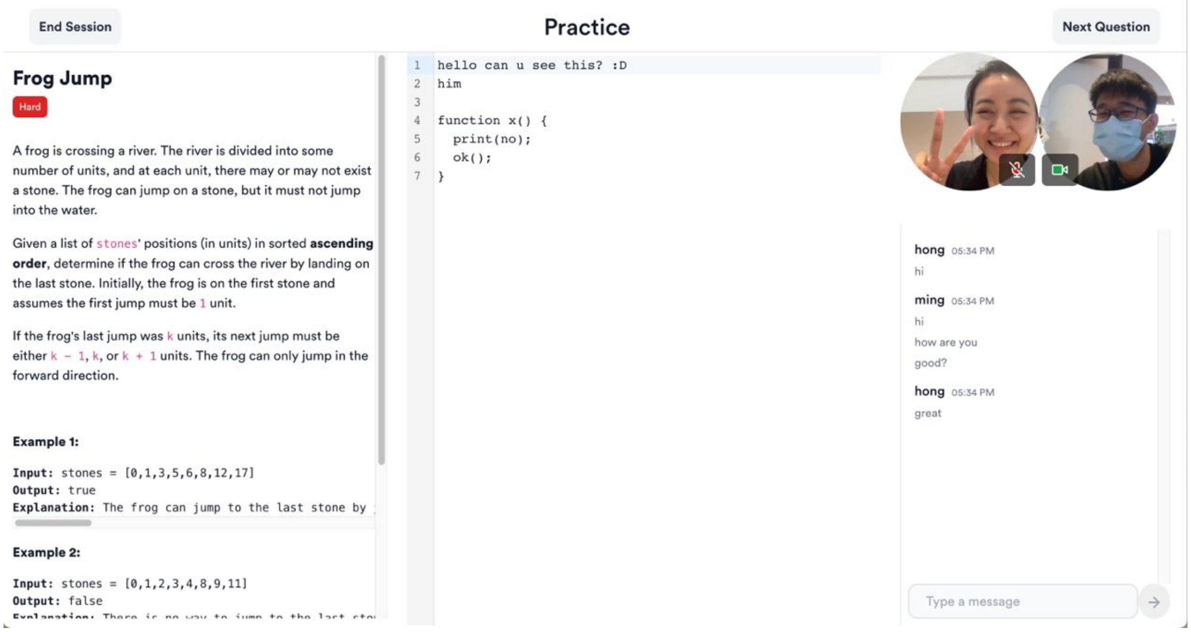
Page	Layout	Page url
PublicLayout	LandingPage	/
	LoginPage	/login
	SignupPage	/signup
	CompleteSignup	/completeSignup
	ForgotPassword	/forgotPassword
	ResetPassword	/passwordReset
ProtectedLayout	HomePage	/home
	ProfilePage	/profile/:username
	SettingsPage	/settings

For example, in the event an unauthenticated user tries to access a page within the protected layout, they will be automatically redirected to the login page, prompting them to log in.

Conversely, if an already logged in user tries to access a page within the public layout, they will be automatically redirected to the home page (where they can select difficulty).

6.4.3.2 Video Calling

The video calling within the Interview room is implemented using PeerJS and SocketIO. Broadly speaking, the socket is used for the “host user” to transmit his peer ID to the other user, allowing the other user to make use of the PeerJS API to call him.



Frog Jump

A frog is crossing a river. The river is divided into some number of units, and at each unit, there may or may not exist a stone. The frog can jump on a stone, but it must not jump into the water.

Given a list of stones' positions (in units) in sorted **ascending order**, determine if the frog can cross the river by landing on the last stone. Initially, the frog is on the first stone and assumes the first jump must be 1 unit.

If the frog's last jump was k units, its next jump must be either k - 1, k, or k + 1 units. The frog can only jump in the forward direction.

Example 1:

```
Input: stones = [0,1,3,5,6,8,12,17]
Output: true
Explanation: The frog can jump to the last stone by
```

Example 2:

```
Input: stones = [0,1,2,3,4,8,9,11]
Output: false
Explanation: There is no way to jump to the last sto
```

```
hello can u see this? :D
him

function x() {
  print(no);
  ok();
}
```

hong 05:34 PM
hi

ming 05:34 PM
hi
how are you
good?

hong 05:34 PM
great

Figure 15: Video calling in Interview room

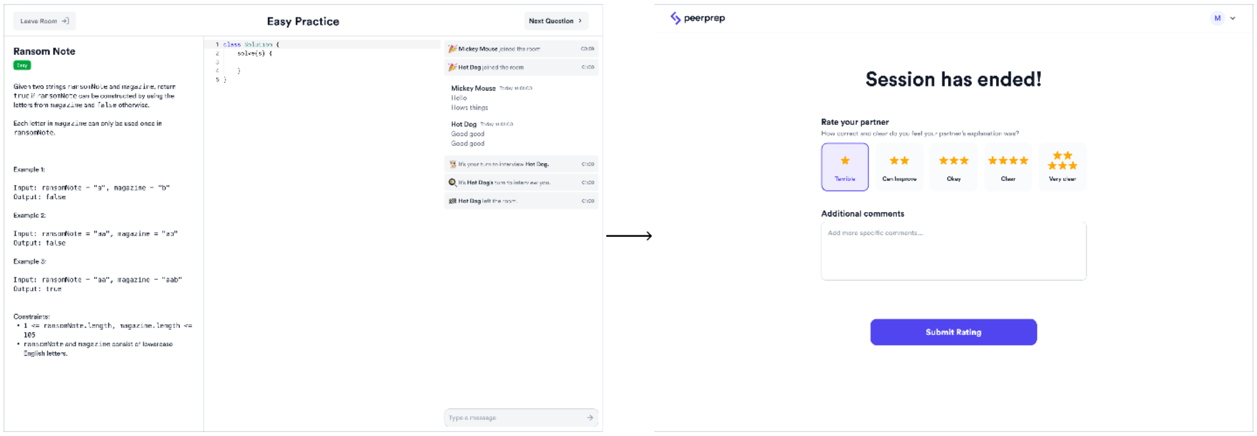
In addition, we opted to make the video calling as non-obtrusive as possible, and allowing for functions such as muting oneself and switching off one’s camera.

6.4.3.3 Error Handling in the Interview Room

Refer to Section 6.5.4.1 for the activity diagram of how a user gives feedback for their partner. The left path in the diagram shows a high-level error handling flow for when someone disconnects during the coding session.

Happy Path:

When one user clicks on the button to end the session, he will be warned that this will end the session for everyone. If he accepts, both users will be brought to the feedback page showing them that the session has ended and prompting them to leave each other some feedback on the session.



Ransom Note

Easy

Given two strings ransomNote and magazine, return true if ransomNote can be constructed by using the letters from magazine and false otherwise.

Each letter in magazine can only be used once in ransomNote.

Example 1:

```
Input: ransomNote = "a", magazine = "b"
Output: false
```

Example 2:

```
Input: ransomNote = "aa", magazine = "ab"
Output: false
```

Example 3:

```
Input: ransomNote = "aa", magazine = "aab"
Output: true
```

Constraints:
- 1 <= ransomNote.length, magazine.length <= 105
- ransomNote and magazine consist of lowercase English letters.

```
class Solution {
    solve(s) {

    }
}
```

Mickey Mouse joined the room 00:59

Hot Dog joined the room 01:00

Mickey Mouse Today at 01:00
Hello
Hows things

Hot Dog Today at 01:00
Good good
Good good

It's your turn to interview Hot Dog. 01:00

It's Hot Dog's turn to interview you. 01:00

Hot Dog left the room. 01:00

Session has ended!

Rate your partner

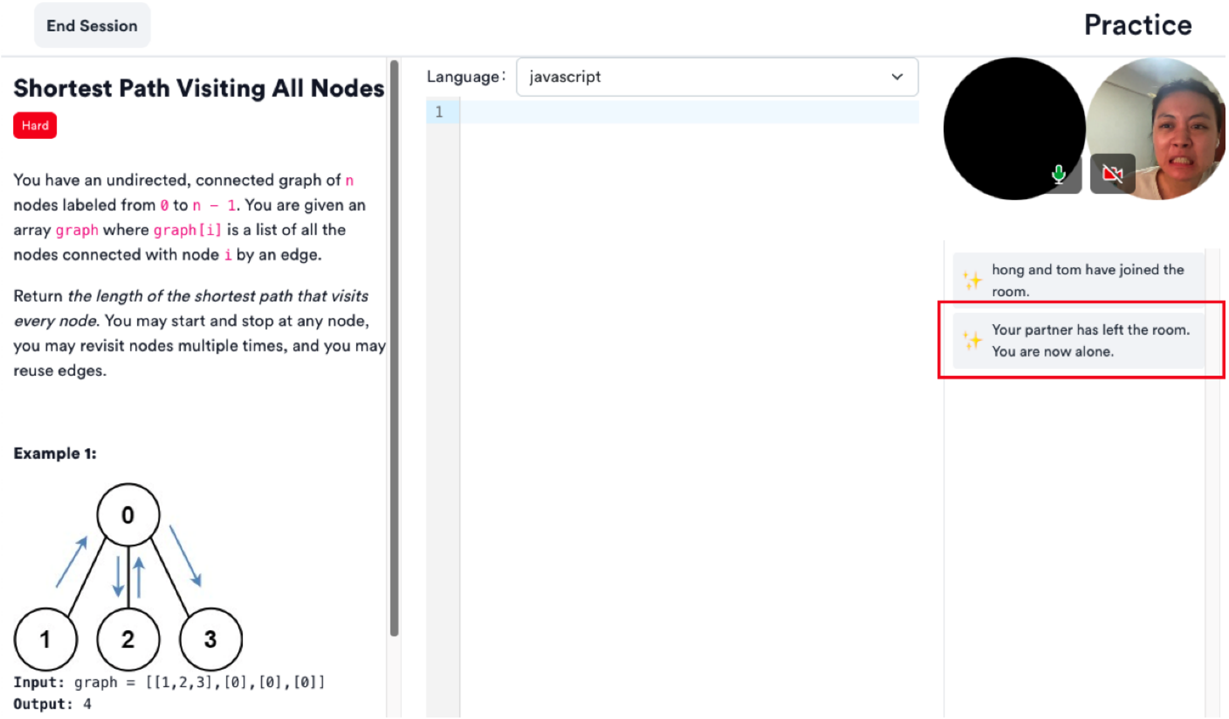
How correct and clear do you feel your partner's explanation was?

Additional comments

Figure 16: Happy path where users can leave feedback after the session

If partner quits midway through the interview:

A notification is sent to the remaining user that their partner has left. However, they are still able to continue practicing alone, albeit **without receiving any feedback at the end**. They will thus be routed directly back to the home page after ending the session instead of the feedback page.



Shortest Path Visiting All Nodes

Hard

You have an undirected, connected graph of n nodes labeled from 0 to n - 1. You are given an array graph where graph[i] is a list of all the nodes connected with node i by an edge.

Return *the length of the shortest path that visits every node*. You may start and stop at any node, you may revisit nodes multiple times, and you may reuse edges.

Example 1:

```
Input: graph = [[1,2,3],[0],[0],[0]]
Output: 4
```

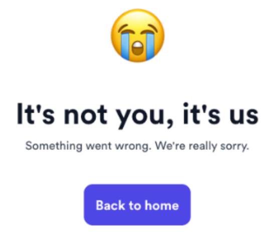
Language: javascript

hong and tom have joined the room.

Your partner has left the room. You are now alone.

Figure 17: Notification is sent if partner disconnects

If partner disconnects on the feedback page:
Currently, our implementation requires that both users submit their feedback for both ends to receive it. As such, if one user disconnects midway, the remaining user will be routed to an error page.



It's not you, it's us

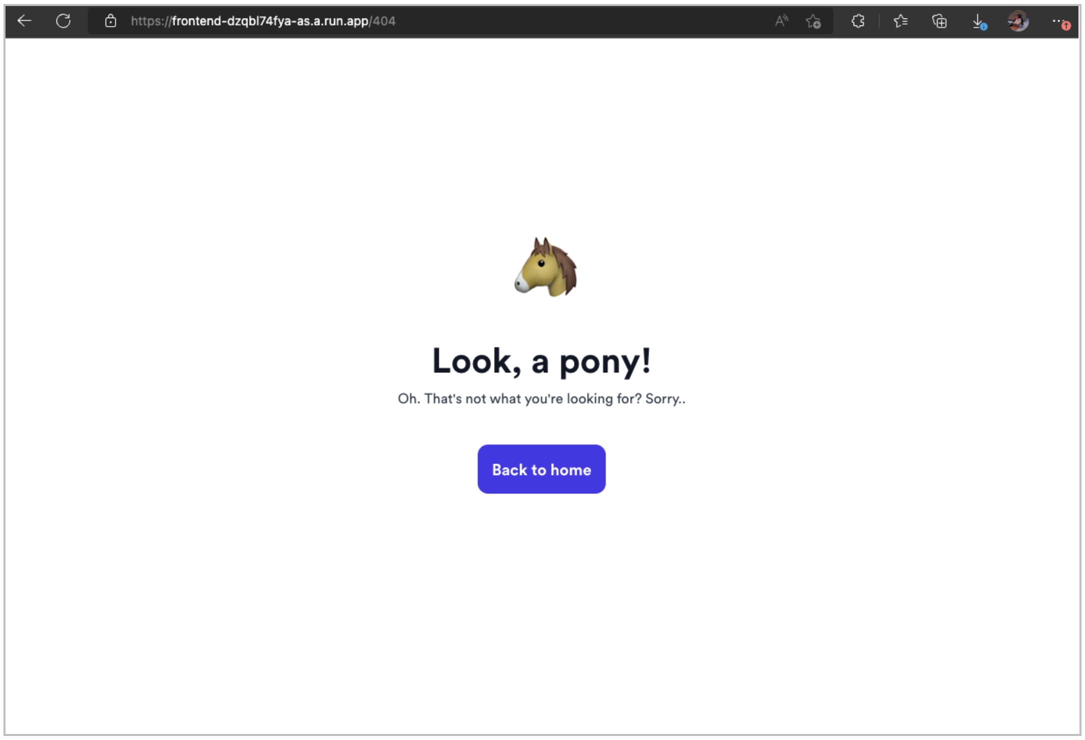
Something went wrong. We're really sorry.

Figure 18: Error page

We acknowledge this is not the most ideal implementation due to the limited time frame; however should we have had more time we would like to implement this asynchronously instead, in other words either user may submit feedback at any time without worrying about disconnection since the feedback would be received asynchronously.

6.4.3.4 Error and 404 Pages

To ensure a better user experience in terms of handling errors, we also implemented error pages and 404 pages, which we would route the user to in the event of an error or a 404 (e.g. user tries to view the profile of another user who doesn't exist, user types in an invalid url).



https://frontend-dzqbl74fya-as.a.run.app/404

Look, a pony!

Oh. That's not what you're looking for? Sorry..

Figure 19: User is routed to 404 page when they try to access an invalid url

6.5 User History Service

The responsibility of the User History is to allow users to see the history of questions they have attempted and view the feedback they received from their partners. Till this end, the service provides the relevant APIs for adding a user history record for a particular user into our database, and for retrieving such user history records to provide profile page for them.

6.5.1 Functional Requirements

S/N	Requirements	Priority
FR5.1	Maintains a record of the past attempts e.g., difficulty levels or questions attempted.	Medium
FR5.2	Allow users to rate and give feedback to their practice partner based on the practice session they had	Low
FR5.3	Scoring system based on rating received	Low
FR5.4	Award users with PeerPoints whenever they complete questions and reach certain milestones	Low

6.5.2 Non-Functional Requirements

S/N	Category	Requirements	Priority
NFR3.1	Usability	Provides an aesthetically pleasing frontend for users to intuitively and easily view their past activities and achievements in PeerPrep	Medium

6.5.3 APIs

Request	API Route	Description
GET	/api/user-history/	Gets all user histories from all users
POST	/api/user-history/	Inserts a user history into our collection
PATCH	/api/user-history/	Updates an existing user history
DELETE	/api/user-history/	Deletes an existing user history
GET	/api/user-history/:username	Gets the user histories of a particular user
GET	/api/user-history/health	Checks the operational status of the service efficiently

6.5.4 Implementation

This section describes how the User History microservice is implemented.

6.5.4.1 Adding a User History record into the Collection

The diagram below shows the activity diagram of the User History feature to demonstrate the actions and control flows that make up the activity of a user giving feedback to their partner once their collaboration session has ended. A feedback from the partner forms a user history record for the users.

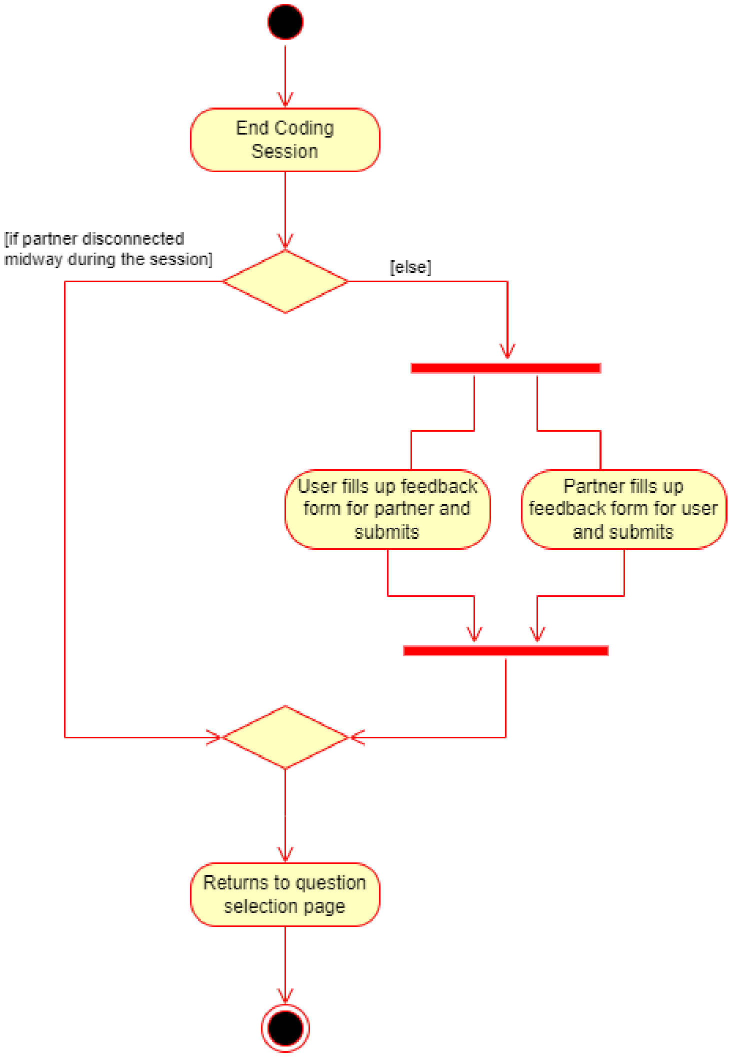

End Coding Session
[if partner disconnected midway during the session]
[else]
User fills up feedback form for partner and submits
Partner fills up feedback form for user and submits
Returns to question selection page

Figure 20: Activity diagram of the User History feature

As seen in the left path of the activity diagram above, a user will neither give nor receive feedback for the session if their partner has already disconnected midway during the session. This is because we believe that feedback is valuable in PeerPrep as it not only serves to give a history record of the question attempted and who their partner was, but it also serves as an indication to how well the user performed for the question so that they can keep track of their progress. Hence, in order to avoid complicated cases such as only one of the users (the disconnected one in this case) receives a feedback or the case of users receiving a bad feedback because they disconnected, we decided to void the feedback function completely whenever any user disconnected midway.

However, while we understand that this design choice might deprive users from keeping a full record of their past attempted questions, especially when the session was almost completed before one party disconnected, we felt that it was a better option to avoid the complicated cases for our current development, considering the short time frame we have. Therefore, if given more time, we will implement a solution to handle the possible cases that may occur during a session. For example, we could possibly allow the feedback giving process to continue for the user who remains in the session. We can also enforce the disconnected user to provide a feedback when they log back on to PeerPrep.

Nonetheless, if both users complete the session together, they will be directed to a feedback form page where they can rate and comment on their partner for how they have performed during the session. The figure below shows how the feedback form looks like.

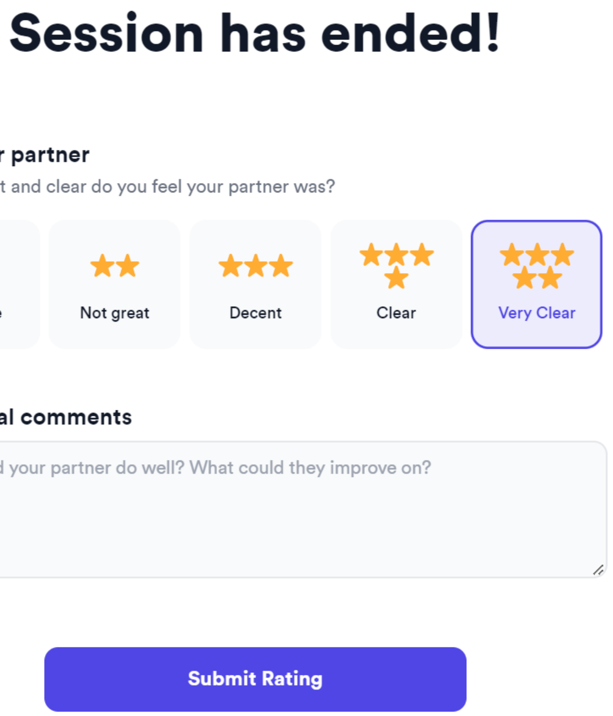
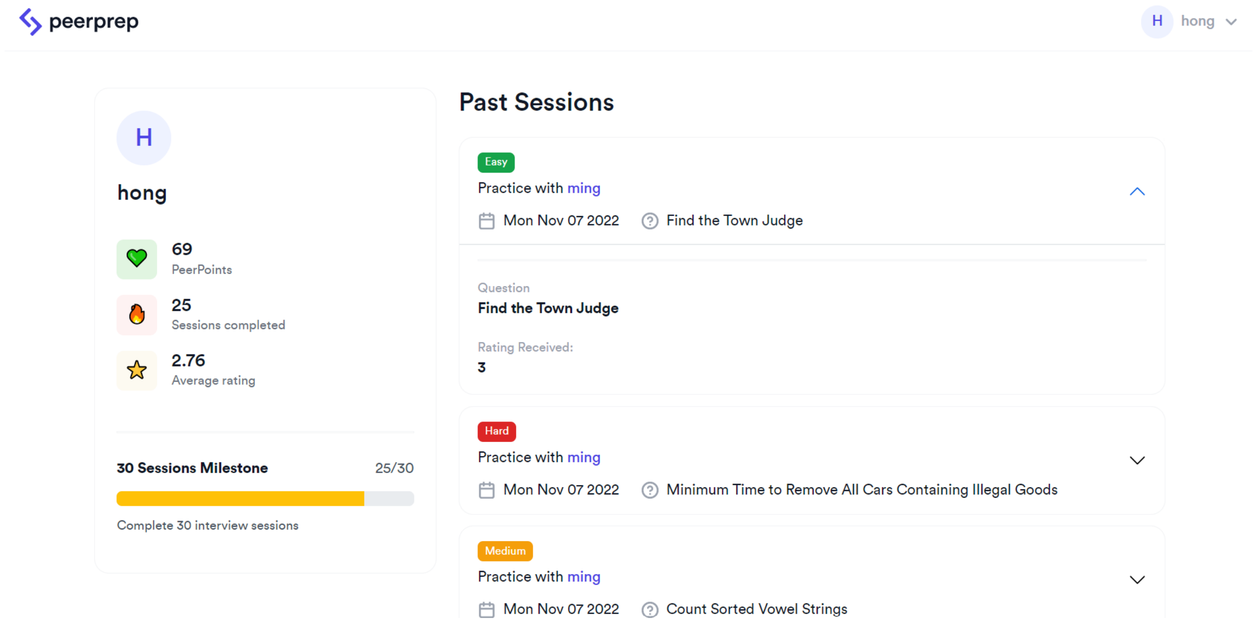

Session has ended!
Rate your partner
How correct and clear do you feel your partner was?
Terrible
Not great
Decent
Clear
Very Clear
Additional comments
What did your partner do well? What could they improve on?
Submit Rating

Figure 21: Feedback form

Once both users have submitted the feedback form, a User History record will then be created, where its fields will be automatically filled with information of the user, the user’s partner, the question they attempted and the feedback they received from their partner. The record will then be inserted into our MongoDB collection.

6.5.4.2 Profile Page

The figure below shows a snapshot of the profile page in PeerPrep.



H

hong

69
PeerPoints

25
Sessions completed

2.76
Average rating

30 Sessions Milestone 25/30

Complete 30 interview sessions

Past Sessions

Easy
Practice with ming
Mon Nov 07 2022 Find the Town Judge

Question
Find the Town Judge

Rating Received:
3

Hard
Practice with ming
Mon Nov 07 2022 Minimum Time to Remove All Cars Containing Illegal Goods

Medium
Practice with ming
Mon Nov 07 2022 Count Sorted Vowel Strings

Figure 22: Profile page

Each past session record in the page corresponds to one user history record in our collection. Through the `/api/user-history/:username` route, whenever a user enters the profile page, user history records specific to the user will be retrieved and used to populate and be displayed to the user via the page. Clicking on any of the record will allow the user to see who their partner was for the session, the question title, the rating and comments (if any) that the partner has given them.

Our frontend also compiles and displays useful statistics, such as the average rating they have received, to the user in an aesthetically pleasing manner. This serves to provide them with a quick overall view of their progress in PeerPrep. We also added a progress bar with incremented milestones for users, to motivate them to practice more. With more time, we would like to introduce more nuanced and strategic milestones, or even achievement badges.

In addition, users are able to view the profiles of other users and reference their past sessions and stats. We felt this would motivate users to practice more in terms of introducing a competitive element.

6.5.4.3 Collection Schema

The following table depicts the collection schema for the User History microservice.

Field Name	**Field Property**
● Username	● Type: String ● Required
● PartnerUsername	● Type: String ● Required
● QuestionId	● Type: Number ● Required
● QuestionDifficultyIndex	● Type: Number ● Required
● QuestionTitle	● Type: String ● Required
● AnswerProvided	● Type: String
● RatingReceived	● Type: Number
● CommentsReceived	● Type: String
● DateTime	● Type: Date

6.6 Authentication Service

The authentication service is responsible for verifying JWT tokens passed together with API requests. It ensures that sensitive APIs are only accessible by users who have an appropriate and valid JWT token that has been issued by the User Service.

6.6.1 Functional Requirements

S/N	Functional Requirements	Priority
FR6.1	The system should check if a provided JWT token is valid.	High
FR6.2	The system should invalidate the provided Refresh Token of a user intending to log out.	High
FR6.3	The system should provide routes based on signature type to ensure a user can correctly validate their token based on purpose.	High

6.6.2 Non-Functional Requirements

S/N	Category	Requirements	Priority
NFR6.1	Security	The system should only validate JWT tokens that are signed with a particular type of signature.	Medium
NFR6.2	Security	The system should cross-check with the Redis instance to identify if a validated token is blacklisted and reject if true.	Medium

6.6.3 API Routes

The table below states the various routes supported by the Authentication Service

Request	API Route	Description
GET	/api/auth/verification	Verifies if a provided token is valid based on a verification secret
GET	/api/auth/refresh	Verifies if a provided token is valid based on a refresh secret

GET	/api/auth/access	Verifies if a provided token is valid based on a access secret
POST	/api/auth/:username	Disables a provided refresh token by inserting it to redis for future reference
GET	/api/auth/health	Checks the operational status of the service

6.6.4 Services

6.6.4.1 Validation of JWT Tokens

Through the authorization header, this service extracts out the embedded JWT token and using the jsonwebtoken module, verify it against a signature held by the service. This signature will differ depending on the purpose of the token as described in the User Service. If the token is verified to have its signature matching that required by the request, another check will be made to the connected Redis instance to ensure that the token is not present there. If present, it means that the token has been blacklisted by the service and will be rejected. Rejected tokens will prompt the server to throw a 401 Unauthorized error, which will be used when integrated with the gateway service to protect identified API calls to the other microservices.

With this feature, the authentication service can complement the needs of the gateway service. When the gateway receives an API request that requires token validation, it can perform a temporary routing to this service for token verification based on the type of token. If the token is invalid, the gateway can stop the API request from functioning with the response from this service to inform the user that they are not authorized to use the service they are requesting. On the other hand, a successful verification will allow the gateway to perform a proxy pass of the original request to the desired microservice.

The figure below shows the flow of validation of the passed tokens for the user service.

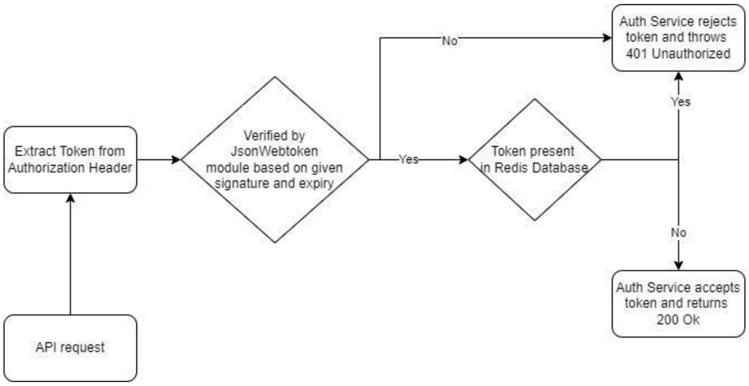

Extract Token from Authorization Header
Verified by JsonWebtoken module based on given signature and expiry
No
Yes
Auth Service rejects token and throws 401 Unauthorized
Token present in Redis Database
Yes
No
Auth Service accepts token and returns 200 Ok
API request

Figure 23: Flow of validation of JWT Token

6.6.4.2 Logout of users

Through the authorization header, the identified token (if it is a Bearer Token) will be decoded to obtain its details. If the token is already expired or invalid, no further action will be needed to be taken. On the other hand, the token will be inserted as a key-value pair (key being the token and value a constant variable) into the redis instance. It will be set to expire based on the exp attribute of the token, which was set at the point of token creation. After the set expiry, this token will be thrown away from the redis instance, preventing it from being filled with tokens that have been expired and unable to accept new tokens as a result.

As such, if a user attempts to use this token for authentication subsequently, it will get rejected since a check in this storage during token verification will reveal its existence, proving that it has been blacklisted.

7 Deployment

7.1 Local Setup

7.1.1 Rationale

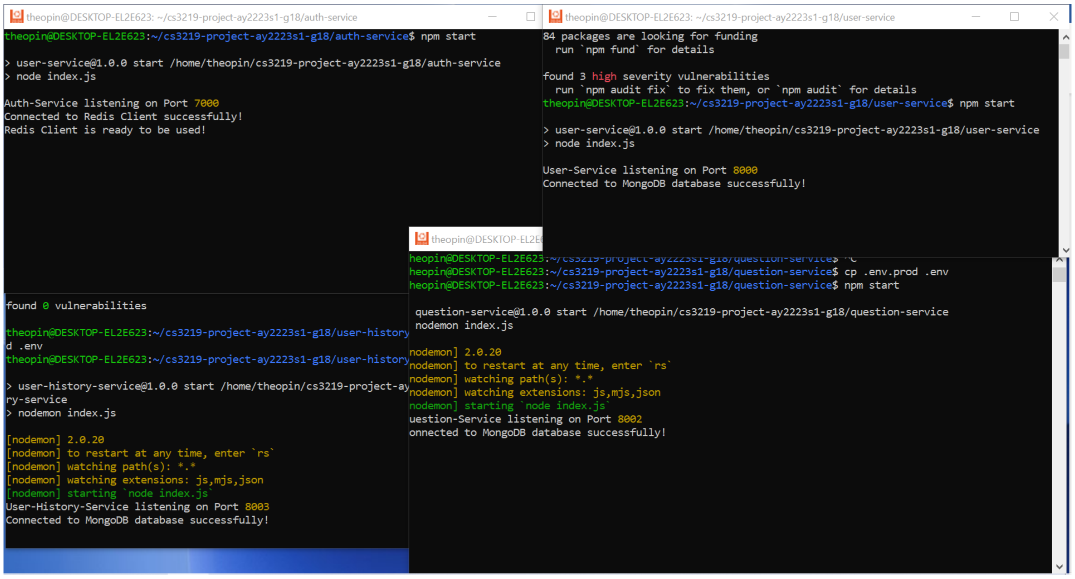
This method was used to allow our team to explore and develop features to build up the microservice we were tasked to build individually.

7.1.2 Implementation

To run each microservice individually the steps in the table below have to be performed, with the user being able to skip to step 4 once the microservice has been initialized.

S/N	**Requirements**
1	Navigate to the directory of the microservice
2	Run `npm i` to install dependencies required
3	Create a `.env` file and fill in the necessary environment variables
4	Run `npm start`/`npm test` to run or test the application respectively

An example of using step 4 to start up some microservices of PeerPrep is shown below.


theopin@DESKTOP-EL2E623: ~/cs3219-project-ay2223s1-g18/auth-service
theopin@DESKTOP-EL2E623: ~/cs3219-project-ay2223s1-g18/user-service
theopin@DESKTOP-EL2E

```
theopin@DESKTOP-EL2E623:~/cs3219-project-ay2223s1-g18/auth-service$ npm start

> user-service@1.0.0 start /home/theopin/cs3219-project-ay2223s1-g18/auth-service
> node index.js

Auth-Service listening on Port 7000
Connected to Redis Client successfully!
Redis Client is ready to be used!
```

```
84 packages are looking for funding
  run `npm fund` for details

found 3 high severity vulnerabilities
  run `npm audit fix` to fix them, or `npm audit` for details
theopin@DESKTOP-EL2E623:~/cs3219-project-ay2223s1-g18/user-service$ npm start

> user-service@1.0.0 start /home/theopin/cs3219-project-ay2223s1-g18/user-service
> node index.js

User-Service listening on Port 8000
Connected to MongoDB database successfully!
```

```
found 0 vulnerabilities

theopin@DESKTOP-EL2E623:~/cs3219-project-ay2223s1-g18/user-history
d .env
theopin@DESKTOP-EL2E623:~/cs3219-project-ay2223s1-g18/user-history

> user-history-service@1.0.0 start /home/theopin/cs3219-project-ay
ry-service
> nodemon index.js

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
User-History-Service listening on Port 8003
Connected to MongoDB database successfully!
```

```
heopin@DESKTOP-EL2E623:~/cs3219-project-ay2223s1-g18/question-service$ ^C
heopin@DESKTOP-EL2E623:~/cs3219-project-ay2223s1-g18/question-service$ cp .env.prod .env
heopin@DESKTOP-EL2E623:~/cs3219-project-ay2223s1-g18/question-service$ npm start

 question-service@1.0.0 start /home/theopin/cs3219-project-ay2223s1-g18/question-service
 nodemon index.js

nodemon] 2.0.20
nodemon] to restart at any time, enter `rs`
nodemon] watching path(s): *.*
nodemon] watching extensions: js,mjs,json
nodemon] starting `node index.js`
uestion-Service listening on Port 8002
onnected to MongoDB database successfully!
```

Figure 24: Example of utilizing a local setup to start up some microservices of PeerPrep

7.1.3 Evaluation

Initially, this method proved to be useful since our team was working on different microservices at the same time, which usually only required 1 terminal to run development and local testing efforts.

However, as we began combining microservices to bring PeerPrep together, debugging through this method was rather troublesome as multiple terminals had to be launched per microservice to proceed with integration efforts.

On the other hand, each of our team members were working on different platforms and Node versions, which made the integration process difficult as there were moments where some microservices could not run on another team member's machine due to such differences.

In the long run, this did not seem to be a viable solution especially with more microservices being planned to be integrated in the future.

7.2 Local Orchestration

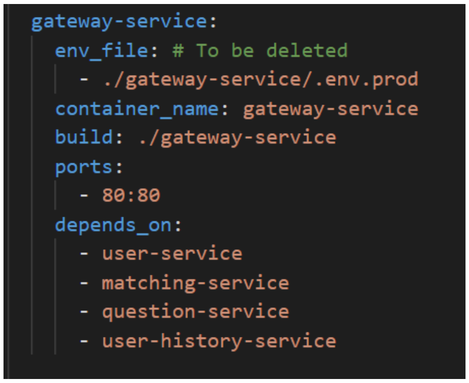
7.2.1 Rationale

Recognizing the issues that were coming up with a local setup (described in section 7.1.3 above), we decided to look for an alternative that could automate such a process. We identified Docker-Compose as a viable alternative as its functionalities allowed for a rapid, simple and standardized setup of PeerPrep on a single localhost machine.

7.2.2 Implementation

Through a docker-compose file, we will specify the instructions to instruct docker to build images corresponding to each microservice of PeerPrep as well as to run them with the required configurations. Each microservice will also have its own Dockerfile to specify the instructions on building itself, which can vary from microservice to microservice.

The figure below shows the instructions for running gateway-service, the API gateway used for this application.



```yaml
gateway-service:
  env_file: # To be deleted
    - ./gateway-service/.env.prod
  container_name: gateway-service
  build: ./gateway-service
  ports:
    - 80:80
  depends_on:
    - user-service
    - matching-service
    - question-service
    - user-history-service
```

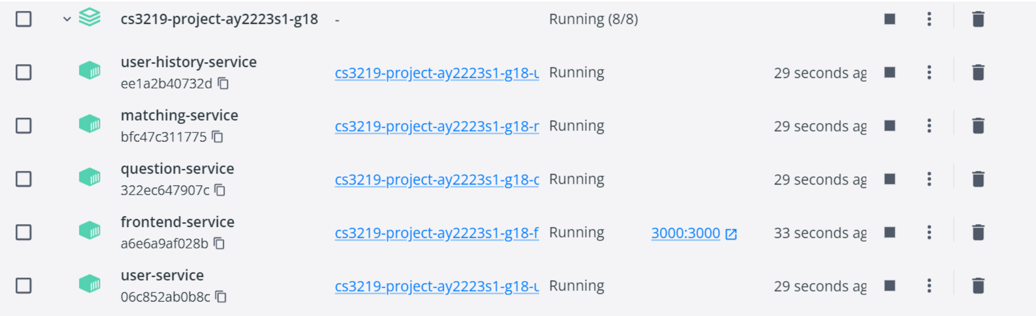
Figure 25: Local Orchestration for Gateway Service

As seen above, there are a few configurations involved in building this service. The table below describes their purposes. Depending on the implementation of microservices, these variables usage may change:

S/N	Variable	Function
1	env_file	Specifies the location of the environment file containing environment variables required for microservice functionality
2	container_name	Name of container running the microservice.

3	build	Location of the folder containing Dockerfile, the file containing instructions to build the isolated environment for the microservice
4	ports	Instructions on how to map the external and internal port if microservice is being exposed on the localhost. Currently only enabled on frontend and gateway-service
5	image	Publicly available image to be pulled (eg. redis)
6	depends-on	Sets order of container startup by providing names of containers to be run beforehand

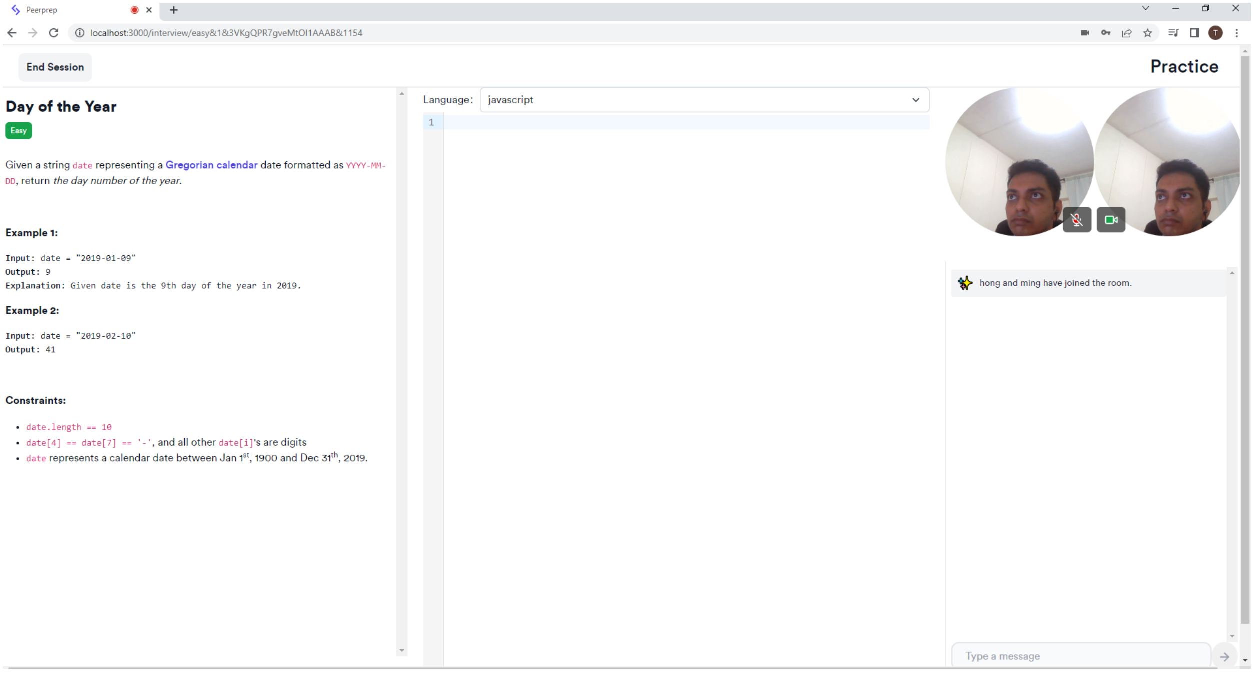
With this file, running the command ‘docker compose up -d’ will build the images (if not present) and run them in the background based on the given configurations. A screenshot of a successful docker compose viewed from Docker Desktop is shown in the figure below.



□	cs3219-project-ay2223s1-g18	-	Running (8/8)		
□	user-history-service ee1a2b40732d	cs3219-project-ay2223s1-g18-u	Running		29 seconds ag
□	matching-service bfc47c311775	cs3219-project-ay2223s1-g18-r	Running		29 seconds ag
□	question-service 322ec647907c	cs3219-project-ay2223s1-g18-c	Running		29 seconds ag
□	frontend-service a6e6a9af028b	cs3219-project-ay2223s1-g18-f	Running	3000:3000	33 seconds ag
□	user-service 06c852ab0b8c	cs3219-project-ay2223s1-g18-u	Running		29 seconds ag

Figure 26: Local Orchestration of microservices snippet view from Docker Desktop

The figure below shows an interview with 2 users logged in on the same orchestration on the local machine.



Day of the Year

Easy

Given a string `date` representing a Gregorian calendar date formatted as `YYYY-MM-DD`, return *the day number of the year.*

Example 1:

```
Input: date = "2019-01-09"
Output: 9
Explanation: Given date is the 9th day of the year in 2019.
```

Example 2:

```
Input: date = "2019-02-10"
Output: 41
```

Constraints:

- `date.length == 10`
- `date[4] == date[7] == '-'`, and all other `date[i]`'s are digits
- `date` represents a calendar date between Jan 1st, 1900 and Dec 31th, 2019.

hong and ming have joined the room.

Figure 27: Interview Session of 2 user accounts through local orchestration

7.2.3 Evaluation

With each docker image being specified to be built through a docker file, consistency is achieved as the same base environment for each microservice is standardized when built and usable across any type of platform other essential applications used (eg. Node), allowing PeerPrep to be more **portable**. This resolved the issue of errors arising due to cross-environment development of the team. The isolated nature of containers ensures that each microservice is run independently of each other except for the gateway service which acts as the central node to transfer requests around microservice. With such benefits, docker compose helped our team to complete integration efforts more quickly.

However, even with such an ease of starting up PeerPrep on our local environments, having it on a single localhost machine meant that it was not possible to connect to another user using PeerPrep on their own machines, which was one of the main objectives of PeerPrep – to allow for an avenue for collaboration to happen between different users. Moreover, this method of starting up PeerPrep is only suitable for development purposes and not for end users interested in utilizing PeerPrep.

7.3 Production

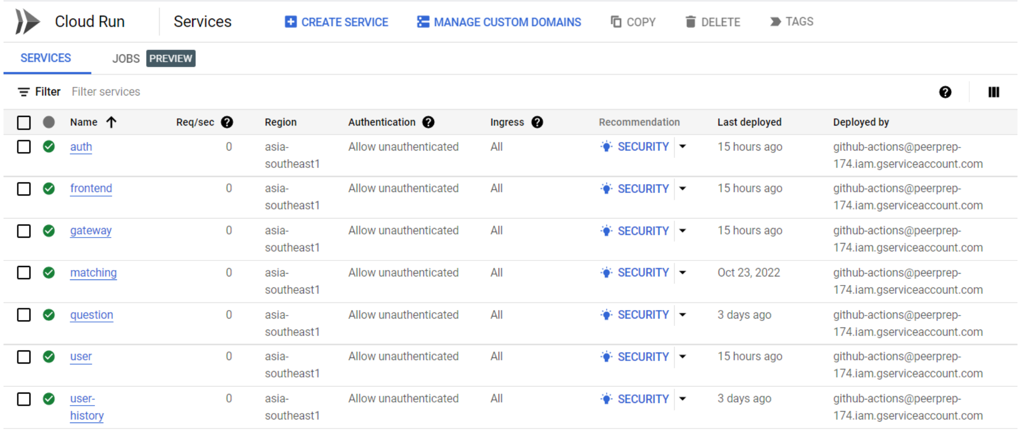
7.3.1 Rationale

To resolve the limitations of a local orchestration addressed in section 7.2.3 above, we decided to host PeerPrep publicly through Cloud Run of Google Cloud Platform (GCP). Cloud Run was chosen due to the simplicity of deploying applications as well as the

7.3.2 Implementation

The orchestration of services in Cloud Run is similar to a local orchestration. For each microservice to be deployed, deployment configuration such as environment variables, instance type and the docker image of the microservice stored in GCP is specified so that the service on Cloud Run understands how to deploy the microservice. Using CD (detailed in section 8.4), we are also able to configure each Cloud Run service to use a new image consisting of the changes made in the latest pull request.

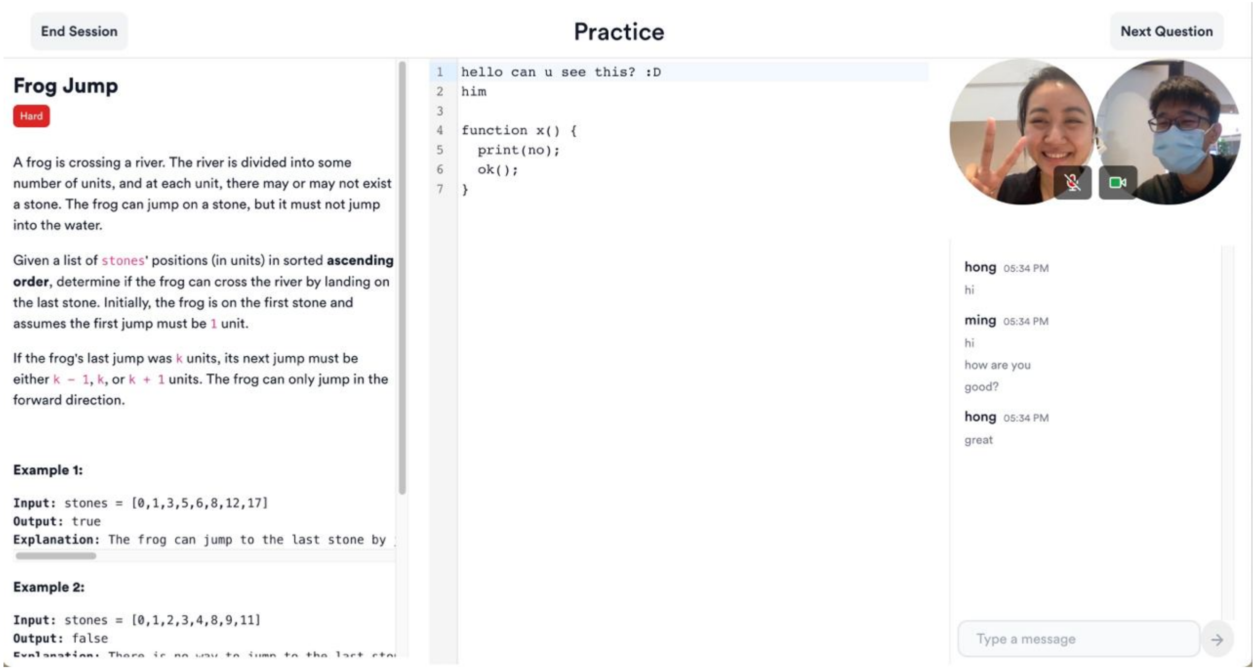
The figure below shows the orchestration of all the microservices on PeerPrep after such configurations.



□	●	Name ↑	Req/sec	Region	Authentication	Ingress	Recommendation	Last deployed	Deployed by
□	✓	auth	0	asia-southeast1	Allow unauthenticated	All	SECURITY	15 hours ago	github-actions@peerprep-174.iam.gserviceaccount.com
□	✓	frontend	0	asia-southeast1	Allow unauthenticated	All	SECURITY	15 hours ago	github-actions@peerprep-174.iam.gserviceaccount.com
□	✓	gateway	0	asia-southeast1	Allow unauthenticated	All	SECURITY	15 hours ago	github-actions@peerprep-174.iam.gserviceaccount.com
□	✓	matching	0	asia-southeast1	Allow unauthenticated	All	SECURITY	Oct 23, 2022	github-actions@peerprep-174.iam.gserviceaccount.com
□	✓	question	0	asia-southeast1	Allow unauthenticated	All	SECURITY	3 days ago	github-actions@peerprep-174.iam.gserviceaccount.com
□	✓	user	0	asia-southeast1	Allow unauthenticated	All	SECURITY	15 hours ago	github-actions@peerprep-174.iam.gserviceaccount.com
□	✓	user-history	0	asia-southeast1	Allow unauthenticated	All	SECURITY	3 days ago	github-actions@peerprep-174.iam.gserviceaccount.com

Figure 28: Cloud Run GUI of PeerPrep's deployed microservices

By bringing PeerPrep to a public domain, users can use different computers communicate with each other remotely. An example of the interview session using such a deployment linking 2 different users is shown in the Figure below.



Frog Jump

Hard

A frog is crossing a river. The river is divided into some number of units, and at each unit, there may or may not exist a stone. The frog can jump on a stone, but it must not jump into the water.

Given a list of `stones`' positions (in units) in sorted **ascending order**, determine if the frog can cross the river by landing on the last stone. Initially, the frog is on the first stone and assumes the first jump must be `1` unit.

If the frog's last jump was `k` units, its next jump must be either `k - 1`, `k`, or `k + 1` units. The frog can only jump in the forward direction.

Example 1:

```
Input: stones = [0,1,3,5,6,8,12,17]
Output: true
Explanation: The frog can jump to the last stone by
```

Example 2:

```
Input: stones = [0,1,2,3,4,8,9,11]
Output: false
Explanation: There is no way to jump to the last sto
```

```
hello can u see this? :D
him

function x() {
  print(no);
  ok();
}
```

hong 05:34 PM
hi

ming 05:34 PM
hi
how are you
good?

hong 05:34 PM
great

Figure 29: Connection of 2 different users through deployment

7.3.3 Evaluation

As such, having such a public web environment for PeerPrep expands its degree of **Portability** as users can use it regardless of location, browser or operating system as long as they have a stable internet connection. With the complex setup of orchestration now being handled by Cloud Run, users need not worry of how to set up PeerPrep on their Desktop and can enjoy the same experience of PeerPrep simply by navigating to its deployed website.

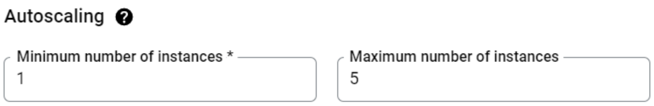
7.4 Scalability

7.4.1 Rationale

Having achieved the ability to run PeerPrep online, our team recognized the concern that with PeerPrep being widely available, increased concurrent network traffic would prevent others from being able to use PeerPrep. Hence, we decided to explore how to allow PeerPrep to scale on Cloud Run to respond to such a potential issue and allow as many users to be able to use PeerPrep concurrently as possible.

7.4.2 Implementation

Cloud Run allows for users to scale the number of instances of their containers as shown in the figure below.



Autoscaling

Minimum number of instances *: 1

Maximum number of instances: 5

Figure 30: Autoscaling Configuration of Cloud Run

As shown above, we have set a minimum and maximum of 1 and 5 instances respectively for each microservice. Such a configuration ensures that an incoming request does not have to wait for a new container instance to be started for it to be processed (cold starts). Moreover, more instances of a particular microservice can be activated if the demand for it increases in the form of increased requests.

7.4.3 Evaluation

Having the ability to **scale** microservices independently allows PeerPrep to react to a spike in network traffic in the event demand for any of its microservices increases and reduce the scale when its demand dies on the other hand. Having such a scaling capability allows for efficient allocation of resources on Cloud Run to support the needs of PeerPrep when deployed on a public domain. The tests of this capability are detailed in the Stress test in Section 9.3 below.

7.5 API Gateway

7.5.1 Rationale

Although the design allowed for PeerPrep's microservices to be accessible through the frontend service, it had to maintain a list of all the locations of the deployed endpoints of such microservices. As such, we decided to explore the usage of an API gateway to streamline these endpoints into a common endpoint, with sections of the endpoint varied to distinguish the intended microservice the request intended to reach the gateway and utilize the gateway's service to route the request.

7.5.2 Implementation

To implement a gateway, we used a nginx docker image with a custom configuration file to specify the mapping from the exposed endpoints to that of each microservices through a proxy pass.

Once the development of the authentication service was complete, the next step was to get the gateway to perform redirection into its service for the purpose of token validation to ensure that users could not freely access protected API calls without having a valid token.

A summarized flow of the gateway service is shown in the figure below.

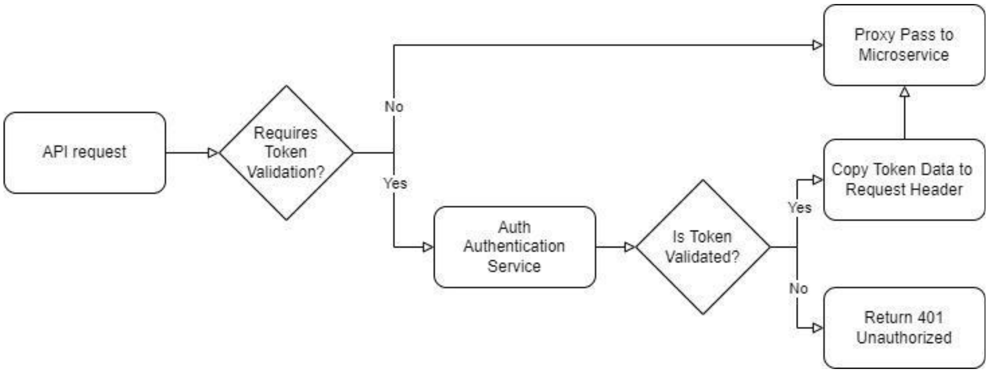

API request
Requires Token Validation?
No
Yes
Proxy Pass to Microservice
Auth Authentication Service
Is Token Validated?
Yes
No
Copy Token Data to Request Header
Return 401 Unauthorized

Figure 31: Flowchart of API Gateway

As seen above, if the request does not require a token verification, a proxy pass is made directly to the target microservice. On the other hand, the request is first passed to the authentication service to check the validity of the token through the auth_request functionality of nginx. If successful, the details of the token are inserted into a header before proxy passing the request to the target microservice. If an invalid token is supplied to such request (eg. expired token, invalid token type), the request will stop at the authentication service with an error code of 401 being returned to the user.

7.5.3 Evaluation

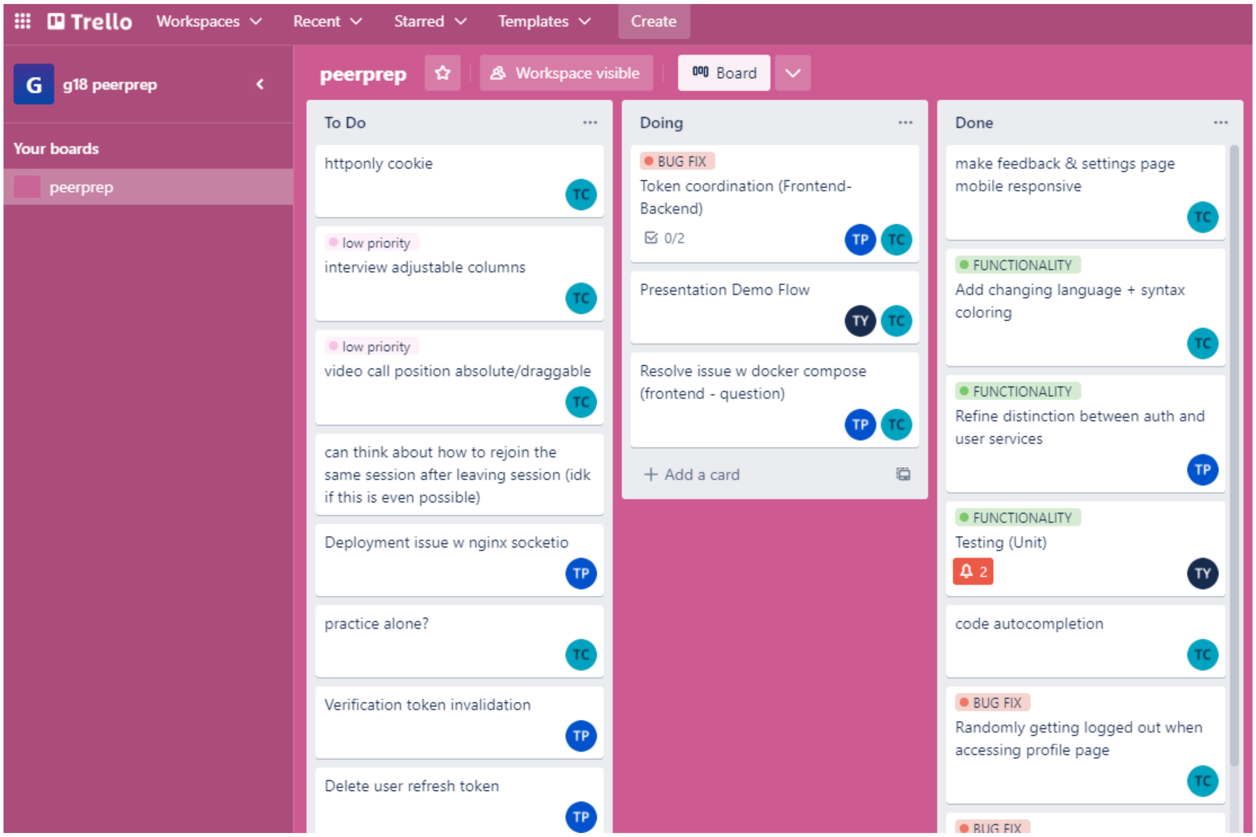
The API gateway in this implementation adopts a Mediator pattern rather than a Façade pattern because on top of just routing API requests to the respective services, there is additional logic included to authenticate each request by communicating with the Authentication service. In a sense it mediates the communication between the other microservices with the authentication service, to bring about enhanced security by enforcing that only request with a verified token is allowed to be proxy passed onwards. Using the Mediator pattern has helped us reduce the coupling amongst the services because it encapsulates how the services interact when a new API request comes in. As a result of implementing this pattern, services do not have to refer to the authentication service explicitly as the interaction with it is being handled by the API gateway.

8 Development Process

8.1 Scrum

We have weekly scrum meetings every Monday evening. During each scrum meeting, we have a Sprint Review and plan for the next Sprint. Each Sprint last for a week, and the Sprint Backlog is detailed in the Project Timeline.

We also made use of Trello as a project management tool, to keep track of the tasks to do for each Sprint, and to allocate the workload to each member of the team.



To Do

- httponly cookie — TC
- low priority: interview adjustable columns — TC
- low priority: video call position absolute/draggable — TC
- can think about how to rejoin the same session after leaving session (idk if this is even possible)
- Deployment issue w nginx socketio — TP
- practice alone? — TC
- Verification token invalidation — TP
- Delete user refresh token — TP

Doing

- BUG FIX: Token coordination (Frontend-Backend) — ☑ 0/2 — TP TC
- Presentation Demo Flow — TY TC
- Resolve issue w docker compose (frontend - question) — TP TC

Done

- make feedback & settings page mobile responsive — TC
- FUNCTIONALITY: Add changing language + syntax coloring — TC
- FUNCTIONALITY: Refine distinction between auth and user services — TP
- FUNCTIONALITY: Testing (Unit) — 2 — TY
- code autocompletion — TC
- BUG FIX: Randomly getting logged out when accessing profile page — TC
- BUG FIX

Figure 32: Trello Board for PeerPrep Project

8.2 Coding Practices

8.2.1 ESLint

We place heavy emphasis on writing and always pushing clean code, ensuring everyone adheres to the same code formatting guidelines. This led us to implement ESLint for code linting. In addition, we added this code linting step within our Continuous Integration steps in Github Actions, always ensuring that all code within the repository would be of a minimum standard quality.

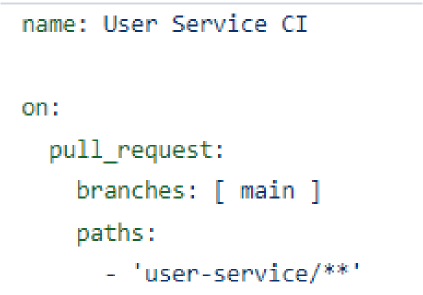
8.3 Continuous Integration (CI)

8.3.1 Rationale

We decided to adopt this approach to enable the automation of testing only on specific microservices which have changes made to their codebase, and are being pushed to the main branch though a pull request. We used Github Actions to facilitate the implementation of CI Processes for each microservices.

8.3.2 Implementation

As shown below, each CI workflow is triggered to automatically run upon a pull request to the main branch, provided changes are made to the implementation of the microservice it is designed for. An example of a workflow, targeting the testing of the user service, is shown below.



```
name: User Service CI

on:
  pull_request:
    branches: [ main ]
    paths:
      - 'user-service/**'
```

Figure 33: Trigger Criteria of CI for User Service

Each workflow is comprised of a few tasks and follows a standard flow comprising a few major steps. The illustration and details of each step are given in the figure and table below.

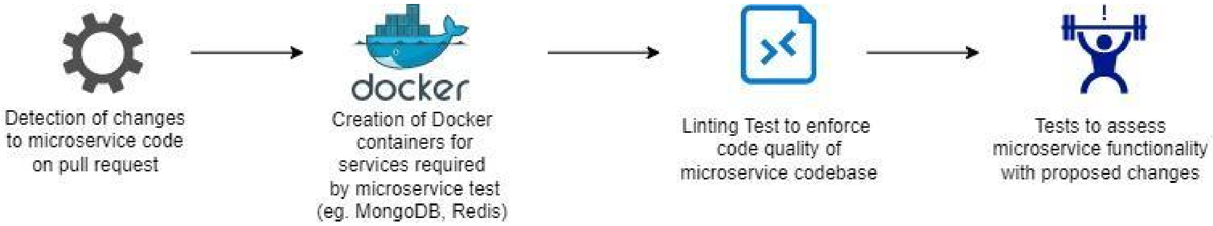
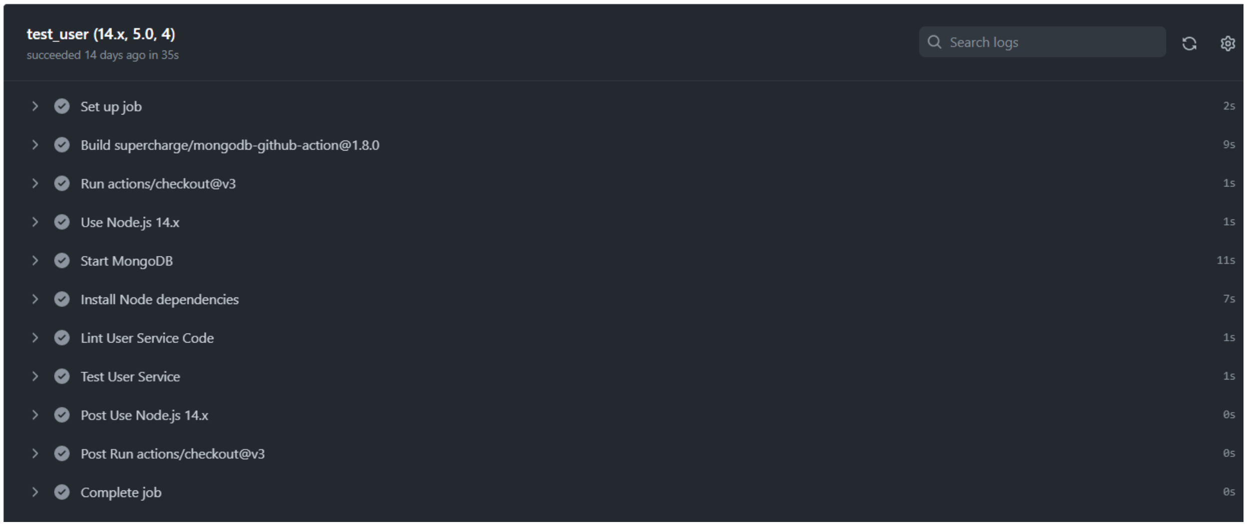

Detection of changes to microservice code on pull request
docker
Creation of Docker containers for services required by microservice test (eg. MongoDB, Redis)
Linting Test to enforce code quality of microservice codebase
Tests to assess microservice functionality with proposed changes

Figure 34: Flow of CI Workflow

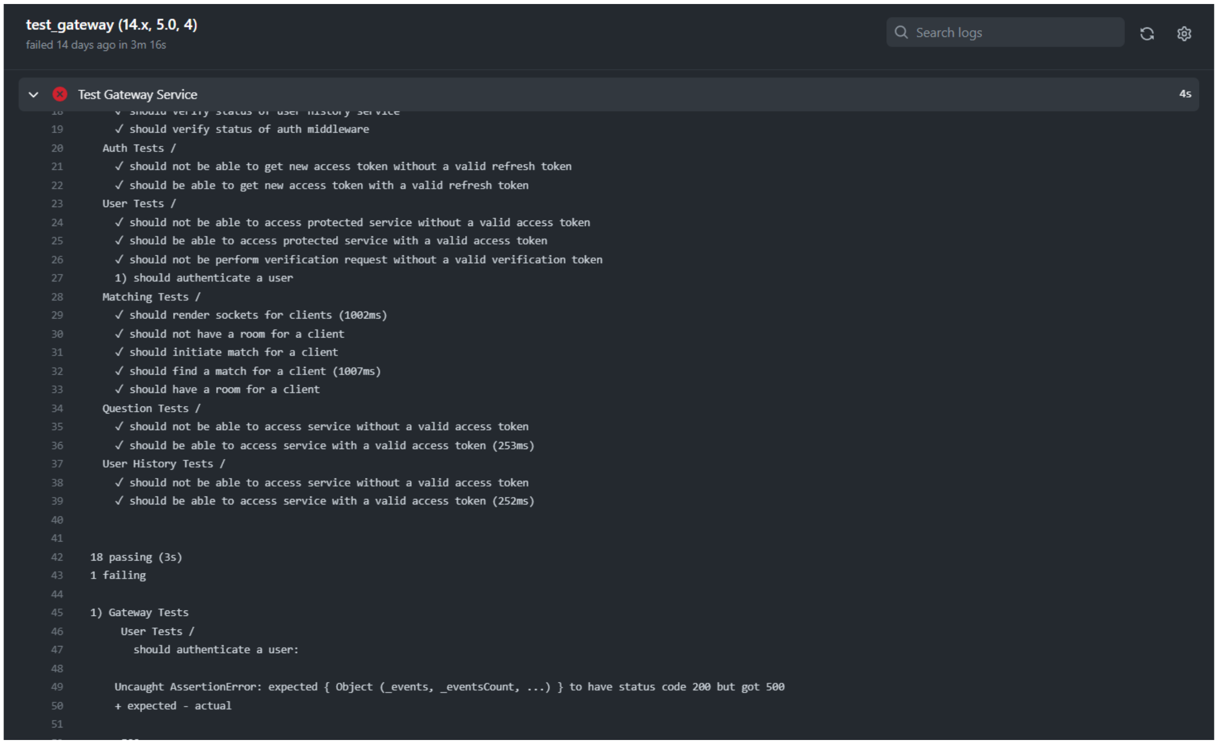
S/N	Step	Rationale
1	Trigger Conditions	Ensures that only microservices that have their codebase changed have their functionalities tested.
2	Creation of Supporting Docker Containers	Start local instances of MongoDB and Redis in the runner if required to support functionality of the microservice.
3	Linting Test	Checks codebase of target microservice to ensure it conforms to an appropriate coding standard using ESLint.
4	Microservice	Runs tests given on microservice to check if existing features work as intended even with changes to codebase using Mocha and Chai.

From the results of the triggered tests, both the author and reviewers can assess if the pull request can be ready to be merged and subsequently deployed to the production version. An example of a successful and failed CI test is shown in the Figure below.



```
test_user (14.x, 5.0, 4)
succeeded 14 days ago in 35s

Set up job                                          2s
Build supercharge/mongodb-github-action@1.8.0       9s
Run actions/checkout@v3                             1s
Use Node.js 14.x                                    1s
Start MongoDB                                      11s
Install Node dependencies                           7s
Lint User Service Code                              1s
Test User Service                                   1s
Post Use Node.js 14.x                               0s
Post Run actions/checkout@v3                        0s
Complete job                                        0s
```

Figure 35: Successful CI Test of User Service



test_gateway (14.x, 5.0, 4)
failed 14 days ago in 3m 16s

Test Gateway Service
4s

```
    ✓ should verify status of user history service
    ✓ should verify status of auth middleware
  Auth Tests /
    ✓ should not be able to get new access token without a valid refresh token
    ✓ should be able to get new access token with a valid refresh token
  User Tests /
    ✓ should not be able to access protected service without a valid access token
    ✓ should be able to access protected service with a valid access token
    ✓ should not be perform verification request without a valid verification token
    1) should authenticate a user
  Matching Tests /
    ✓ should render sockets for clients (1002ms)
    ✓ should not have a room for a client
    ✓ should initiate match for a client
    ✓ should find a match for a client (1007ms)
    ✓ should have a room for a client
  Question Tests /
    ✓ should not be able to access service without a valid access token
    ✓ should be able to access service with a valid access token (253ms)
  User History Tests /
    ✓ should not be able to access service without a valid access token
    ✓ should be able to access service with a valid access token (252ms)



18 passing (3s)
1 failing

1) Gateway Tests
     User Tests /
       should authenticate a user:

    Uncaught AssertionError: expected { Object (_events, _eventsCount, ...) } to have status code 200 but got 500
    + expected - actual

```

Figure 36: Failed CI Test of User Service

8.3.3 Evaluation

By separating workflows, the CI workflow can be optimized to only run tests which are specialized to the microservices and in parallel. This ensures that the results can be analyzed independently of each other, much like how the microservices are designed to run.

With the results of the test being displayed to the author of the Pull Request, he/she can identify if there are any issues with the proposed change in code, either due to an oversight of linting practices or conflicts with existing features as flagged by the microservice tests. Having such a second layer of automated checks.

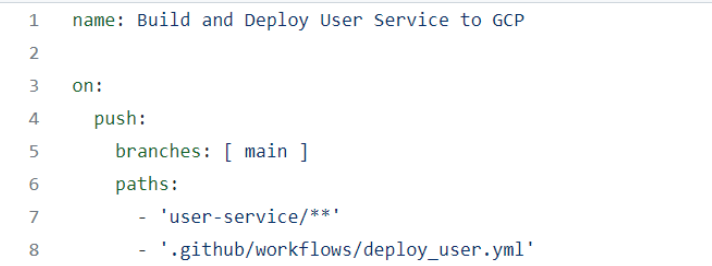
8.4 Continuous Deployment (CD)

8.4.1 Rationale

We decided to adopt this approach to enable the automation of deployment of microservices onto Cloud Run. These microservices are the specific few which have changes made to their codebase and have been pushed to the main branch through an approved pull request. Like CI (in Section 8.3), we have used Github Actions to facilitate the implementation of CD Processes for each microservices.

8.4.2 Implementation

Each CD workflow is triggered to automatically run upon a push to the main branch, provided changes are made to the implementation of the microservice it is designed for. An example of a workflow, targeting deployment of the user service, is shown below.



```
name: Build and Deploy User Service to GCP

on:
  push:
    branches: [ main ]
    paths:
      - 'user-service/**'
      - '.github/workflows/deploy_user.yml'
```

Figure 37: Trigger Criteria of CD for User Service

Each workflow is comprised of a few tasks and follows a standard flow comprising a few major steps. The illustration and details of each step are given in the figure and table below.

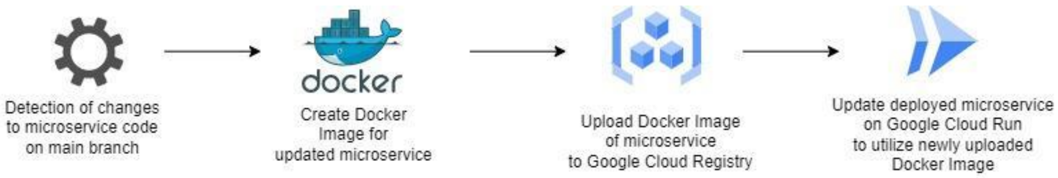
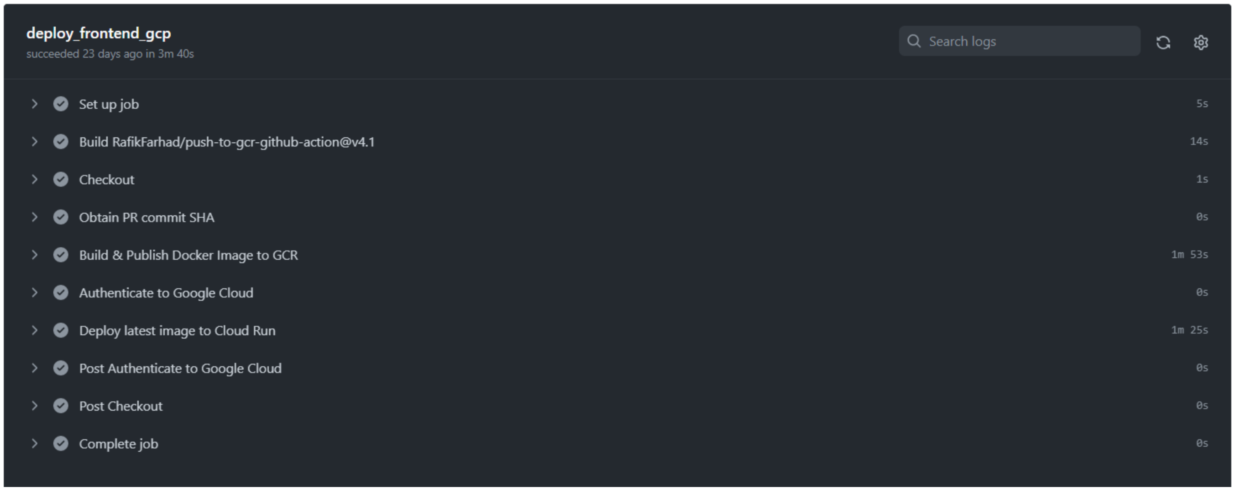


Figure 38: Flow of CD Workflow

S/N	Step	Rationale
1	Trigger Conditions	Ensures that only microservices that have their codebase changed from reference Pull Request to main branch will be redeployed to Cloud Run.
2	Creation of Docker Image	Creates a Docker Image based on the Dockerfile of the target microservice.
3	Upload Docker Image to Google Container Registry (GCR)	Upload previously created Docker Image to GCR to be stored and utilized for service.
4	Redeploy Microservice	Configure service on Cloud Run to use latest image uploaded to GCR.

From the results of the triggered tests, the development team can verify the success of a deployment of an identified microservice to Cloud Run. An example of a successful CD deployment is shown in the figure below.



```
deploy_frontend_gcp
succeeded 23 days ago in 3m 40s

Set up job                                        5s
Build RafikFarhad/push-to-gcr-github-action@v4.1  14s
Checkout                                          1s
Obtain PR commit SHA                              0s
Build & Publish Docker Image to GCR               1m 53s
Authenticate to Google Cloud                      0s
Deploy latest image to Cloud Run                  1m 25s
Post Authenticate to Google Cloud                 0s
Post Checkout                                     0s
Complete job                                      0s
```

Figure 39: Successful CD Deployment of Frontend

8.4.3 Evaluation

We decided to adopt this approach to maximize the effectiveness of deploying microservices. By separating workflows, we can conduct tests which are specialized to the microservices and only build and deploy the relevant microservices which had their code changed instead of defaulting to rebuilding and deploying every microservice, which can be rather time intensive.

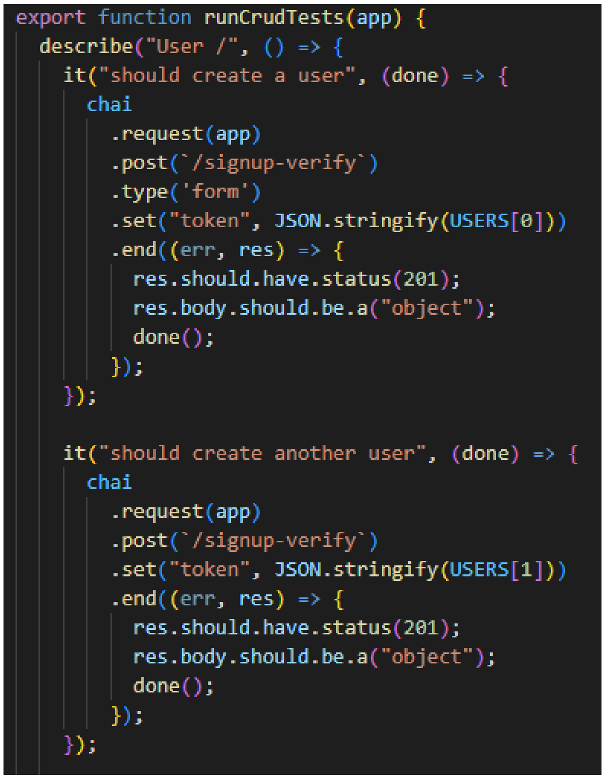
Moreover, by observing how the actions to build Docker images as well as deploying them to Cloud run can be time intensive, we decided on such an approach to run such operations in parallel, which allows all targeted microservices to be deployed to Cloud Run as fast as possible rather than to wait for the previous microservice if a sequential order is developed. A sequential deployment also risks the possibility of the subsequent microservices not being deployed if the current microservices has issues with deployment.

8.5 Testing

8.5.1 Unit Testing

Unit testing is implemented on each backend microservice. This is done to ensure that each microservice still works as intended as we continue to develop each microservice and expand our application. We used Mocha library as the JavaScript test framework to test our Node.js application and Chai as the assertion library.

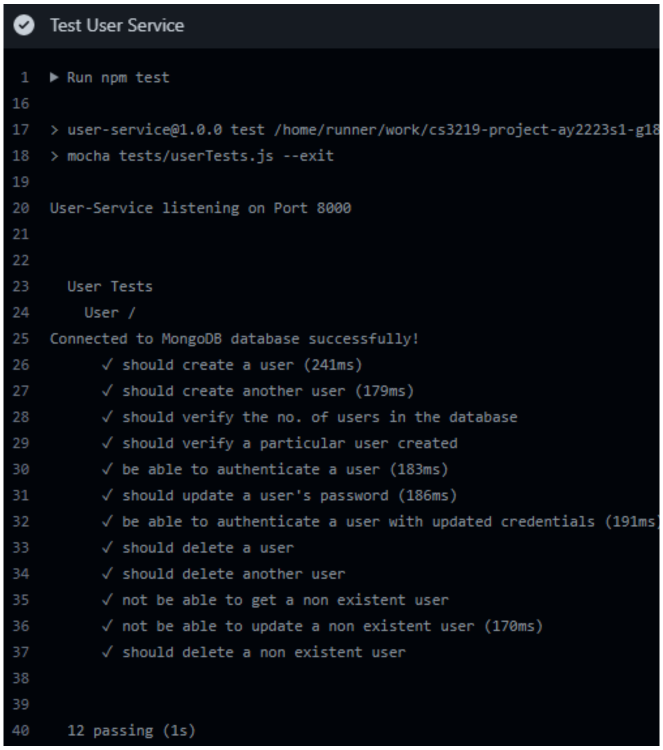
An example of the Unit Testing code for User Service is shown below.



```
export function runCrudTests(app) {
  describe("User /", () => {
    it("should create a user", (done) => {
      chai
        .request(app)
        .post(`/signup-verify`)
        .type('form')
        .set("token", JSON.stringify(USERS[0]))
        .end((err, res) => {
          res.should.have.status(201);
          res.body.should.be.a("object");
          done();
        });
    });

    it("should create another user", (done) => {
      chai
        .request(app)
        .post(`/signup-verify`)
        .set("token", JSON.stringify(USERS[1]))
        .end((err, res) => {
          res.should.have.status(201);
          res.body.should.be.a("object");
          done();
        });
    });
```

Figure 40: Code Snippet of User Service Unit Test

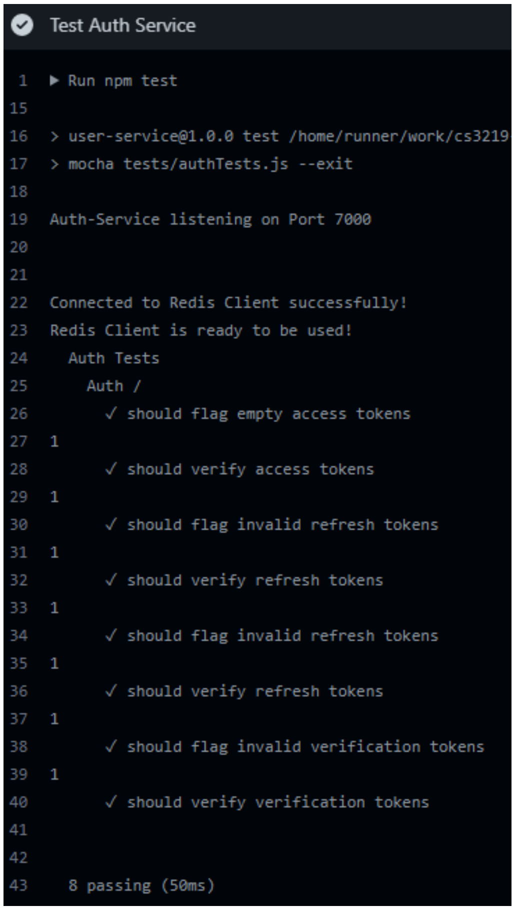
The Unit Tests for the various microservices executing during Continuous Integration is shown in the figures below.



Test User Service

```
▶ Run npm test

> user-service@1.0.0 test /home/runner/work/cs3219-project-ay2223s1-g18
> mocha tests/userTests.js --exit

User-Service listening on Port 8000



  User Tests
    User /
Connected to MongoDB database successfully!
      ✓ should create a user (241ms)
      ✓ should create another user (179ms)
      ✓ should verify the no. of users in the database
      ✓ should verify a particular user created
      ✓ be able to authenticate a user (183ms)
      ✓ should update a user's password (186ms)
      ✓ be able to authenticate a user with updated credentials (191ms
      ✓ should delete a user
      ✓ should delete another user
      ✓ not be able to get a non existent user
      ✓ not be able to update a non existent user (170ms)
      ✓ should delete a non existent user


  12 passing (1s)
```

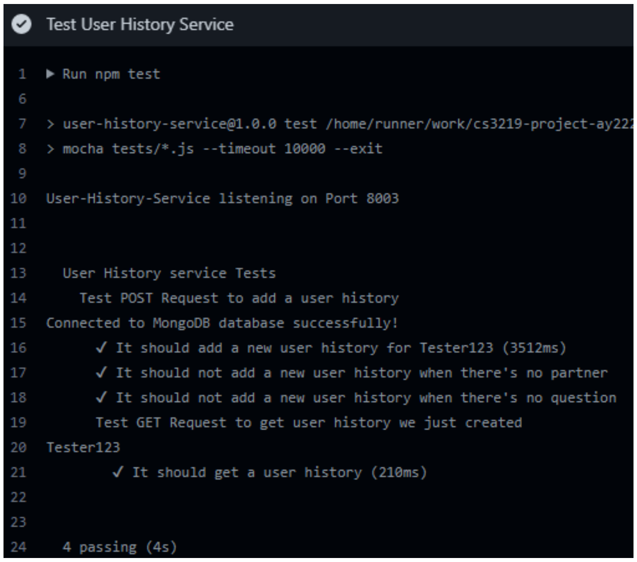
Figure 41: Unit Testing of User Service in GitHub Actions



Test Auth Service

```
▶ Run npm test

> user-service@1.0.0 test /home/runner/work/cs3219
> mocha tests/authTests.js --exit

Auth-Service listening on Port 7000


Connected to Redis Client successfully!
Redis Client is ready to be used!
  Auth Tests
    Auth /
      ✓ should flag empty access tokens
1
      ✓ should verify access tokens
1
      ✓ should flag invalid refresh tokens
1
      ✓ should verify refresh tokens
1
      ✓ should flag invalid refresh tokens
1
      ✓ should verify refresh tokens
1
      ✓ should flag invalid verification tokens
1
      ✓ should verify verification tokens


  8 passing (50ms)
```

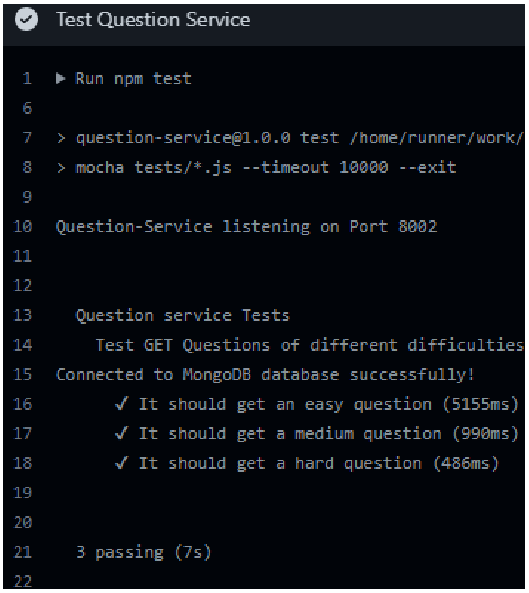
Figure 42: Unit Testing of Auth Service in GitHub Actions



✓ Test User History Service

```
▶ Run npm test

> user-history-service@1.0.0 test /home/runner/work/cs3219-project-ay222
> mocha tests/*.js --timeout 10000 --exit

User-History-Service listening on Port 8003


  User History service Tests
    Test POST Request to add a user history
Connected to MongoDB database successfully!
      ✓ It should add a new user history for Tester123 (3512ms)
      ✓ It should not add a new user history when there's no partner
      ✓ It should not add a new user history when there's no question
      Test GET Request to get user history we just created
Tester123
        ✓ It should get a user history (210ms)


  4 passing (4s)
```

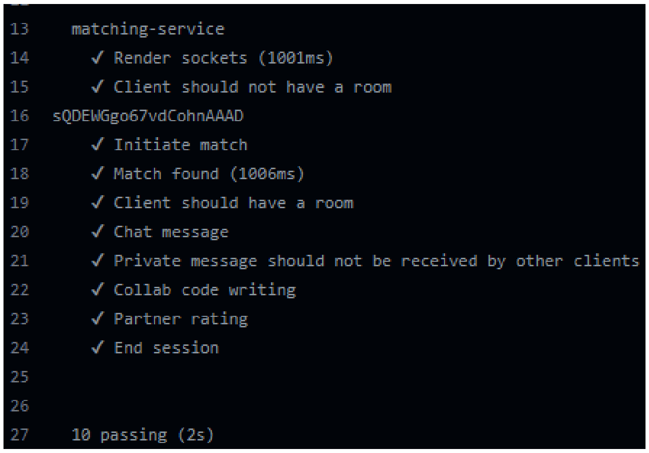
Figure 43: Unit Testing of User History Service in GitHub Actions



✓ Test Question Service

```
▶ Run npm test

> question-service@1.0.0 test /home/runner/work/
> mocha tests/*.js --timeout 10000 --exit

Question-Service listening on Port 8002



  Question service Tests
    Test GET Questions of different difficulties
Connected to MongoDB database successfully!
      ✓ It should get an easy question (5155ms)
      ✓ It should get a medium question (990ms)
      ✓ It should get a hard question (486ms)


  3 passing (7s)

```

Figure 44: Unit Testing of Question Service in GitHub Actions

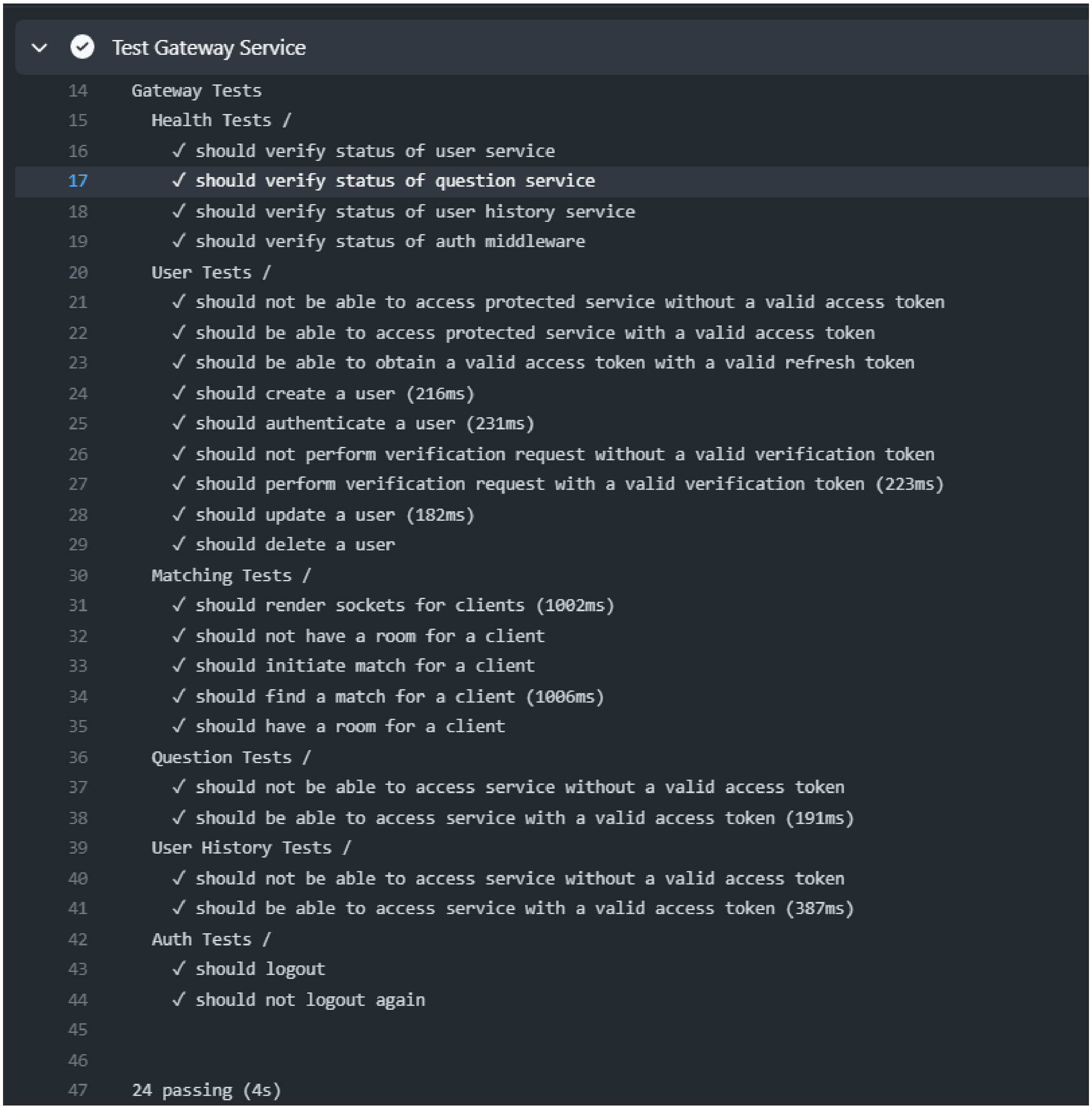


```
  matching-service
    ✓ Render sockets (1001ms)
    ✓ Client should not have a room
sQDEWGgo67vdCohnAAAD
    ✓ Initiate match
    ✓ Match found (1006ms)
    ✓ Client should have a room
    ✓ Chat message
    ✓ Private message should not be received by other clients
    ✓ Collab code writing
    ✓ Partner rating
    ✓ End session


  10 passing (2s)
```

Figure 45: Unit Testing of Collaboration Service in GitHub Actions

8.5.2 Integration Testing

We decided to use this method to test the functionality of the Gateway Service. With the gateway service acting as a mediator between the frontend and the other microservices, testing the gateways interaction with the other microservices would be an effective way to test and resolve any errors that are potentially detected from such tests, whether they stemmed from the gateway service or the other microservices they were linked to. The summary of tests performed through CI are shown in Figure 46 below.



```
Gateway Tests
  Health Tests /
    ✓ should verify status of user service
    ✓ should verify status of question service
    ✓ should verify status of user history service
    ✓ should verify status of auth middleware
  User Tests /
    ✓ should not be able to access protected service without a valid access token
    ✓ should be able to access protected service with a valid access token
    ✓ should be able to obtain a valid access token with a valid refresh token
    ✓ should create a user (216ms)
    ✓ should authenticate a user (231ms)
    ✓ should not perform verification request without a valid verification token
    ✓ should perform verification request with a valid verification token (223ms)
    ✓ should update a user (182ms)
    ✓ should delete a user
  Matching Tests /
    ✓ should render sockets for clients (1002ms)
    ✓ should not have a room for a client
    ✓ should initiate match for a client
    ✓ should find a match for a client (1006ms)
    ✓ should have a room for a client
  Question Tests /
    ✓ should not be able to access service without a valid access token
    ✓ should be able to access service with a valid access token (191ms)
  User History Tests /
    ✓ should not be able to access service without a valid access token
    ✓ should be able to access service with a valid access token (387ms)
  Auth Tests /
    ✓ should logout
    ✓ should not logout again


Gateway
24 passing (4s)
```

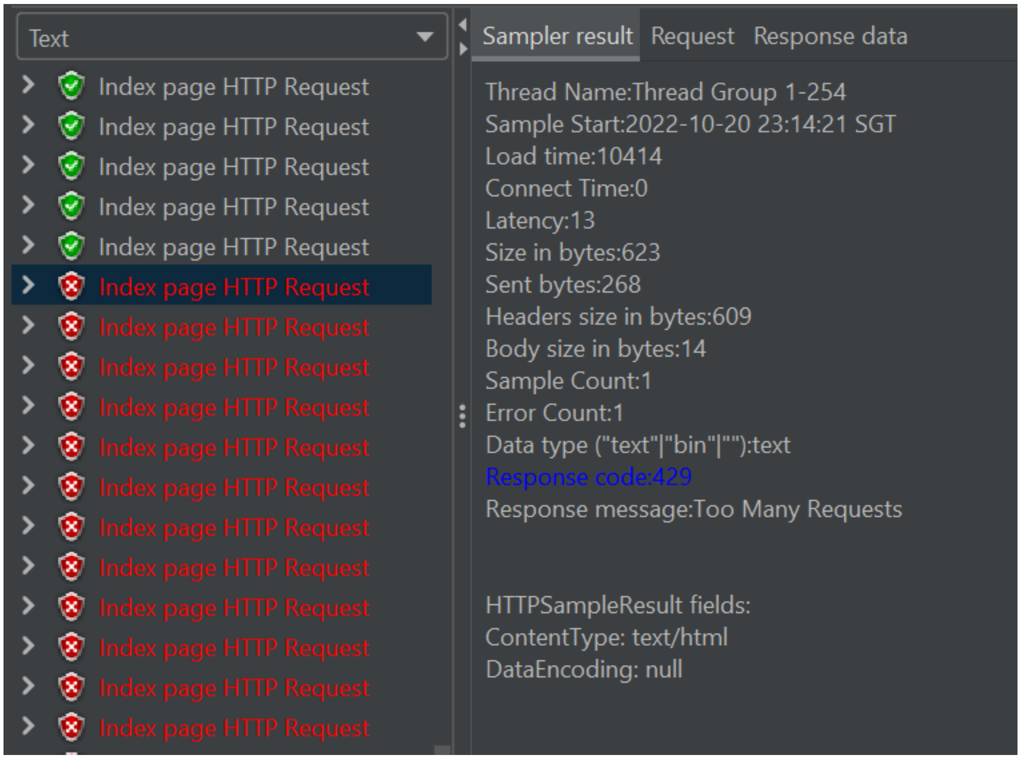
Figure 46: Integration Testing of API Gateway

The tests are designed to fit 3 types of API calls, as detailed below.

S/N	Test	Functionality
1	Health	Checks if the gateway can perform a successful proxy pass to the requested microservice
2	Exposed	Checks if the gateway can return a successful response from an API which does not require an authentication token
3	Protected	Checks if the gateway can return a successful response from an API which requires an authentication token to be verified prior to it.

8.6 Stress Testing

Stress tests were conducted to help us determine the upper limits of our system's capacity so that we will know how much traffic PeerPrep can handle, and how we can approach scaling up or better optimizing our system, or parts of our system, in future. We used Apache JMeter to stress test the primary API calls in all our microservices by progressively increasing the traffic load on each service until one started showing errors. The figure below shows a view of the results the JMeter displays.



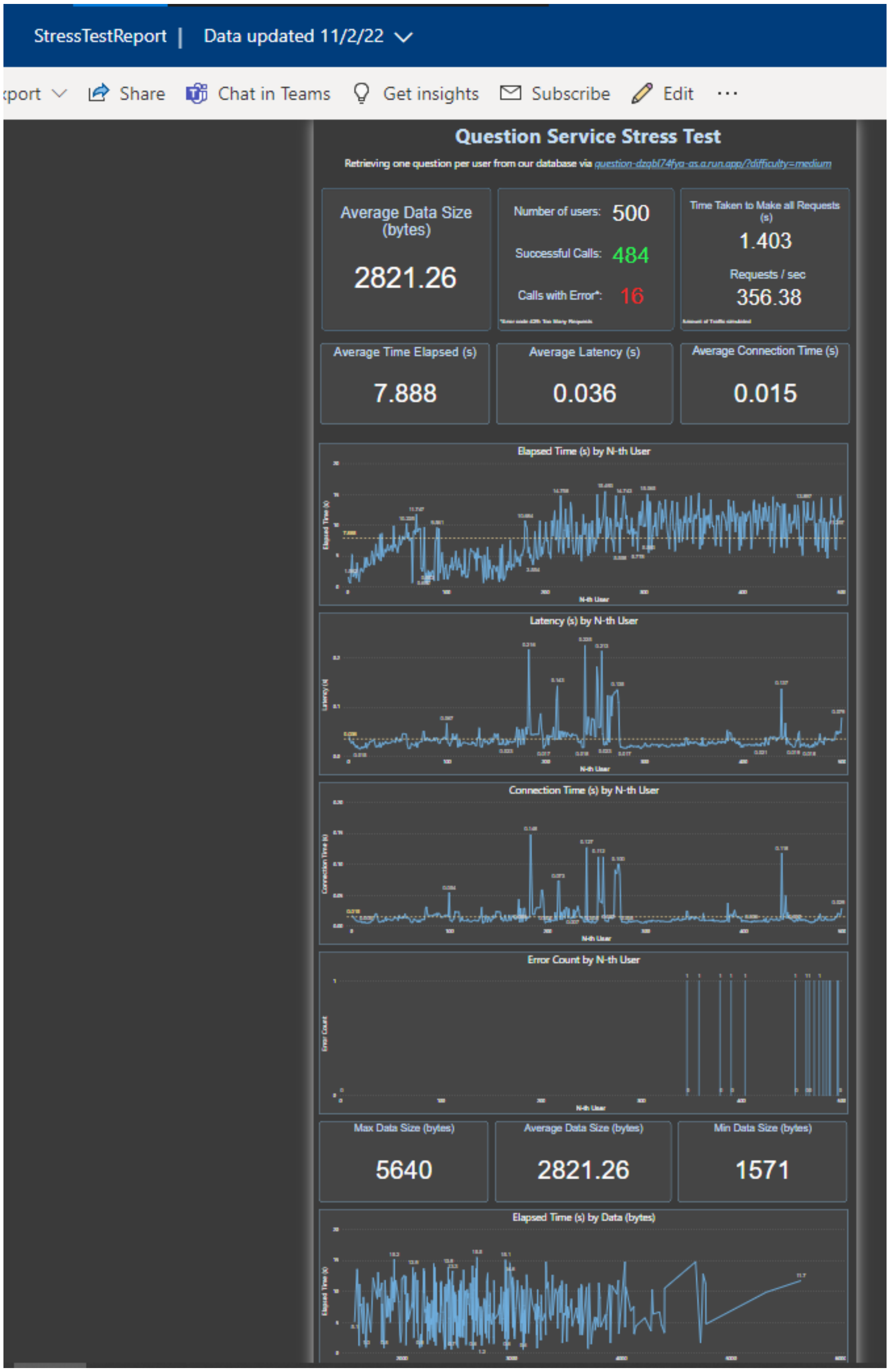
```
✓ Index page HTTP Request
✓ Index page HTTP Request
✓ Index page HTTP Request
✓ Index page HTTP Request
✓ Index page HTTP Request
✗ Index page HTTP Request
✗ Index page HTTP Request
✗ Index page HTTP Request
✗ Index page HTTP Request
✗ Index page HTTP Request
✗ Index page HTTP Request
✗ Index page HTTP Request
✗ Index page HTTP Request
✗ Index page HTTP Request
✗ Index page HTTP Request
✗ Index page HTTP Request
✗ Index page HTTP Request
```

```
Thread Name:Thread Group 1-254
Sample Start:2022-10-20 23:14:21 SGT
Load time:10414
Connect Time:0
Latency:13
Size in bytes:623
Sent bytes:268
Headers size in bytes:609
Body size in bytes:14
Sample Count:1
Error Count:1
Data type ("text"|"bin"|""):text
Response code:429
Response message:Too Many Requests


HTTPSampleResult fields:
ContentType: text/html
DataEncoding: null
```

Figure 47: JMeter display of the results

From the results of our stress tests, we realized that Question Service was the first service that started showing signs of being overloaded when we increased the traffic load to simulate 500 users requesting from the service simultaneously. On the other hand, other services were error free when given the same load.

However, while we were able to obtain surface-level results from such a display as it directly tells us the status of the API calls, we felt that more insights could be generated from the data if we could visualize the data better. Therefore, we decided to manually create a PowerBI report to visualize the key data that is returned to us for each microservice from the tests. The figure below shows one such PowerBI report for the Question service. Please refer to Appendix for the PowerBI reports for the other microservices.



Question Service Stress Test

Retrieving one question per user from our database via question-dzqbl74fya-as.a.run.app/?difficulty=medium

Average Data Size (bytes): 2821.26

Number of users: 500
Successful Calls: 484
Calls with Error*: 16
*Error code 429: Too Many Requests

Time Taken to Make all Requests (s): 1.403
Requests / sec: 356.38
Amount of Traffic simulated

Average Time Elapsed (s): 7.888

Average Latency (s): 0.036

Average Connection Time (s): 0.015

Max Data Size (bytes): 5640

Average Data Size (bytes): 2821.26

Min Data Size (bytes): 1571

Figure 48: PowerBI report of the stress test results for Question Service

The PowerBI report was generated after we have cleaned the data and performed relevant transformations and calculations on the data. From this PowerBI report, we could now see that the Question Service was overloaded when we made around 356 API calls per second. From this insight, our team feel that this limitation might be attributed to the CloudRun configurations or resources we are providing for the services, and hence we could start looking at how to more efficiently distribute or allocate such resources when we aim to scale the services. From the report, we could also see meaningful statistics such as the amount of data we are receiving for every call (since each call would return a different question). The graphs in the report also tell us how the service was performing under the load it was given, for instance we can see that the errors started occurring more frequently nearer to the 500th user accessing the service. Hence, through such data visualization, our stress test essentially also helped serve as a load test.

As a result of our stress test, we now know what the upper limits of our system's capacity are, and also how we can increase it, which is probably to first look into scaling up the Question service, which, as mentioned in section 7.4, could be done by changing the autoscaling configurations for question service independently. Apart from scaling up, we also now know which services could be optimized better to better handle an increase in traffic.

9 Reflections

9.1 Suggestions for enhancement

9.1.1 Deployment

Although we have managed to automate the deployment process to GCP, we realized that the automation process has room for improvement. One possibility would be to identify and remove outdated images that are at least 4 revisions earlier than the newly deployed version. Currently, this is possible through direct access to the GCP platform. However, the methods provided for manual update of the registry are tedious and takes up a significant amount of time to run such operations across all deployed microservices, which will not be intuitive if more microservices are to develop in the future. Adopting an automated measure would ensure that the cloud registry would not be unnecessarily storing too many outdated Docker images of each microservices. Moreover, in the event issues occur with the latest deployment and a rollback to the previous image is required, there are sufficient stable images stored that can be used for the rollback, allowing the microservice to remain accessible while allowing the development team space to rectify the issues found with it.

9.1.2 Filtering of Questions by Topics

We could include the capability to include topic as a field for each question into our framework for scraping questions from leetcode. This would enable users to select questions based on specific topics as well. Therefore, within the usual workflow, we envisioned PeerPrep to allow users to first select the question difficulty, followed by the option to filter on a particular subset of topics, before being matched with another user. This might result in another layer of complexity as filtering on a specific set of topics might narrow the search for a match, thus potentially increasing search times and chances of a timeout during the matching process. However, with ample consideration and workarounds for this feature, we believe we could make it work if given more time.

9.1.3 Question Hints

In line with the PeerPoints that players can accumulate, we thought about a future enhancement in future being that they can exchange or “buy” hints with their PeerPoints. This also opens up an avenue for potential monetization of a freemium business model, as we can potentially charge players real money in exchange for PeerPoints as well.

9.1.4 Leaderboard & Achievement Badges

A simple addition could also be that of a leaderboard page, pitting competitive players against each other as they vie for the top spot. This added motivation of competition could potentially add to the gamification aspect and drive these players to work harder during PeerPrep sessions. As for the casual players, their incentive to work harder during the sessions might be to earn awesome looking badges and watch their collection of achievements grow.

9.2 Learning Points

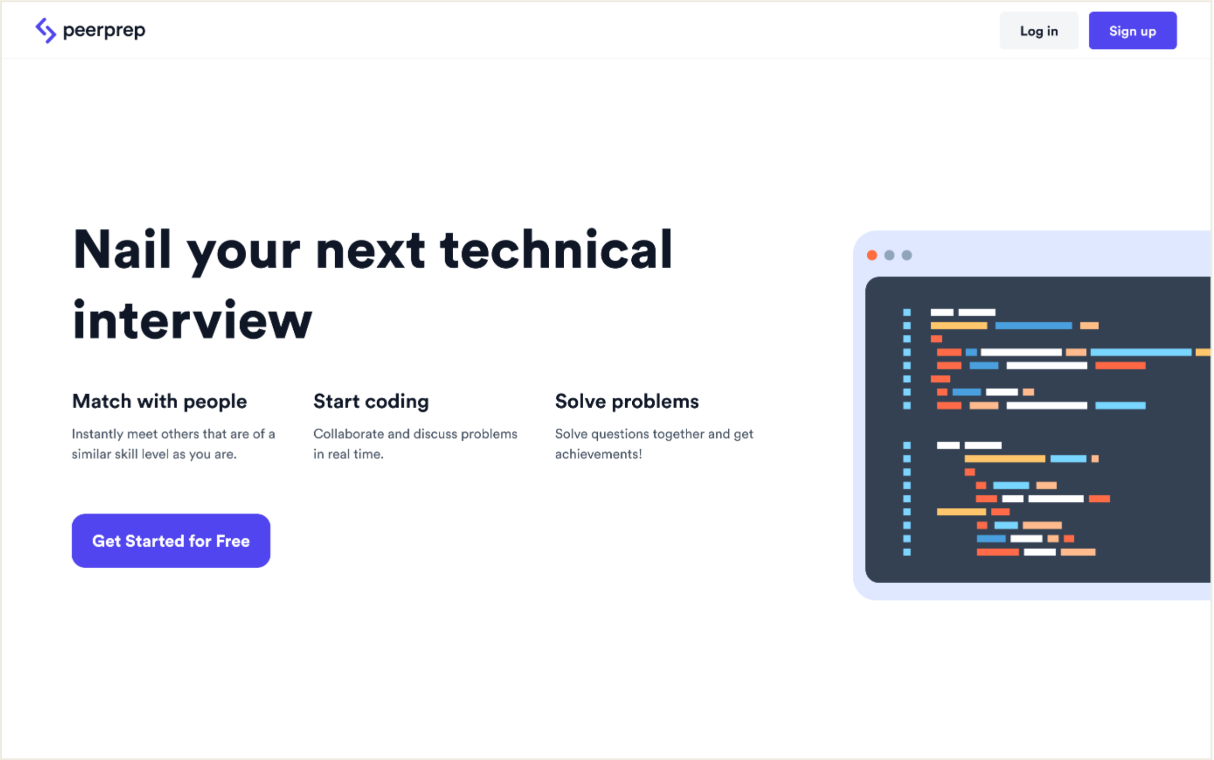
We were exposed to many different design patterns during this module. We learned how to decide on which works best for us and how to implement these design patterns in our project, notably the microservice architecture, Pub-Sub messaging pattern and the mediator pattern. We learned how we could work well as a team when implementing the microservice pattern.

Furthermore, throughout the project process, the team learned the importance of **analysing software requirements** carefully **before planning out the software architecture**, so as to make the most informed design decisions which would simplify the development process. This also entailed exploring multiple tools and platforms that we could leverage on, and analyse the pros and cons for each to select the most suitable tool and platform for the project.

Effective communication was another important soft skill that we have learnt. In a huge software engineering project like PeerPrep, it is important for the team to be able to convey thoughts and ideas among members clearly. With effective communication, we found ourselves to be more efficient and productive during weekly sprints, and the project was completed without major hiccups as a result.

10 Appendix

10.1 Application Screenshots



Nail your next technical interview

Match with people

Instantly meet others that are of a similar skill level as you are.

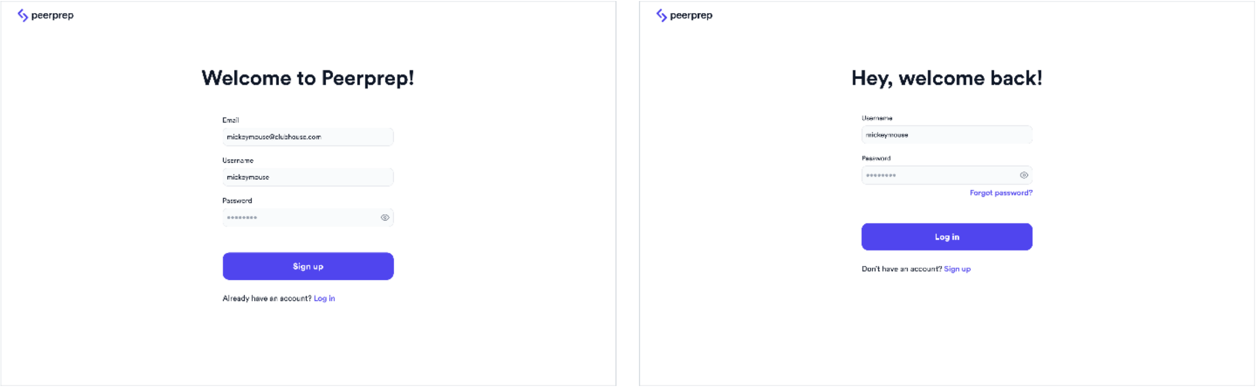
Start coding

Collaborate and discuss problems in real time.

Solve problems

Solve questions together and get achievements!

Figure 49: Landing Page



Welcome to Peerprep!

Email: mickeymouse@clubhouse.com

Username: mickeymouse

Password: ••••••••

Already have an account? Log in

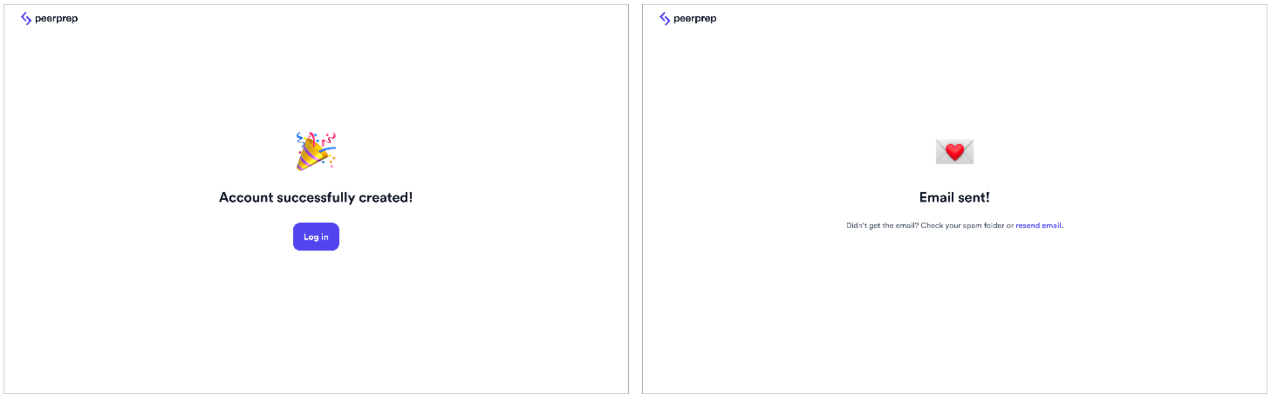
Hey, welcome back!

Username: mickeymouse

Password: ••••••••

Don't have an account? Sign up

Figure 50: Registration and login page

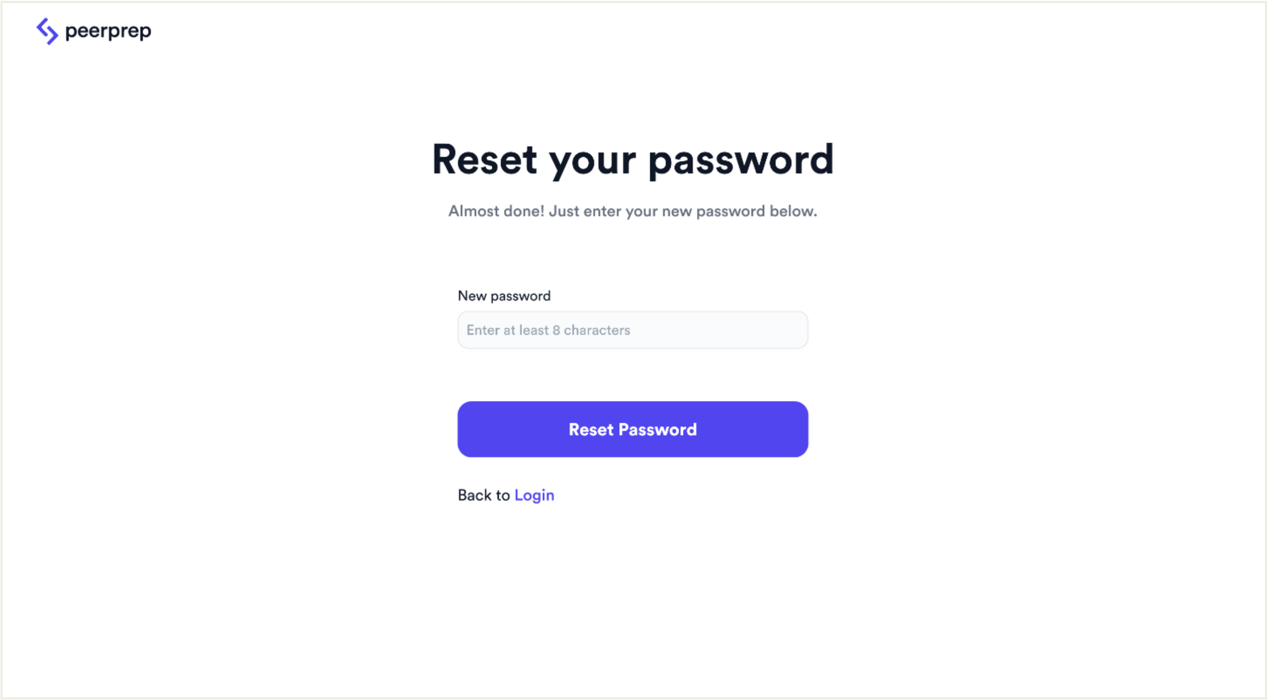


Account successfully created!

Email sent!

Didn't get the email? Check your spam folder or resend email.

Figure 51: Verification pages



Reset your password

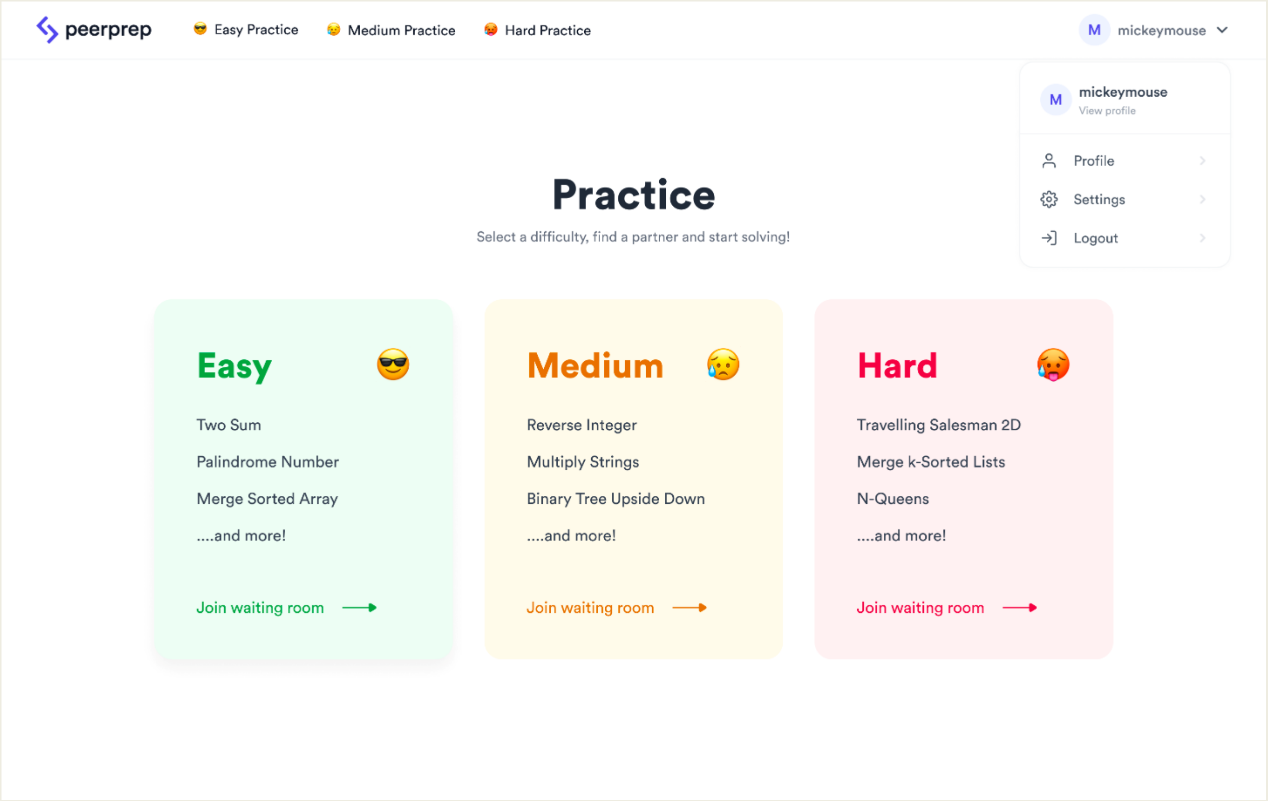
Almost done! Just enter your new password below.

New password

Enter at least 8 characters

Back to Login

Figure 52: Reset password page



Practice

Select a difficulty, find a partner and start solving!

Easy

Two Sum

Palindrome Number

Merge Sorted Array

....and more!

Medium

Reverse Integer

Multiply Strings

Binary Tree Upside Down

....and more!

Hard

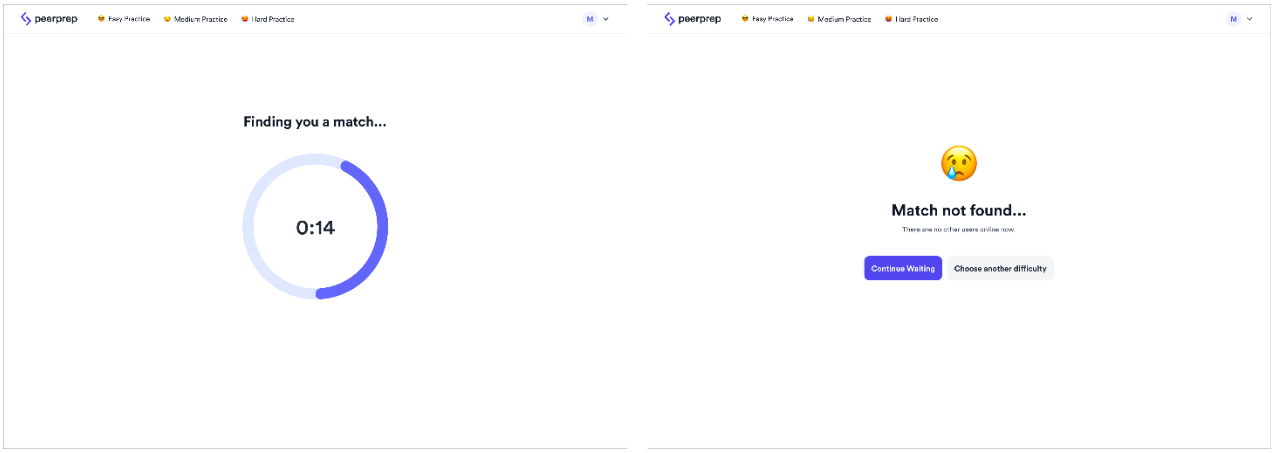
Travelling Salesman 2D

Merge k-Sorted Lists

N-Queens

....and more!

Figure 53: Question difficulty selection page (Home page)

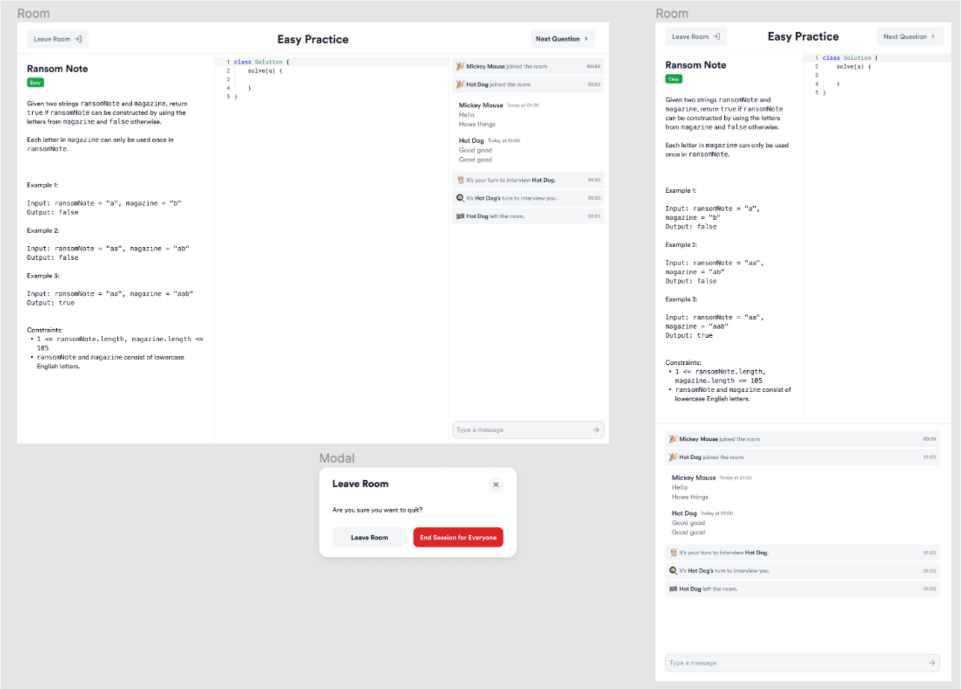


Finding you a match...

Match not found...

There are no other users online now.

Figure 54: Matching and loading page



Easy Practice

Ransom Note

Given two strings `ransomNote` and `magazine`, return `true` if `ransomNote` can be constructed by using the letters from `magazine` and `false` otherwise.

Each letter in `magazine` can only be used once in `ransomNote`.

Example 1:

```
Input: ransomNote = "a", magazine = "b"
Output: false
```

Example 2:

```
Input: ransomNote = "aa", magazine = "ab"
Output: false
```

Example 3:

```
Input: ransomNote = "aa", magazine = "aab"
Output: true
```

Constraints:

- `1 <= ransomNote.length, magazine.length <= 105`
- `ransomNote` and `magazine` consist of lowercase English letters.

```
class Solution {
    solve(s) {

    }
}
```

🎉 **Mickey Mouse** joined the room 00:59

🎉 **Hot Dog** joined the room 01:00

Mickey Mouse Today at 01:00
Hello
Hows things

Hot Dog Today at 01:00
Good good
Good good

It's your turn to interview **Hot Dog.** 01:00

🔍 It's **Hot Dog's** turn to interview you. 01:00

🏁 **Hot Dog** left the room. 01:00

Leave Room

Are you sure you want to quit?

Room

Leave Room

Easy Practice

Next Question

Ransom Note

Easy

Given two strings ransomNote and magazine, return true if ransomNote can be constructed by using the letters from magazine and false otherwise.

Each letter in magazine can only be used once in ransomNote.

Example 1:

Input: ransomNote = "a", magazine = "b"
Output: false

Example 2:

Input: ransomNote = "aa", magazine = "ab"
Output: false

Example 3:

Input: ransomNote = "aa", magazine = "aab"
Output: true

Constraints:
• 1 <= ransomNote.length, magazine.length <= 105
• ransomNote and magazine consist of lowercase English letters.

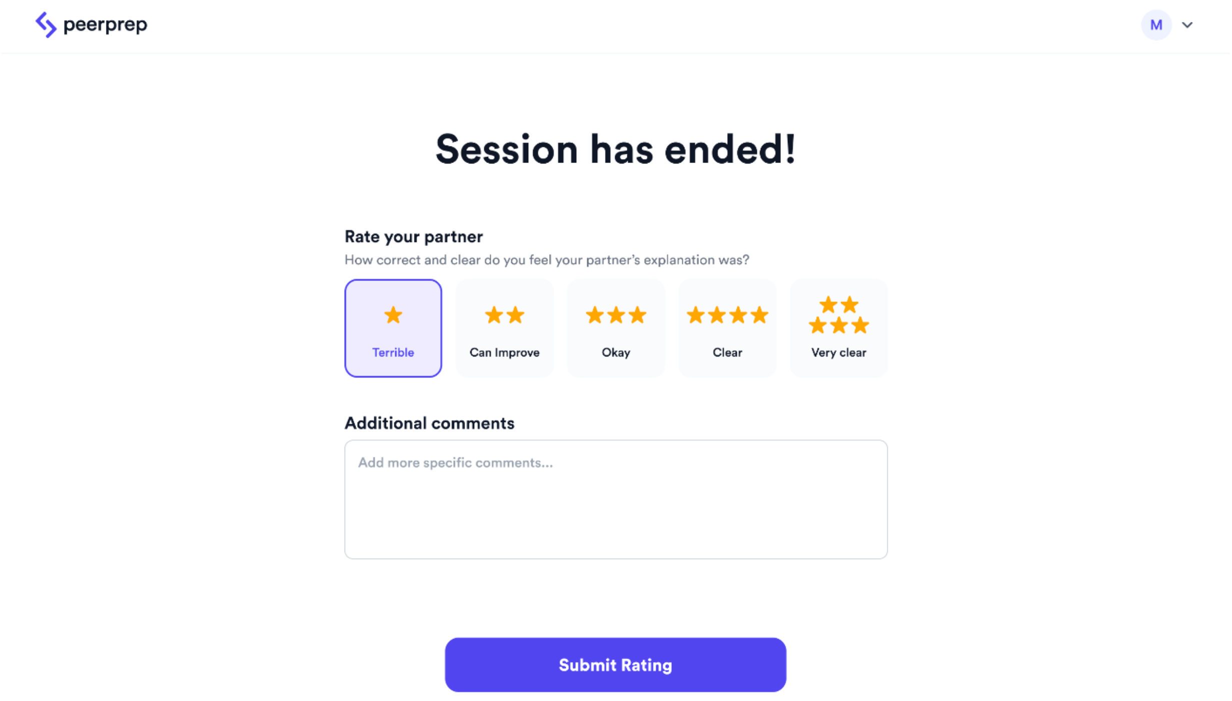
class Solution {
    solve(s) {

    }
}

Mickey Mouse joined the room 00:59
Hot Dog joined the room 01:00
Mickey Mouse Today at 01:00
Hello
Hows things
Hot Dog Today at 01:00
Good good
Good good
It's your turn to interview Hot Dog. 01:00
It's Hot Dog's turn to interview you. 01:00
Hot Dog left the room. 01:00

Type a message

Figure 55: Responsiveness of the app



Session has ended!

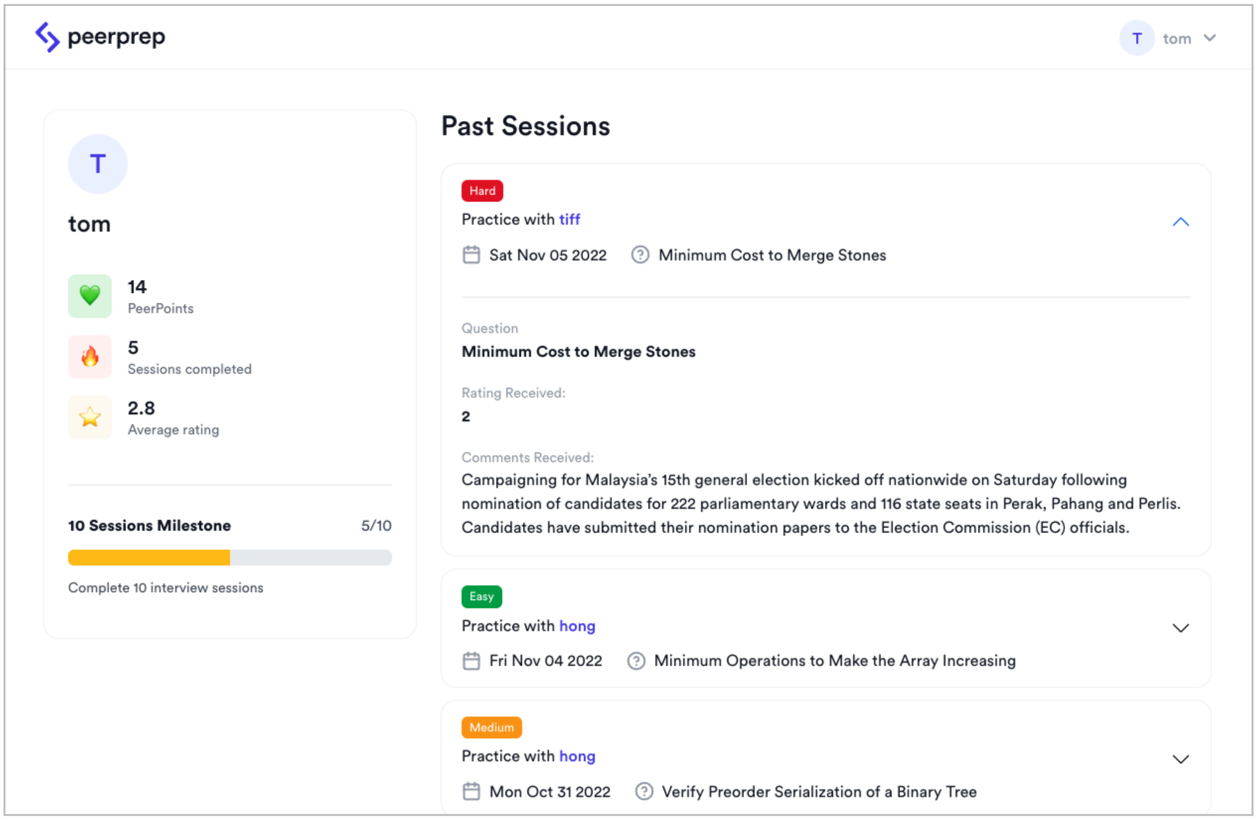
Rate your partner

How correct and clear do you feel your partner's explanation was?

[✓ ★ Terrible] ★★ Can Improve ★★★ Okay ★★★★ Clear ★★★★★ Very clear

Additional comments

Figure 56: Feedback form



T

tom

💚 14
PeerPoints

🔥 5
Sessions completed

⭐ 2.8
Average rating

10 Sessions Milestone 5/10

Complete 10 interview sessions

Past Sessions

Hard

Practice with tiff

Sat Nov 05 2022 Minimum Cost to Merge Stones

Question

Minimum Cost to Merge Stones

Rating Received:

2

Comments Received:

Campaigning for Malaysia's 15th general election kicked off nationwide on Saturday following nomination of candidates for 222 parliamentary wards and 116 state seats in Perak, Pahang and Perlis. Candidates have submitted their nomination papers to the Election Commission (EC) officials.

Easy

Practice with hong

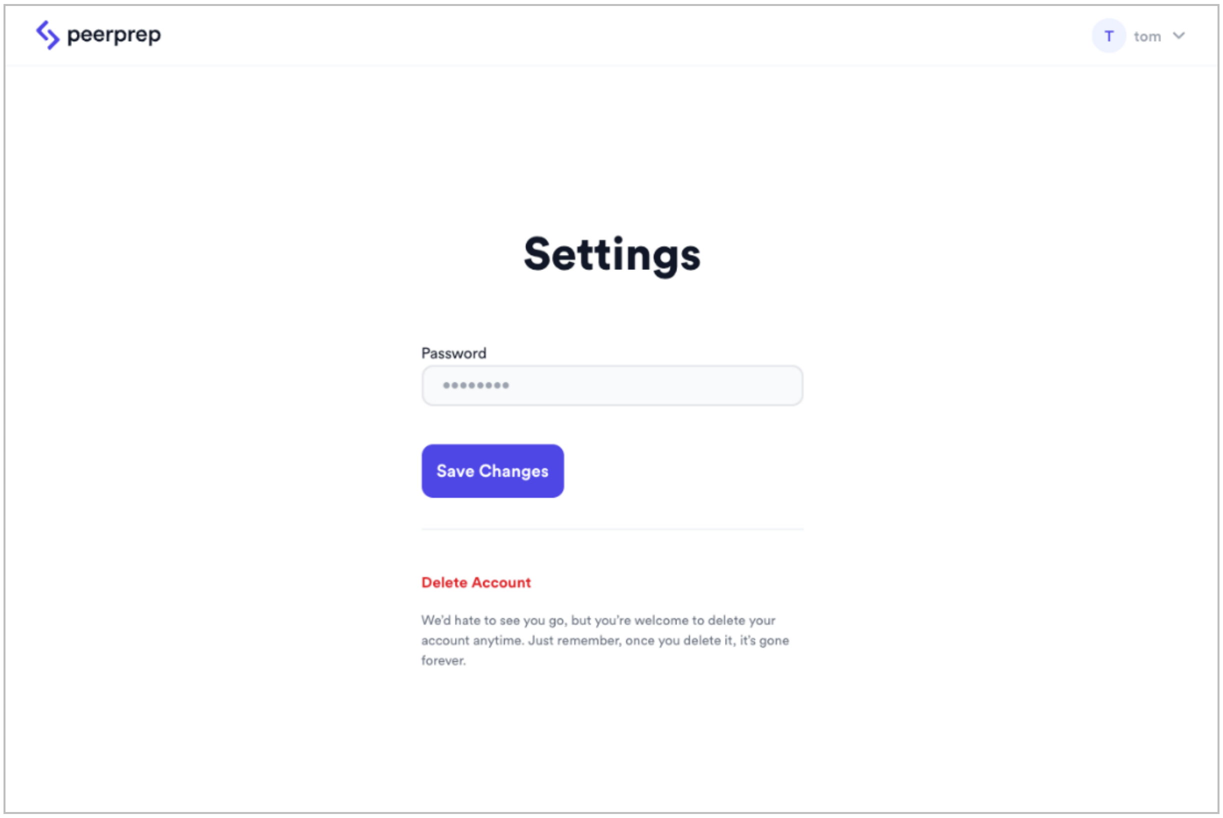
Fri Nov 04 2022 Minimum Operations to Make the Array Increasing

Medium

Practice with hong

Mon Oct 31 2022 Verify Preorder Serialization of a Binary Tree

Figure 57: Profile page



Settings

Password

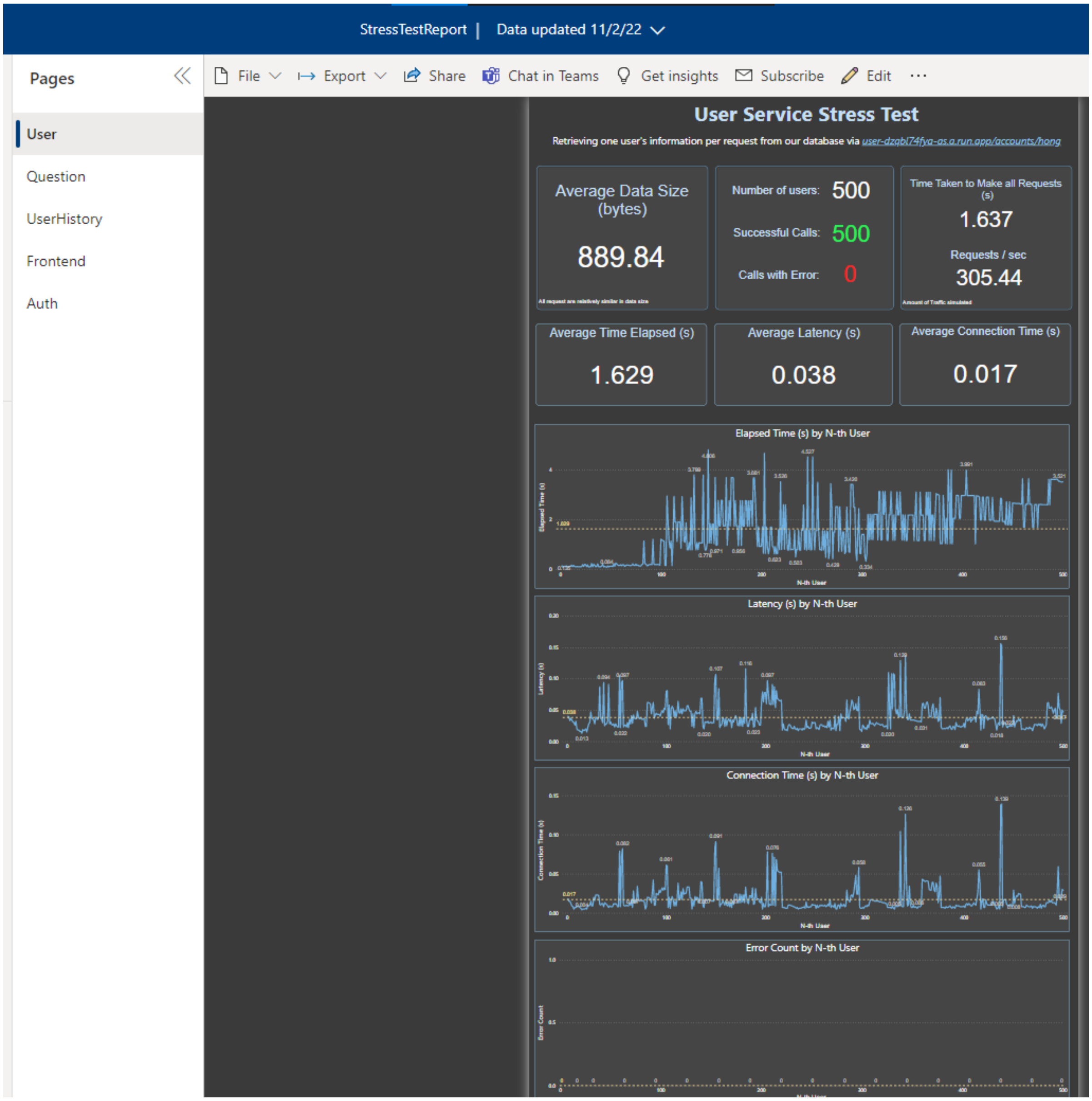
••••••••

Delete Account

We'd hate to see you go, but you're welcome to delete your account anytime. Just remember, once you delete it, it's gone forever.

Figure 58: Settings page

10.2 Screenshots of Stress Test Results



User Service Stress Test

Retrieving one user's information per request from our database via user-dzqbl74fya-as.a.run.app/accounts/hong

Average Data Size (bytes): 889.84

All request are relatively similar in data size

Number of users: 500

Successful Calls: 500

Calls with Error: 0

Time Taken to Make all Requests (s): 1.637

Requests / sec: 305.44

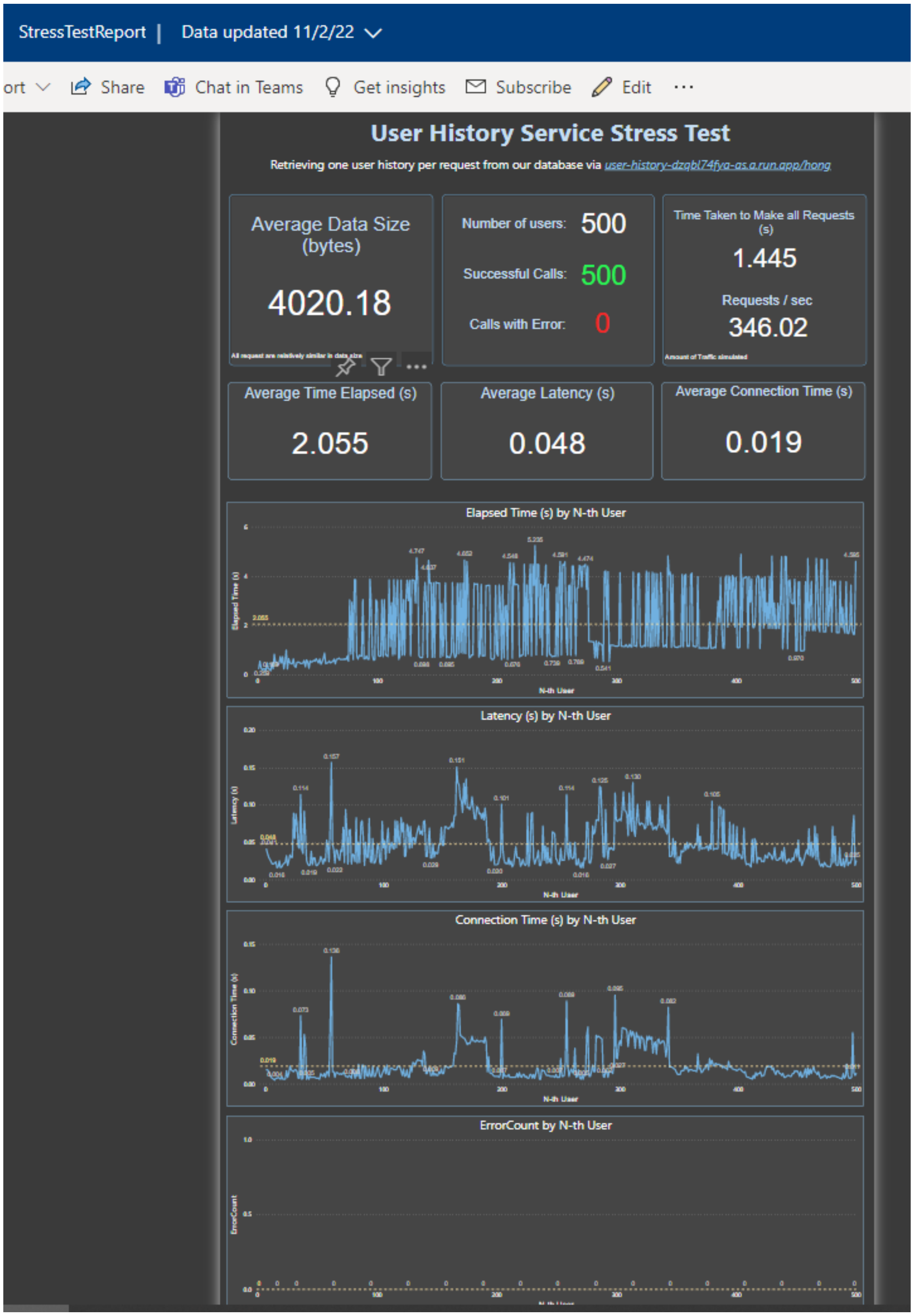
Amount of Traffic simulated

Average Time Elapsed (s): 1.629

Average Latency (s): 0.038

Average Connection Time (s): 0.017

Figure 59: User Service stress test report



StressTestReport | Data updated 11/2/22

User History Service Stress Test

Retrieving one user history per request from our database via user-history-dzqbl74fya-as.a.run.app/hong

Average Data Size (bytes)
4020.18
All request are relatively similar in data size

Number of users: 500
Successful Calls: 500
Calls with Error: 0

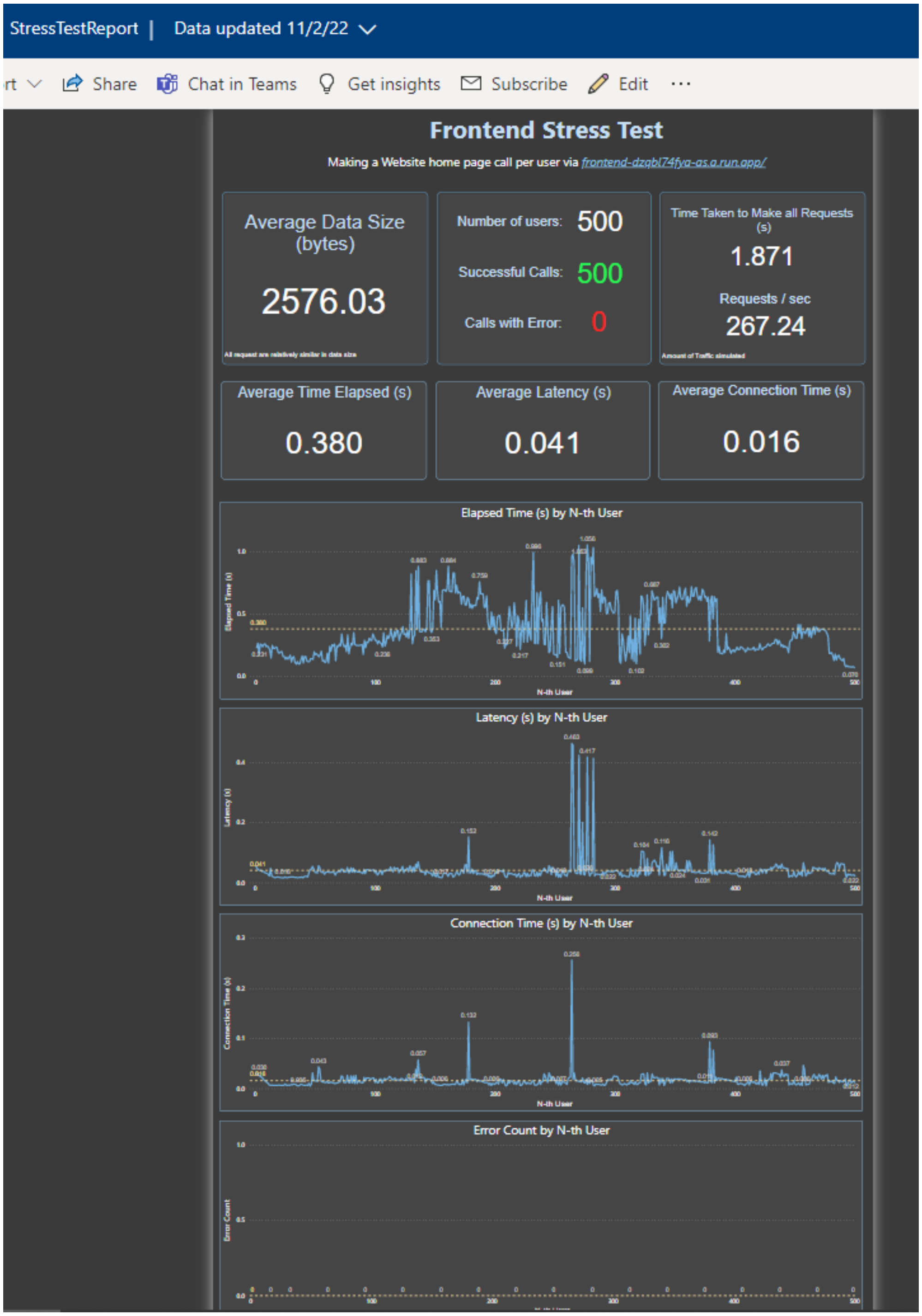
Time Taken to Make all Requests (s)
1.445
Requests / sec
346.02
Amount of Traffic simulated

Average Time Elapsed (s)
2.055

Average Latency (s)
0.048

Average Connection Time (s)
0.019

Figure 60: User History Service stress test report



Frontend Stress Test

Making a Website home page call per user via *frontend-dzqbl74fya-as.a.run.app/*

Average Data Size (bytes)

2576.03

All request are relatively similar in data size

Number of users: 500

Successful Calls: 500

Calls with Error: 0

Time Taken to Make all Requests (s)

1.871

Requests / sec

267.24

Amount of Traffic simulated

Average Time Elapsed (s)

0.380

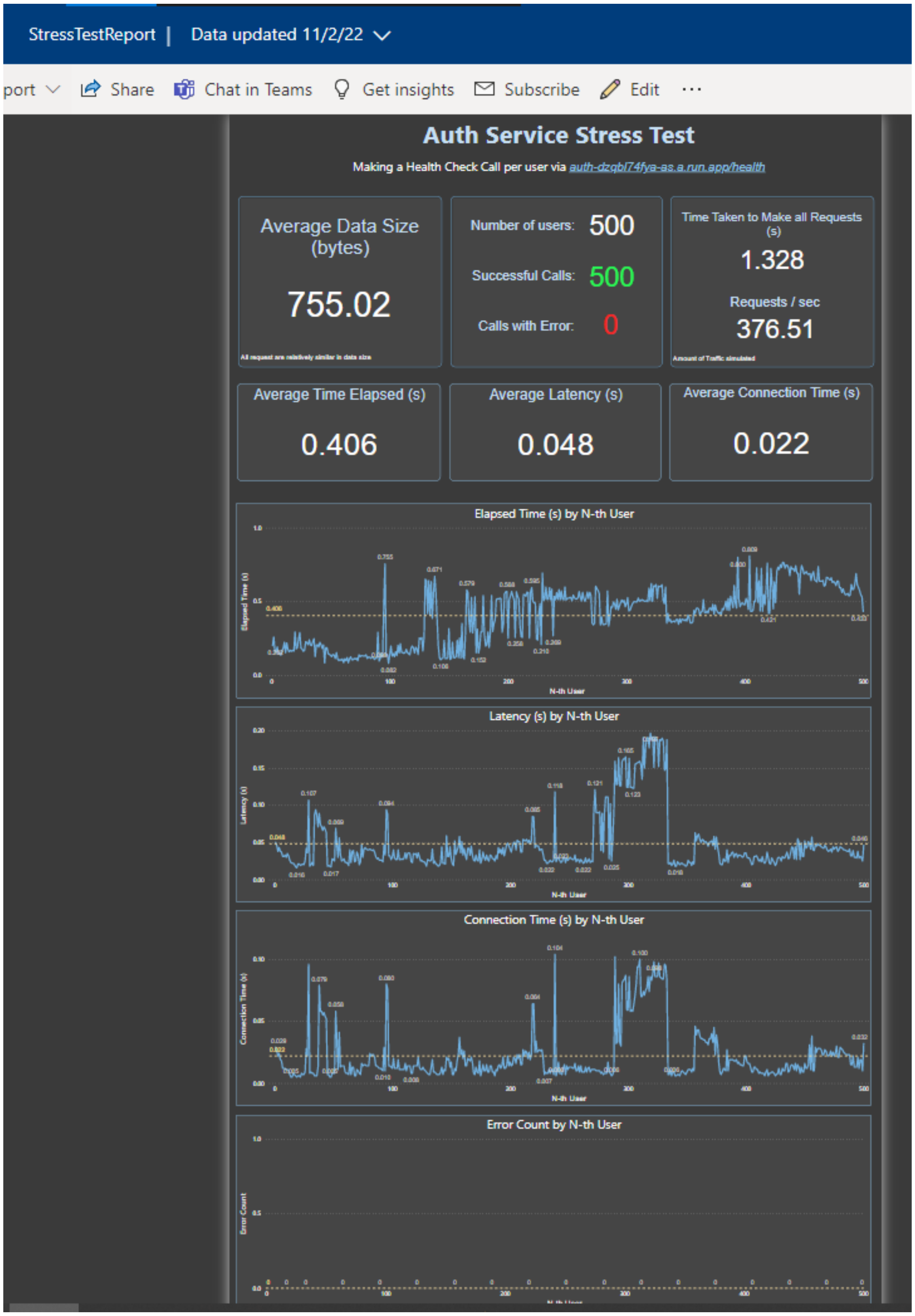
Average Latency (s)

0.041

Average Connection Time (s)

0.016

Figure 61: Frontend Service stress test report



Auth Service Stress Test

Making a Health Check Call per user via auth-dzqbl74fya-as.a.run.app/health

Average Data Size (bytes)
755.02
All request are relatively similar in data size

Number of users: 500
Successful Calls: 500
Calls with Error: 0

Time Taken to Make all Requests (s)
1.328
Requests / sec
376.51
Amount of Traffic simulated

Average Time Elapsed (s)
0.406

Average Latency (s)
0.048

Average Connection Time (s)
0.022

Figure 62: Auth service stress test report

10.3 Project Timeline

Week	Description
3	● Set up skeleton code ● Frontend design template (Figma)
4	● User service o Create user DB o Hash salt password o Create controller for User service o Create model for User service ● Frontend service o Create view for User service o Create view for Collaboration service
5	● Collaboration service o Create matching DB o Create controller for Collaboration service o Create model for Collaboration service o Implement Socket IO for matching ● User service o Implement email service o Generate JWT token ● Frontend service o Integrate User service o Set up protected routes and layouts
6	**Milestone 1** ● Collaboration service o Implement Chat feature o Implement text editing collaboration feature ● Frontend service o Integrate User service signup flow o Integrate Collaboration service o Create Forgot & Reset password feature
7	● Create Docker-Compose config for local orchestration ● Add CI/CD frameworks to trigger on changes to target microservices

	● User service o Handle access and refresh token o Integrate Forgot & Reset password feature ● Collaboration service o Implement Video Chat feature ● Question service o Create question DB o Create controller for Question service o Create model for Question service ● Frontend o Integrate Video Chat feature
8	● Deploy services to GCP ● Update CD workflows to connect to GCP ● Frontend o Create view for Question service o Integrate Question service ● Begin project report
9	**Milestone 2** ● Create API gateway for non-frontend microservices ● User History service o Create user history DB o Create controller for user history o Create model for user history ● Frontend o Create view for User History service o Integrate User History service ● Work on project report
10	● Create API gateway for microservices ● Implement CI on GitHub Actions ● User History service o Create rating system for user ● Work on Project Report
11	● Create API gateway for microservices ● Frontend service o Create view of rating feature ● Work on Project Report

	● Work on Presentation Slides
12	● Create API gateway for microservices ● Complete Project Report ● Complete Presentation Slides
13	**Milestone 3** ● Demo

10.4 Project Setup and Access

10.4.1 Deployment

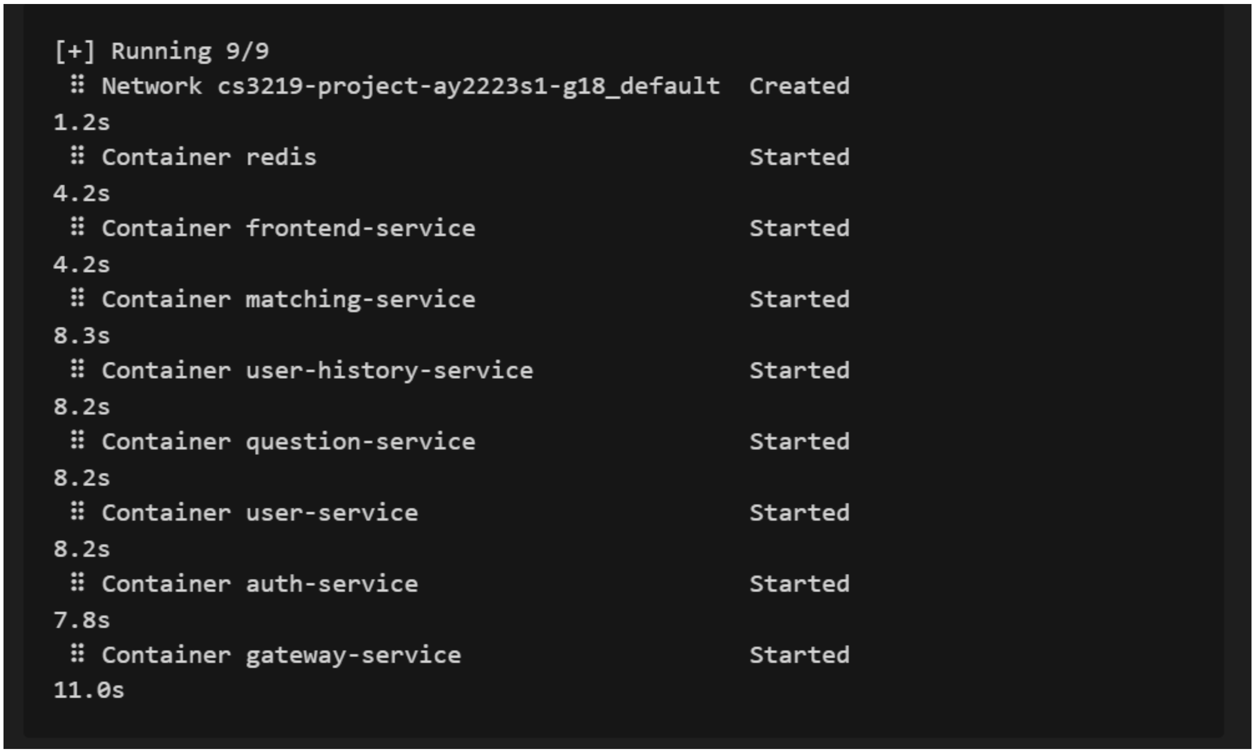
Navigate to https://frontend-dzqbl74fya-as.a.run.app/ to use PeerPrep online.

10.4.2 Local Orchestration

1. Ensure that you have Docker installed on your local machine.
2. Copy the env variable snippets in Environment Variables to the indicated directories as .env.prod
3. In the main directory where docker-compose.yml is located, run the following command:

```
docker compose up -d
```

4. If the services are working properly, all the services should be started as shown below.



```
[+] Running 9/9
 ⠿ Network cs3219-project-ay2223s1-g18_default  Created
1.2s
 ⠿ Container redis                              Started
4.2s
 ⠿ Container frontend-service                   Started
4.2s
 ⠿ Container matching-service                   Started
8.3s
 ⠿ Container user-history-service               Started
8.2s
 ⠿ Container question-service                   Started
8.2s
 ⠿ Container user-service                       Started
8.2s
 ⠿ Container auth-service                       Started
7.8s
 ⠿ Container gateway-service                    Started
11.0s
```

Figure 63: Successful Docker Compose Operation

5. Navigate to http://localhost:3000/ to access the frontend service.
6. Navigate to http://localhost:80/ to access the backend service.

10.4.3 Environment Variables

10.4.3.1 User Service

```
- Location: ./user-service
DB_CLOUD_URI="<insert_uri_here>"
DB_LOCAL_URI="mongodb://mongodb:27017"
ENV="PROD"
REFRESH_TOKEN_SECRET="<insert_generated_token>"
ACCESS_TOKEN_SECRET="<insert_generated_token>"
VERIFICATION_TOKEN_SECRET="<insert_generated_token>"
REFRESH_TOKEN_EXPIRY="12h"
ACCESS_TOKEN_EXPIRY="30m"
VERIFICATION_TOKEN_EXPIRY="15m"
EMAIL_HOST='smtp.gmail.com'
EMAIL_USERNAME='<insert_email>'
EMAIL_PASSWORD='<insert_api_key>'
```

10.4.3.2 User History Service

```
- Location: ./user-history-service
DB_CLOUD_URI="<insert_uri_here>"
DB_LOCAL_URI="mongodb://mongodb:27017"
ENV="PROD"
```

10.4.3.3 Question Service

```
- Location: ./question-service
DB_CLOUD_URI="<insert_uri_here>"
DB_LOCAL_URI="mongodb://mongodb:27017"
ENV="PROD"
```

10.4.3.4 Gateway Service

```
- Location: ./gateway-service
AUTH_SERVICE_URL = http://auth-service:7000
USER_SERVICE_URL = http://user-service:8000
MATCHING_SERVICE_URL = http://matching-service:8001
QUESTION_SERVICE_URL = http://question-service:8002
USER_HISTORY_SERVICE_URL = http://user-history-service:8003
```

10.4.3.5 Authentication Service

```
- Location: ./gateway-service
REDIS_URI="redis://redis:6379"
REFRESH_TOKEN_SECRET="<insert_generated_token>"
```

```
ACCESS_TOKEN_SECRET="<insert_generated_token>"
VERIFICATION_TOKEN_SECRET="<insert_generated_token>"
REFRESH_TOKEN_EXPIRY="12h"
ACCESS_TOKEN_EXPIRY="30m"
VERIFICATION_TOKEN_EXPIRY="15m"
```

10.4.3.6 Frontend Service

```
- Location: ./frontend
REACT_APP_API_SVC = 'http://localhost:80'
REACT_APP_IS_USING_GATEWAY = 'true'
```