PeerPrep

CS3219 Software Engineering Principles and Patterns

Team 34 Project Report

Repo: https://github.com/CS3219-AY2223S1/cs3219-project-ay2223s1-g34

Member	Muhammad Assyarul Ariffin Bin Omar	Cheong Ying Yi Clara	Hogan Tan Shao Han	Roberto Morassi del Blanco
Student Number	A0196529M	A0206330X	A0201725R	A0265239X

Table of Content

1. Individual Contributions

Each member contributed to different aspects of the project. The following table contains more details.

Team Member	Contributions
Muhammad Assyarul Ariffin Bin Omar	Technical: ● Implement the Collaboration Service. ● Implement the initial foundation of Code Editor. ● Implement the Chat Room function in Collaboration Service. Non-Technical: ● Project Report 　○ NFR/FR for Collaboration Service 　○ Creating the Prioritisation Matrix 　○ Creating Use Cases for Collab Service ● Document weekly minutes
Cheong Ying Yi Clara	Technical: ● Implement Matching Service ● Implement notification in Chat for User left and matching successful by invitation Non-Technical: ● Project Report 　○ Functional and Non-Functional Requirements for Matching Service 　○ Features for Matching Service 　○ Design and Application Flow for Matching Service
Hogan Tan Shao Han	Technical: ● Implement User Service ● Implement Code Editor Syntax in Collaboration Service ● Frontend Design and Implementation ● Integrate CI/CD ● Deploy on GCP ● Set up Database Non-Technical: ● Handled GitHub Project and Issues ● Project Report 　○ Sections involving the above technical aspects and remaining components not covered by other members
Roberto Morassi del Blanco	Technical: ● Implement the Question Service ● Testing for Question Service ● Continuous Integration extension Non-Technical:

	● Project Report ○ Functional Requirements and Non-Functional Requirements for Question Service ○ Features for Question Service ○ Various diagrams for Question Service

2. Introduction

2.1 Motivation

As the process of obtaining a tech role/job usually requires applicants to go through some form of coding assessment, more and more applicants are inclined to embark on the “Leetcode grind” to prepare for such assessments.

However, some applicants do face daunting issues when sitting for these assessments despite practising coding questions by themselves. 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 alone. Moreover, grinding practice questions alone can be tedious and monotonous.

2.2 Background and Scope

To solve this issue, PeerPrep provides an interview preparation platform and peer matching where students can find peers to practise whiteboard-style interview questions together. Users will be able to match quickly, communicate with another to solve these questions together and form the solutions that both users are able to use in the end. With PeerPrep, we hope to provide a platform for applicants to be able to work together and help each other to beat the “Leetcode grind”!

3. Requirements Specification

This section describes the various requirements we have identified as well as the relevant justifications of these requirements.

3.1 Functional Requirements

This section specifies the various functional requirements of each feature domain of Peerprep. Each functional requirement is linked to a user case which can be found in the Appendix section.

3.1.1 User Service

S/N	Functional Requirement	Priority	Use Case
FR-U-01	The system should allow users to create an account with username, email address and password.	High	UCU-01
FR-U-02	The system should ensure that every account created has a unique username and email address.	High	UCU-01
FR-U-03	The system should allow users to log into their accounts by entering their email and password.	High	UCU-02
FR-U-04	The system should allow users to log out of their account.	High	UCU-03
FR-U-05	The system should allow users to delete their account so that users clear their data from the system.	High	UCU-04
FR-U-06	The system should generate a token to verify and authenticate a signed in user so that the system will be able to determine whether the user is authenticated when accessing other sensitive features.	High	UCU-02
FR-U-07	The system should allow users to change their password so that users can change their password if they feel that their current password is not secure enough or have been compromised.	Medium	UCU-05
FR-U-08	The system should allow them to reset their password so that users can recover their account even if they forget their password.	Low	UCU-06
FR-U-09	The system should verify that registered emails are verified so as to prevent users from using bogus email addresses to create accounts.	Low	UCU-01

3.1.2 Matching Service

S/N	Functional Requirement	Priority	Use Case
FR-M-01	The system should allow users to select the difficulty level of the questions they wish to attempt.	High	UC-M-01
FR-M-02	The system should be able to match two waiting users with similar difficulty levels and/or topics and put them in the same room.	High	UC-M-01
FR-M-03	If there is a valid match, the system should match the users within 30s.	High	UC-M-01 UC-M-02
FR-M-04	The system should inform the users that no match is available if a match cannot be found within 30 seconds.	High	UC-M-01 UC-M-02
FR-M-05	The system should allow users to select topics of the questions they wish to attempt.	Medium	UC-M-01
FR-M-06	The system should allow users to cancel matching while waiting for a match.	Medium	UC-M-01 UC-M-02
FR-M-07	If a match was not found, the system should prompt the user to rematch.	Low	UC-M-01 UC-M-02
FR-M-08	The system should provide a means for the user to leave a room once matched.	Low	UC-M-01 UC-M-02
FR-M-09	The system should allow users to invite a specific user using the user’s email.	Low	UC-M-01
FR-M-10	The system should be able to match the user and invited user in the same room.	Low	UC-M-01 UC-M-02
FR-M-11	The system should inform the user that the invitation was successful.	Low	UC-M-01

3.1.3 Question Service

S/N	Functional Requirement	Priority	Use Case
FR-Q-01	The system should return the relevant information of a random question such as question contents and	High	UCQ-01

	metadata, when given a question topic and difficulty.		
FR-Q-02	The system should return the relevant information of a specific question such as question contents and metadata, when given a QID.	High	UCQ-01
FR-Q-03	The system should return the relevant information of a list of questions (such as question contents and metadata) that satisfy a particular set of filters.	Medium	UCQ-02

3.1.4 Collaboration Service

S/N	Functional Requirement	Priority	Use Case
FR-C-01	The system should connect the matched users to the same session.	High	UCC-03
FR-C-02	The system should end the session when one of the users presses the 'Finish' button.	High	UCC-04
FR-C-03	The system should update the content of the free text field in real-time when any one of the users modifies it.	High	-
FR-C-04	The system provides (and displays) the two users in the same session the same question.	High	-
FR-C-05	The system should provide a separate chat box for the two users to communicate with each other	Medium	-

3.2 Non-Functional Requirements

This section specifies the various non-functional requirements of each feature domain of Peerprep.

3.2.1 User Service

S/N	Non Functional Requirement	Priority
NFR-U-02	Users should be only able to change their passwords/delete their accounts only if they are authenticated so that the security of user accounts are preserved.	High
NFR-U-03	Users' passwords should be at least 8 characters long	Medium

	so that passwords are reasonably secure from brute force attacks.	
NFR-U-04	Users' username should be at least 6 characters long and at most 18 characters so that usernames are not not too short or long which could disrupt displaying them on the frontend.	Medium
NFR-U-05	Users' passwords should be hashed and salted before storing in the DB so that passwords are securely stored in the database and that only the user is only aware of the actual password to their account.	Medium

3.2.2 Matching Service

S/N	Non Functional Requirement	Priority
NFR-M-01	A user should be able to submit a matching request within 2 minutes.	Medium
NFR-M-02	The matching system should be able to support at least 100 concurrent matching requests at any instance.	Medium
NFR-M-03	The matching system should be available 99% of the time.	Medium

3.2.3 Question Service

S/N	Functional Requirement	Priority
NFR-Q-01	Each question should have a unique Question Identification (QID) number.	High
NFR-Q-02	Questions should meet specific criteria. For instance its contents cannot be shorter than the minimum length and it should be assigned a specific difficulty level and topic.	High
NFR-Q-03	In the question bank, each difficulty level and topic should have at least one question. As such if there are 3 difficulties and 3 topics, there should be a minimum of 3*3 = 9 questions.	High

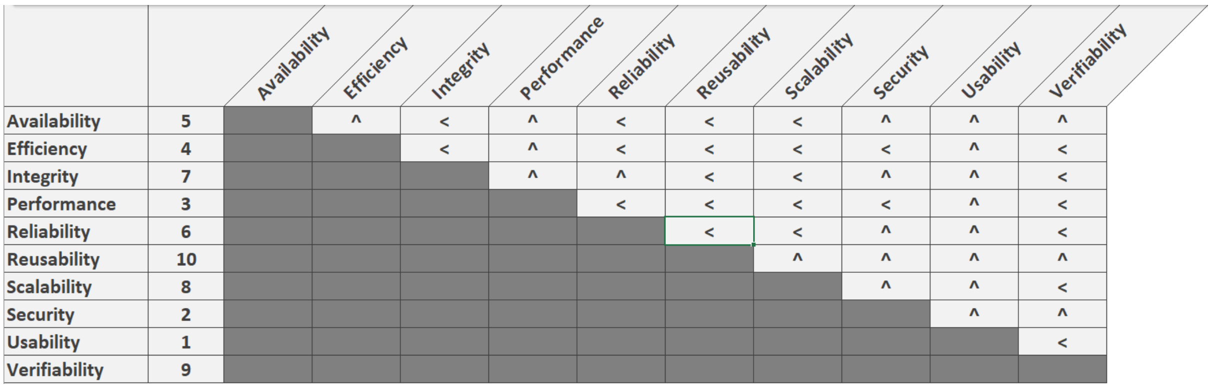
3.2.4 Collaboration Service

S/N	Functional Requirement	Priority
NFR-C-01	Any changes done by the user on the free text field should be updated to the other user within 0.1s	High
NFR-C-02	Any messages entered by a user should be updated on the other user's window within 0.2s	Medium

3.2.5 Frontend

S/N	Functional Requirement	Priority
NFR-F-01	Intuitive and user friendly UI such that the application is easy to navigate through and use.	High
NFR-F-02	Minimalistic design such that users will not be distracted by other elements. This allows for the application to be easily navigated.	Medium

3.3 Quality Attributes Prioritisation Matrix



		Availability	Efficiency	Integrity	Performance	Reliability	Reusability	Scalability	Security	Usability	Verifiability
Availability	5	▨	^	<	^	<	<	<	^	^	^
Efficiency	4	▨	▨	<	^	<	<	<	<	^	<
Integrity	7	▨	▨	▨	^	^	<	<	^	^	<
Performance	3	▨	▨	▨	▨	<	<	<	<	^	<
Reliability	6	▨	▨	▨	▨	▨	<	<	^	^	<
Reusability	10	▨	▨	▨	▨	▨	▨	^	^	^	^
Scalability	8	▨	▨	▨	▨	▨	▨	▨	^	^	<
Security	2	▨	▨	▨	▨	▨	▨	▨	▨	^	^
Usability	1	▨	▨	▨	▨	▨	▨	▨	▨	▨	<
Verifiability	9	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨

Quality Attributes Prioritisation Matrix

Based on the prioritisation matrix above, the team has decided to focus on the top 3 quality attributes: usability, performance and security.

Firstly, the motivation for prioritising usability is that the team hopes that the application takes minimal effort to understand and use. By making the application easy to use, it will be more welcoming to new users as well as being able to retain existing users as well.

Moreover, usability also includes placing great importance in providing effective feedback to users when an error occurs. For example, if a user inputs a wrong password when signing in, the system would inform the user of the error in a comprehensible and informative manner.

Secondly, the focus on performance is imperative because PeerPrep is a collaborative web application where two users are interacting with each other in real time. In order to provide real time communication, the team seeks to ensure that latency between users are kept to a minimum. This way the user experience will be much better as users will not have to expect long delays between their communication.

Lastly, the aspect of security is also of high priority to us because our application does handle sensitive personal data which should not be available to unwanted parties. Therefore, the team has to ensure that we handle sensitive data securely, specifically in the frontend as it is the component that connects all the microservices together.

3.4 Features

PeerPrep provides 4 main feature domains which are elaborated briefly below.

3.4.1 User Service

The User Service feature domain provides the various features as follow:

Feature	Description
Account Creation	- Allows users to create an account with a unique username, email and password
Account Verification	- Ensures that only verified emails are able to create an account on PeerPrep
Sign In / Sign Out	- Generates an authentication token when the user signs in - Blacklist the generated authenticated token when the user signs out
Changing of Account Password	- Allow users to change their existing password through PeerPrep
Resetting of Account Password	- Allow users to reset their account password in the case that users forget their password. - This is done through the registered email of the account.
Account Deletion	- Deletes the account from the PeerPrep system

3.4.2 Matching Service

The Matching Service feature domain provides the various features as follow:

Feature	Description
Match user by Difficulty Level and Topic	- Allows users to find a matching user based on the same difficulty level and topic
Invite user	- Allows users to invite another specific user by email

3.4.3 Question Service

Feature	Description
Specific Question fetching	- Allows the system to fetch a specific question (with specific Question ID)
Random Question fetching	- Allows the system to fetch a random question that matches a difficulty and topic
Filtered Question fetching	- Allows the system to fetch a set of questions that match a particular set of filters. Filters can be any attribute of the question including the topic, QID, difficulty, etc..

3.4.4 Collaboration Service

Feature	Description
Chat Room	- Allows the two users to communicate between one another easily. This nice-to-have feature is essential for users to have an easier time to collaborate.
Code Editor	- Allows the two users to edit the same code on a single code editor where it is being updated in real time anytime the code changes.

4. Software Development Process

This section describes the Software Development Lifecycle and the project schedule of the team's development of the PeerPrep project.

4.1. Software Development Lifecycle

Our team follows an AGILE development process. Our weekly workflow is that of a typical AGILE workflow as follows:

1. Weekly meetings on Mondays 10am
2. Each member will share what they have accomplished for the previous week and whether they are able to meet the targets set and to surface any issues encountered which were noted down via GitHub Issues
3. Discuss and set what needs to be done by the next meeting. Deliverables are tracked using GitHub Projects and Issues.
4. Weekly minutes for meetings are recorded for reference
5. Spend the next few days till the next meeting implementing and working on the goals set for the week
6. If there any new additions to the codebase, members are expected to do a pull requests which requires one member to approve, the pull request is tested before any approval is done to protect the main branch
7. After being approved, the pull request will be merged into the main branch
8. In our development process, the use of Continuous Integration / Deployment via GitHub Actions helps us to check for compile/build errors on the main branch so that we can ensure that the application built on main is deployable.

4.2 Project Plan

The team has decided to follow an iterative breadth-first approach of development where tasks are distributed across multiple milestones within each iteration. An iterative approach was chosen as it allows for rapid feedback about the progress of various components and also enables the team to adjust development plans based on issues and opportunities discovered at each milestone. A breadth-first approach was chosen as it allowed everyone to concurrently work on all components, allowing for rapid development.

<u>Schedule</u>

Period: Week 3 to Week 4 (22 Aug 2022 to 26 Aug 2022)

1. Understand project requirements
2. Familiarise with PeerPrep skeleton code
3. Research on Tech Stack to use
4. Come up with Functional and Nonfunctional Requirements

Period: Week 4 to Week 5 (29 Aug 2022 to 2 Sep 2022)

1. Implement basic must-have backend features
    a. User Service

    b. Matching Service
    c. Question Service
    d. Collaboration Service

Period: Week 5 to Week 6 ***(Milestone 1)*** *(5 Sep 2022 to 9 Sep 2022)*

1. Polish must-have features
    a. User Service
    b. Matching Service
    c. Question Service
    d. Collaboration Service
2. Implement frontend interface

Period: Week 7 to Week 8 (26 Sep 2022 to 1 Oct 2022)

– Break–

Period: Week 8 to Week 9 (3 Oct 2022 to 7 Oct 2022)

1. Implement nice-to-have improvements for must-have features
    a. Chat Feature for Session Page
2. Polish frontend UI interface

Period: Week 9 to Week 10 ***(Milestone 2)*** *(10 Oct 2022 to 14 Oct 2022)*

1. Start working on project report
2. System/Integration testing of application

Period: Week 10 to Week 11 (17 Oct 2022 to 21 Oct 2022)

1. Continue working on project report
2. Integrate CI/CD to the project
3. Dockerizing the application
4. System testing the application

Period: Week 11 to Week 12 (24 Oct 2022 to 28 Oct 2022)

1. Continue working on project report
2. Dockerizing the application
3. Integrate CI/CD to the project
4. Deploying application
5. Testing the application

Period: Week 12 to Week 13 (31 Oct 2022 to 4 Nov 2022)

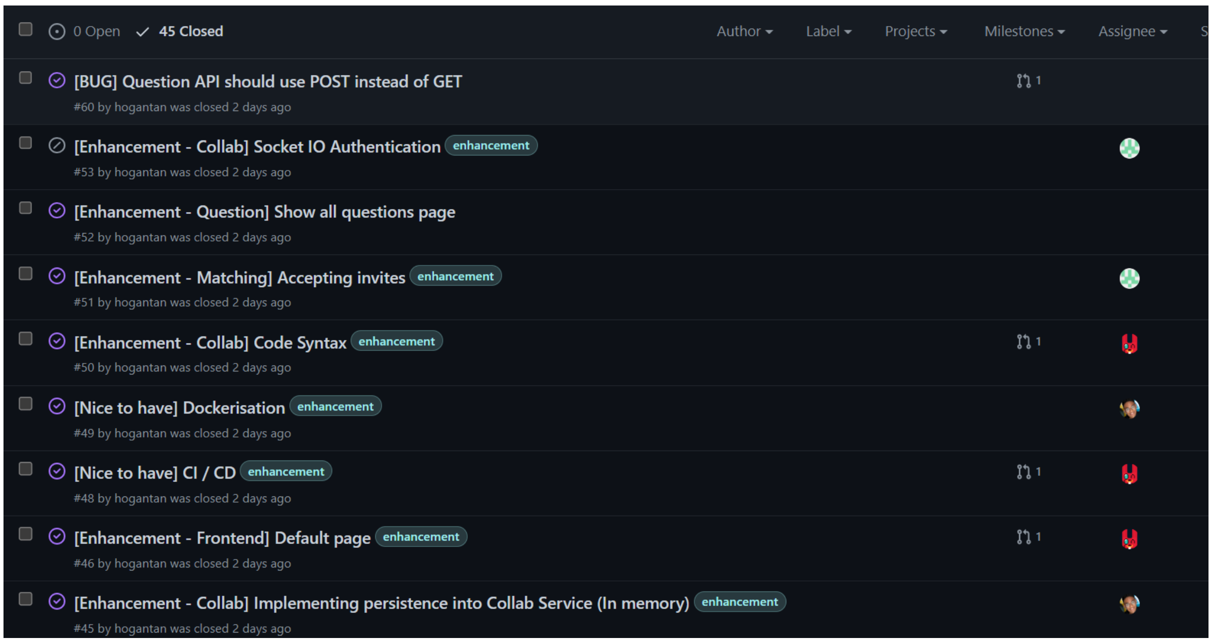
1. Continue working on project report
2. Integrate CI/CD to the project
3. Deploying application on GCP endpoint
4. Work on slides/presentation
5. System testing the application

Period: Week 13 to End ***(Milestone 3)*** *(7 Nov 2022 to 11 Nov 2022)*

1. Continue working on project report
2. Work on slides/presentation

4.3. Project Management

Weekly and milestone goals are tracked using GitHub Issues and GitHub Projects as seen in the following diagrams below. Bugs are tracked using GitHub Issues as well. Different issues and feature implementations are marked with varying priorities to denote the severity and urgency of the issue.


0 Open ✓ 45 Closed — Author ▾ Label ▾ Projects ▾ Milestones ▾ Assignee ▾

[BUG] Question API should use POST instead of GET
#60 by hogantan was closed 2 days ago — 1

[Enhancement - Collab] Socket IO Authentication `enhancement`
#53 by hogantan was closed 2 days ago

[Enhancement - Question] Show all questions page
#52 by hogantan was closed 2 days ago

[Enhancement - Matching] Accepting invites `enhancement`
#51 by hogantan was closed 2 days ago

[Enhancement - Collab] Code Syntax `enhancement`
#50 by hogantan was closed 2 days ago — 1

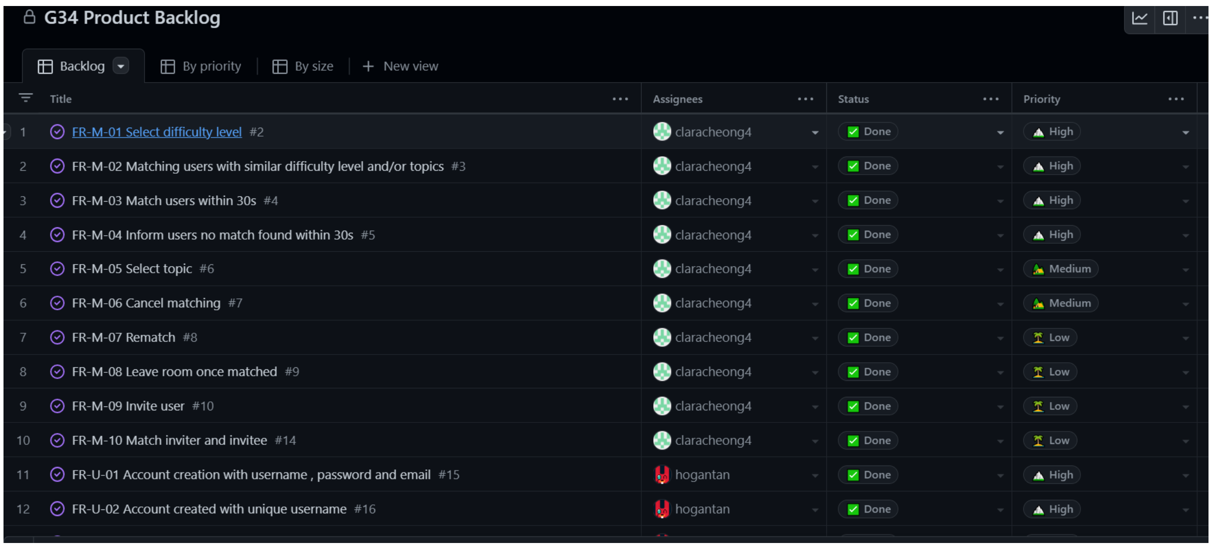
[Nice to have] Dockerisation `enhancement`
#49 by hogantan was closed 2 days ago

[Nice to have] CI / CD `enhancement`
#48 by hogantan was closed 2 days ago — 1

[Enhancement - Frontend] Default page `enhancement`
#46 by hogantan was closed 2 days ago — 1

[Enhancement - Collab] Implementing persistence into Collab Service (In memory) `enhancement`
#45 by hogantan was closed 2 days ago

GitHub Issue Tracker


G34 Product Backlog

Backlog | By priority | By size | + New view

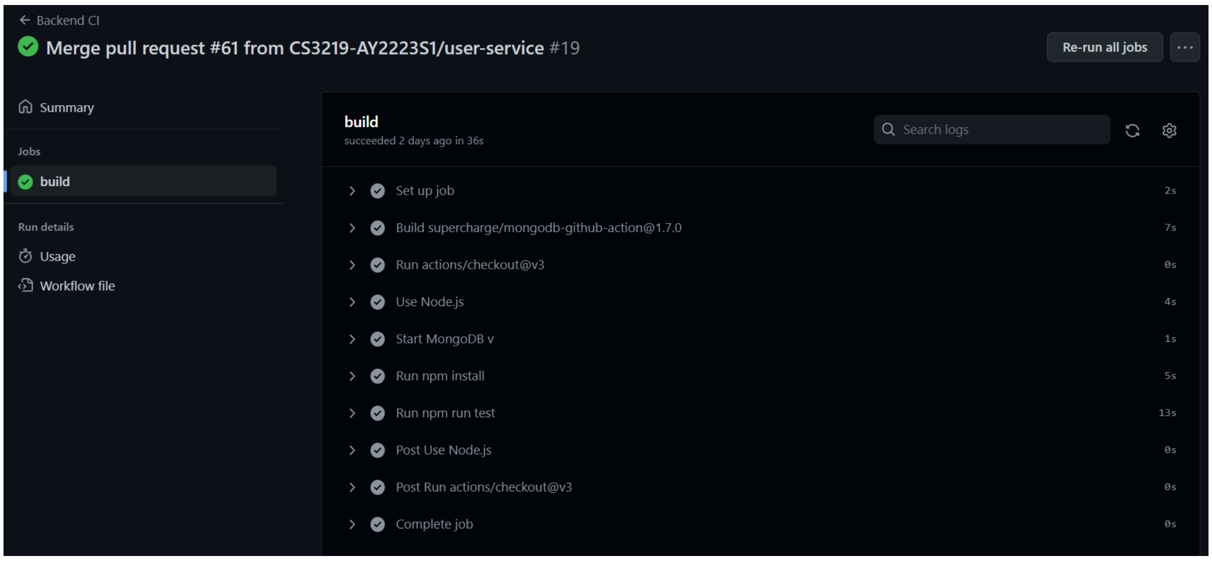
	Title	Assignees	Status	Priority
1	FR-M-01 Select difficulty level #2	claracheong4	Done	High
2	FR-M-02 Matching users with similar difficulty level and/or topics #3	claracheong4	Done	High
3	FR-M-03 Match users within 30s #4	claracheong4	Done	High
4	FR-M-04 Inform users no match found within 30s #5	claracheong4	Done	High
5	FR-M-05 Select topic #6	claracheong4	Done	Medium
6	FR-M-06 Cancel matching #7	claracheong4	Done	Medium
7	FR-M-07 Rematch #8	claracheong4	Done	Low
8	FR-M-08 Leave room once matched #9	claracheong4	Done	Low
9	FR-M-09 Invite user #10	claracheong4	Done	Low
10	FR-M-10 Match inviter and invitee #14	claracheong4	Done	Low
11	FR-U-01 Account creation with username , password and email #15	hogantan	Done	High
12	FR-U-02 Account created with unique username #16	hogantan	Done	High

GitHub Project Backlog

4.4. DevOps

(Nice to have) GitHub Actions is used as the CI tool. This is because it is relatively simple and easy to set up as we just needed to create a .github/workflows/backend.yml file and activate the CI tool in GitHub. Each push to the main branch triggers GitHub Actions to run the CI.

This way we will always be quickly notified about the status of the main branch and fix any issues if the CI fails. Any bugs discovered would be raised as an Issue in GitHub as seen above.



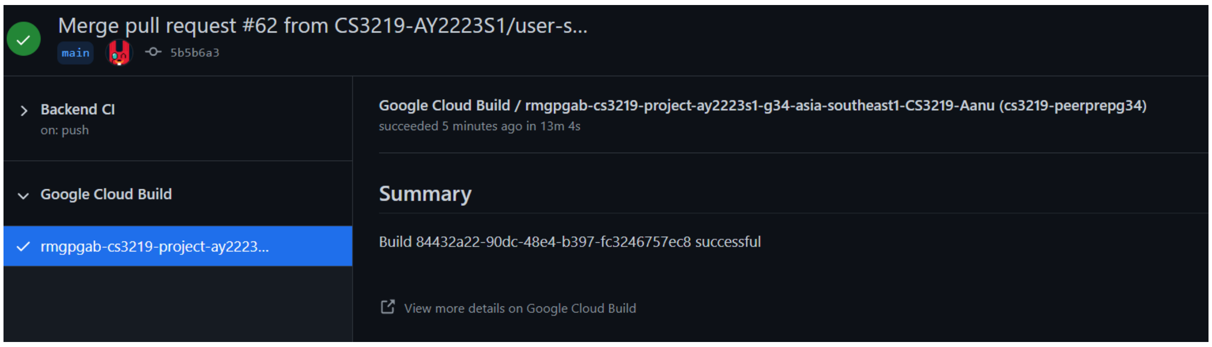
Merge pull request #61 from CS3219-AY2223S1/user-service #19

build
succeeded 2 days ago in 36s

```
Set up job                                        2s
Build supercharge/mongodb-github-action@1.7.0     7s
Run actions/checkout@v3                           0s
Use Node.js                                       4s
Start MongoDB v                                   1s
Run npm install                                   5s
Run npm run test                                  13s
Post Use Node.js                                  0s
Post Run actions/checkout@v3                      0s
Complete job                                      0s
```

Continuous Integration via GitHub

(Nice to have) Our team chose Cloud Build as our CD tool because it complements the usage of Google
Cloud Run when deploying our application. This is done by creating cloudbuild.yaml files in the frontend and various microservices to tell Cloud Build to build, test and deploy the various services. The Cloud Build is triggered every push to the main branch as well with the same reasoning as the CI. This will keep us informed about the status of the deployed application.



main
5b5b6a3
Backend CI
on: push
Google Cloud Build
rmgpgab-cs3219-project-ay2223...
View more details on Google Cloud Build

Merge pull request #62 from CS3219-AY2223S1/user-s...

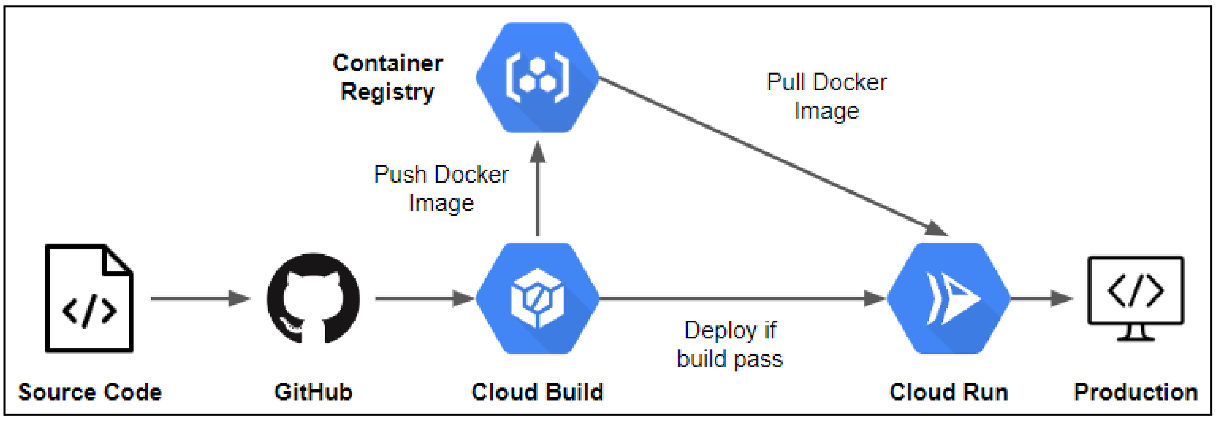
Google Cloud Build / rmgpgab-cs3219-project-ay2223s1-g34-asia-southeast1-CS3219-Aanu (cs3219-peerprepg34)
succeeded 5 minutes ago in 13m 4s

Summary

Build 84432a22-90dc-48e4-b397-fc3246757ec8 successful

Continuous Deployment via GitHub and Google Cloud

(Nice to have) Subsequently, the application is also deployed via Cloud Run as it pairs easily with Cloud Build. Once the Cloud Build successfully builds, the various images of the services will be pushed to the Container Registry and then pulled by Cloud Run to deploy to the production environment. Deployment was made easier since the application has been dockerised which will be explained more later. The application can be accessed on the deployed link: https://frontend-ragb43dwzq-de.a.run.app



Continuous Deployment and Deployment via Google Cloud

4.5. Testing

Unit testing was done for the backend APIs on the application namely in the user service and the question service. This testing is automated via the CI/CD. Integration and system testing was done manually at every pull request.

5. Application Design

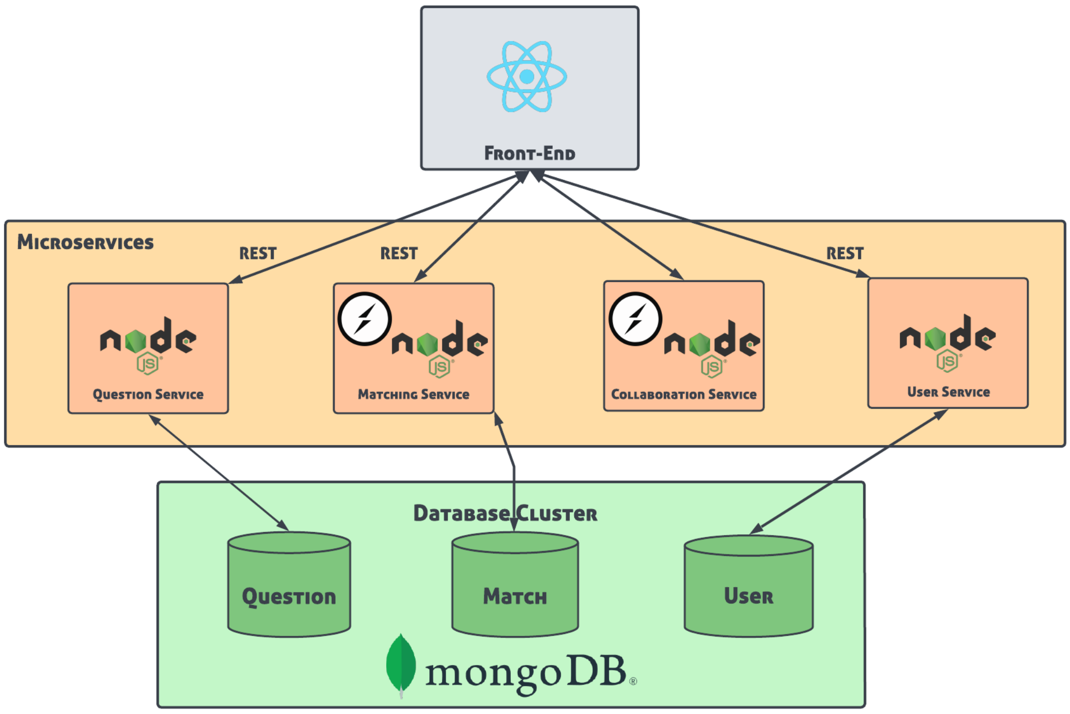
This section describes and justifies the various design decisions the team has come up with when designing the PeerPrep application.

5.1 Tech Stack

This section details and justifies the various tech stack choices for the PeerPrep application.

Choice	Rationale
Frontend: React	- Beginner friendly - Component-based abstraction allows for high reusability - Strong online community support - Virtual DOM allows for quick rendering
Backend: Express.js/Node.js	- Simple and flexible allows to build product of decent performance in a short span of a couple of weeks
Database: MongoDB	- Easy to setup and use - Schema patterns are more familiar than relational patterns - Data models parsed as JSON allows for easy manipulation of data
Pub-Sub Messaging: Socket.io	- Do not require a third party application to implement as socket.io library is available in Node.js, which is already a part of our Backend.
CI Tool: GitHub Actions	- Easy to setup since project is already setup in GitHub
CD Tool: GCP - Google Cloud Build	- Strong online community support - Ease of use
Deployment Tool: GCP - Google Cloud Run	- Strong online community support - Ease of use
Project Management Tools: GitHub Issues / GitHub Projects	- Familiarity - Ease of use - Link issues with pull requests - GitHub repository already set up by teaching team

5.2 Overall Design



Overall Architecture

We incorporated a central repository architecture design for PeerPrep primarily for one reason. There exist zero coupling between the microservices and thus they can be built independently, allowing project members to build it without waiting for progress on the other microservices. This has allowed us to quickly create a prototype and iteratively improve on it slowly. Moreover, due to the loose coupling, each microservice is easily extensible as well without having to worry about affecting the other services.

5.3 Docker

In order to facilitate an easier time on both locally building the application with the four services and frontend being up and also for deploying the application (mentioned earlier in section 4.4), each service and the frontend also has their individual Dockerfiles. A docker-compose.yml exists so that in the context of local development, team members will have an easier time to build the entire application via a single command, rather than individually starting up each service one by one.

5.4 Frontend Design

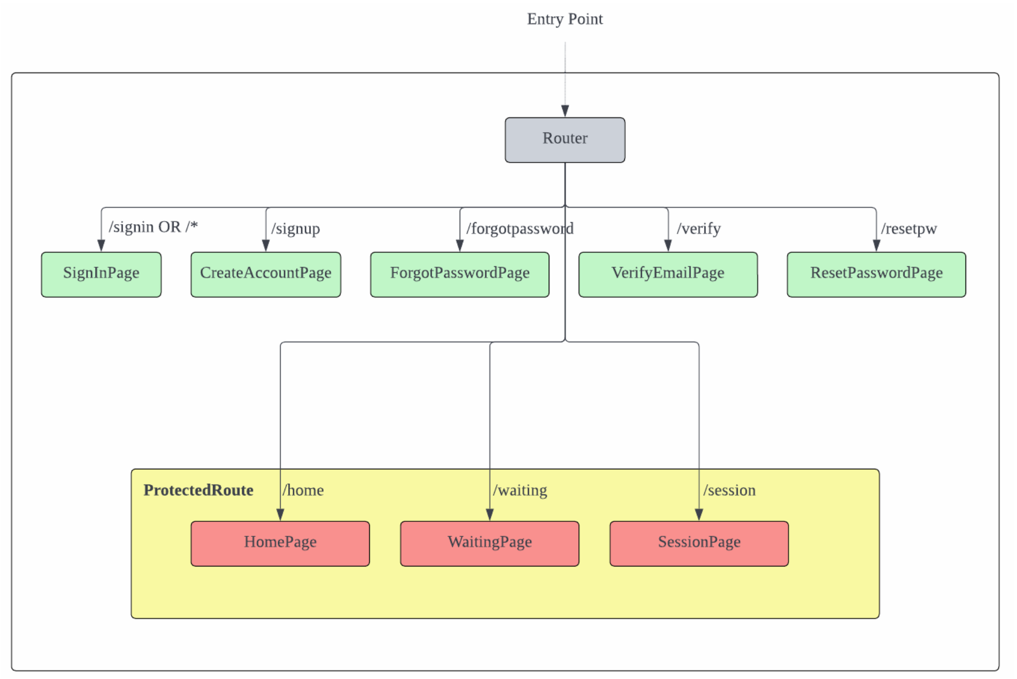
This section explains the design decisions of the frontend.

5.4.1 Overall Design

The team has used the React framework to build the frontend of PeerPrep. Since PeerPrep is a Single Page Application (SPA), there will only be a single time load of the webpage at the start from the server and then subsequently the frontend routing will decide the pages to be rendered. This ensures that the speed of transitioning between pages is fast by letting the frontend do most of the work. Moreover, React also offers hooks and state management which the team has utilised to easily pass data between pages. Therefore, React provides what the team is looking for, which is a framework that is able to build fast and scale user interfaces well with ease.

The frontend library that the team has utilised is Material UI which provides various customizable components which are easy to use. More importantly, it saves the team time from having to implement components via Cascading Style Sheets directly(CSS). The components provided by Material UI also provide a good level of responsiveness as well.

React also allows for easy routing and between pages. The routing design of Peerprep is given by the diagram below.



Frontend Routing Design

Note that if users decide to access random routes which are not defined as above, they will be routed to the SignInPage instead. This rerouting ensures that users are never looking at a blank screen (404 error) when accessing any random routes. This is done to improve the user experience of the application as it redirects the user to the main entry point of the application if any random routes are detected.

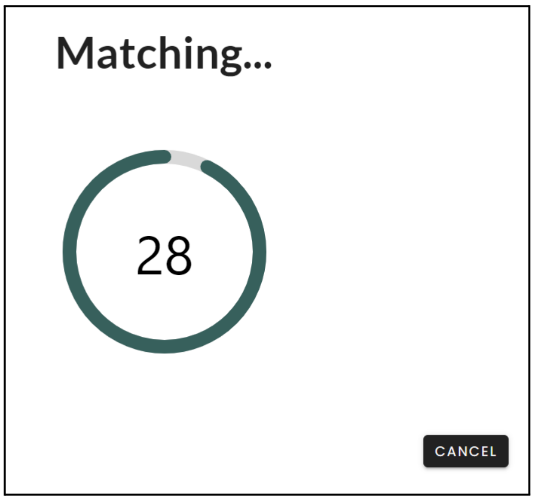
In addition, the HomePage, WaitingPage and SessionPage pages are protected which means for a user to access these pages, the user has to be authenticated (i.e. the user has to be signed in to a valid account). This is done because these pages require some form of user information to function. Therefore, the use of ProtectedRoute is to ensure the security of the functionality of these pages.

5.4.2 User Interface / User Experience

This section details the various user interface / experience considerations when designing the frontend.

5.4.2.1 Loading Animations

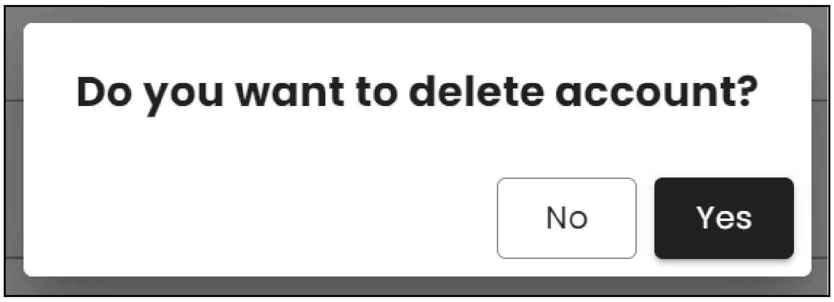
Loading animations are placed in requests that take longer than a second such as the creation of an account, resetting password and waiting for a match. This is to let the user know that the system is loading rather than just freezing the screen which will result in a bad user experience as the user might think that the system has crashed. The loading animations serve as an indicator that the application is running but requires some extra time. Therefore, this lets users know what to expect from the system when it is loading.



Matching Loading Animation

5.4.2.2 Confirmation Dialogs

For key irreversible actions such as logging out and deleting an account, the application prompts an alert dialog for the user to confirm their decision. This is to ensure that users are certain of their actions and willing to go ahead with it. This will prevent users from accidentally deleting their accounts or logging out which would improve the user experience as a whole.



Confirmation Dialog

5.4.2.3 Minimalistic UI

The design for the user interface is designed in a minimalistic manner as we would like the application to be friendly to new users. With a minimalistic UI, it will be clear to the user what the various functionalities of a page provides. In other words, such a UI will be able to guide users along the journey of the application well and the only time where users will have to scratch their heads when using our application is when they are solving the various coding questions. This is achieved by not cluttering the application with unnecessary elements/animations/components but to only provide what is necessary to the specific page the user is on. Subsequently, this allows for unambiguous UI components. For example, buttons are clearly buttons in the application and users do not have to think twice about whether something is clickable. Overall, this will improve the user experience on the application as the flow of the application is intuitive.

5.5 Backend Design

This section explains the design decisions of the backend.

5.5.1 Domain Driven Design

Since the application adopts the Microservices Architecture, the backend architecture follows closely to a Domain-driven-design (DDD) via the use of ORMs/ODMs. The backend is separated into 4 domains namely: user, matching, question and collaboration each with their own independent business logics. This design ensures that the backend design has low coupling among domains.

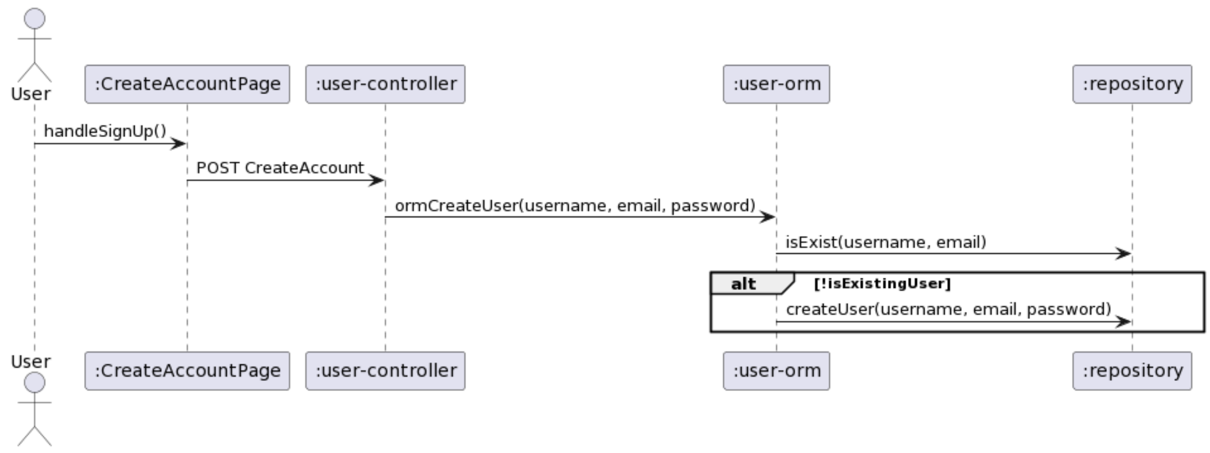
Moreover, each domain exposes their own API which is then called by the frontend. This decouples the frontend from the backend as now the frontend is only interacting with the API of the backend services and has no idea of how each services function internally. As a result, the application is decoupled both horizontally and vertically.

In addition, such a design also ensures that the application is easily extensible. That is, if a new business domain is to be added, one just has to add a new domain without having to affect any of the other domains. Therefore, this design ensures that the application is highly maintainable due to the separation of concerns and single responsibility principle of domains.

5.5.2 User Service Design

This section explains the design decisions of the various features in the user service domain.

5.5.2.1 Create Account



Create Account Sequence Diagram

When creating a new account, a user needs to provide a unique username, unique email address and password. That is the username and email address cannot already exist in the database. Usernames and passwords also do have length requirements which are stated in the functional requirements. Usernames have to have a minimum and maximum length requirement because usernames are displayed in the HomePage as well as during the SessionPage when users are chatting with each other. In order for usernames to be displayed nicely visually, there needs to be some length requirement to prevent any word wrapping or overflowing. The minimum length requirement passwords is for security concerns as in order to ensure that users passwords are of reasonably strong, the team decided that a minimum length is needed. The frontend does the validation of these requirements.

(Nice to have) Upon successfully creating an account, the system will send a verification email to the registered email and users are expected to access the verification link in order to verify their accounts. Only verified accounts are allowed to login. This is done so that users are not able to just create accounts with dummy emails. This is done via the nodemailer library as a middleware. The verification link uses a JWT authentication token to authenticate the verification process as well. Therefore, only owners of the registered email can verify their accounts.

Salted hashing is done to user passwords before storing in the database. This is done using the bcrypt library. This is in line with the security aspect of the application. If there was to be a leak in data, user passwords would still be protected.

5.5.2.2 Sign In / Log out

After successfully signing in, a user is assigned a HTTP-ONLY JWT authentication token which is then used to access authenticated services such as changing of passwords, finding a match and starting a session, etc. This is done via the jsonwebtoken as a middleware. The choice of specifying HTTP-ONLY token is to prevent any client side script from accessing and tampering with it which is a security concern.

After logging out, the system blacklists the earlier generated jwt token so as to prevent the user from using the token to access protected services without signing in.

5.5.2.3 Reset Password

(Nice to have) The reset password feature also utilises the nodemailer library similar to the verification of an account where a reset email will be sent to the user. A reset token is generated in order for the user to access the reset link to provide a form of authentication. This token is also a jwt token. This token has an expiry time of 10 minutes which is the window in which the reset link is valid. This is to ensure that the reset link does not last indefinitely.

5.5.2.4 Change Password

Changing of password requires users to provide their old password and new password. This feature is protected by authentication. That is, a user has to be authenticated in order to access this feature. The need to provide the old password verifies that the actual user and serves as a double confirmation.

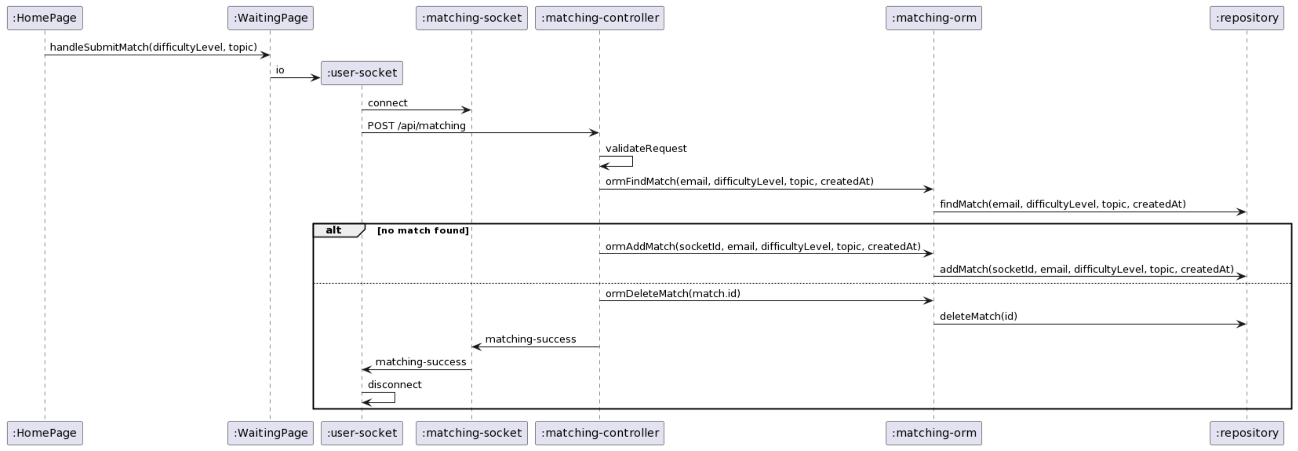
5.5.2.5 Authenticated Services

The following services are protected by authentication. That is, a user has to be authenticated in order to access these services: Delete Account, Reset Password and Change Password. The authentication of Reset Password differs from the other two in that the authentication is done implicitly via sending a reset email to the user’s email address.

5.5.3 Matching Service Design

This section explains the design decisions of the various features in the matching service domain.

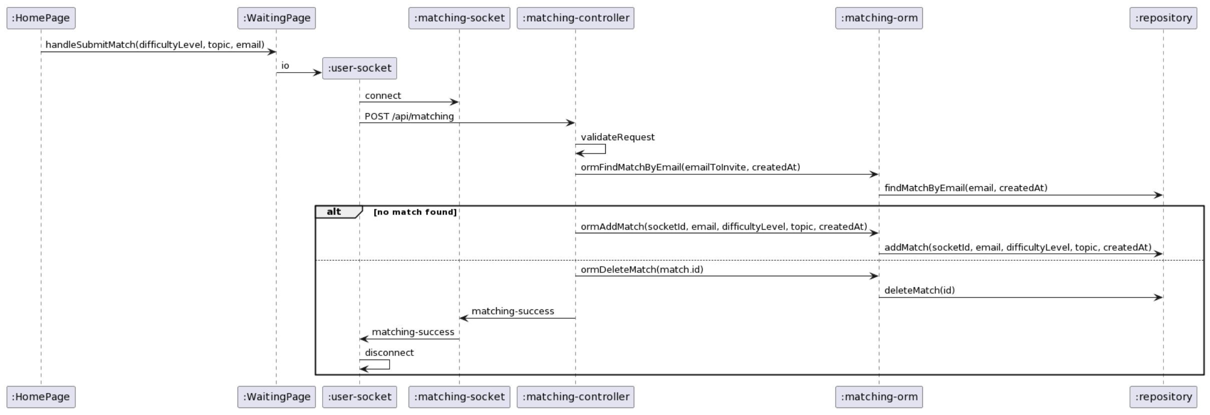
5.5.3.1 Match by Difficulty Level and Topic


:HomePage
:WaitingPage
:matching-socket
:matching-controller
:matching-orm
:repository
handleSubmitMatch(difficultyLevel, topic)
io
:user-socket
connect
POST /api/matching
validateRequest
ormFindMatch(email, difficultyLevel, topic, createdAt)
findMatch(email, difficultyLevel, topic, createdAt)
alt
[no match found]
ormAddMatch(socketId, email, difficultyLevel, topic, createdAt)
addMatch(socketId, email, difficultyLevel, topic, createdAt)
ormDeleteMatch(match.id)
deleteMatch(id)
matching-success
matching-success
disconnect

Match by Difficulty and Topic Sequence Diagram

After a user selects a difficulty level and topic, the system will query the database to find a matching user. If a match is successfully found, it will attempt to connect to sockets of both users and generate a unique room id and send it to both users through their respective sockets using a matching-success event. Else, it will add a matching request to the database with a time to live of 30 seconds. If the user is not successfully matched within 30 seconds, the matching is considered to have failed.

5.5.3.2 Match by Invitation


:HomePage
:WaitingPage
:matching-socket
:matching-controller
:matching-orm
:repository
handleSubmitMatch(difficultyLevel, topic, email)
io
:user-socket
connect
POST /api/matching
validateRequest
ormFindMatchByEmail(emailToInvite, createdAt)
findMatchByEmail(email, createdAt)
alt
[no match found]
ormAddMatch(socketId, email, difficultyLevel, topic, createdAt)
addMatch(socketId, email, difficultyLevel, topic, createdAt)
ormDeleteMatch(match.id)
deleteMatch(id)
matching-success
matching-success
disconnect

Match by Invitation Sequence Diagram

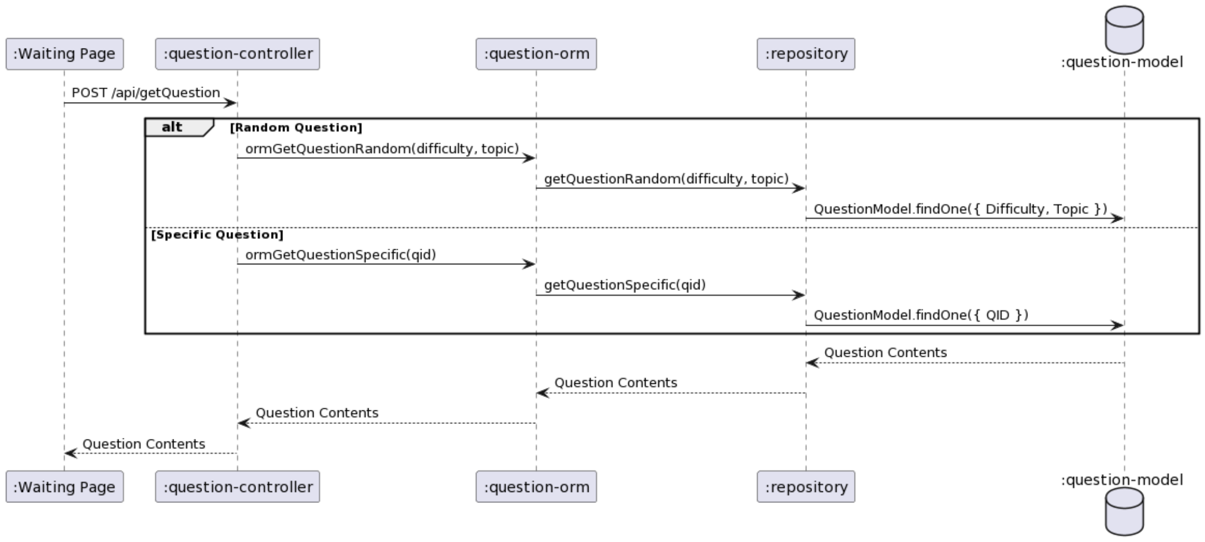
After a user selects a difficulty level and topic and enters an email of a user to invite, the system will query the database to find a matching user. Similarly to 5.4.3.1, if a match is successfully found, it will send an unique room id and a confirmation that the matching was successful by invitation to both users through their respective sockets. Else, it will add a matching request to the database with a time to live of 30 seconds so that the user can get matched to another user. If the user is not successfully matched within 30 seconds, the matching is considered to have failed. If the matching via invitation was successful, the collaboration service will notify the users that they have been matched via an invitation.

5.5.4 Question Service Design

This section explains the design decisions of the various features in the question service domain.

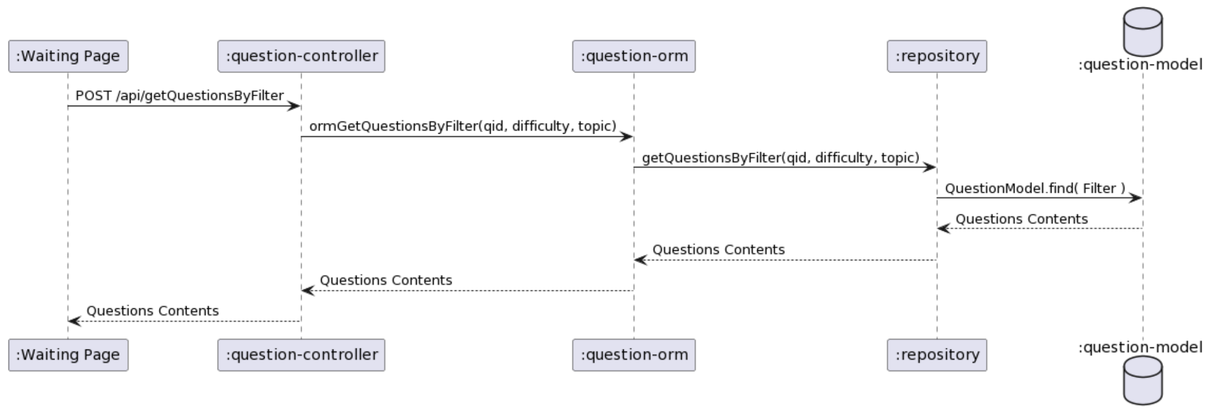
The question service is designed to serve as many use cases as possible without unnecessarily creating many API endpoints that serve niche purposes. As such, there are currently two multi-use API endpoints, namely “getQuestion” and “getQuestionsByFilter”.

The endpoint “getQuestion” works by accepting three parameters. These are “difficulty”, “topic”, and “qid”. However, only “difficulty” and “topic” OR “qid” are needed at any one point. The endpoint then returns to the client, in JSON format, the contents of one question. In the case that “difficulty” and “topic” are provided, a random question is chosen among the remaining cohort. In the case that “qid” is provided, the specific question with said ID is returned.



Get Question Sequence Diagram

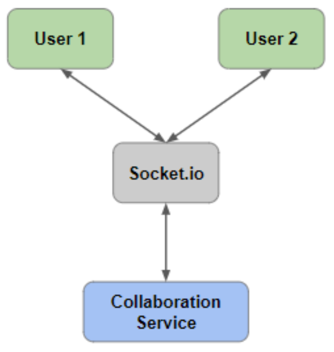
The endpoint “getQuestionsByFilter” works by accepting the same three parameters as the previous endpoint. However, this endpoint is designed to return the contents of multiple questions at once – in a list (all in JSON format as before). This endpoint filters all questions in the database for matching questions that have a specific “difficulty”, “topic”, and/or “qid”. Note that none of these parameters must be provided in the API query. As such, for example, if a client does not provide any filter parameters then a list containing the contents of all questions in the database will be returned.



Get Questions By Filter Sequence Diagram

5.5.5 Collaboration Service Design

This section explains the design decisions of the various features in the collaboration service domain.



Collaboration Service Pub-Sub Pattern

The Collaboration service uses Socket.io to enable real time communication between the users in a session.

The collaboration service is the endpoint used to establish the socket.io connections necessary to facilitate two key functionalities,

1. Real-time code editing between the two users
2. Real-time chat communication between the two users.

The two functionalities above can be separated similarly into two different topics, 'editor' and 'chat' (Nice to have).

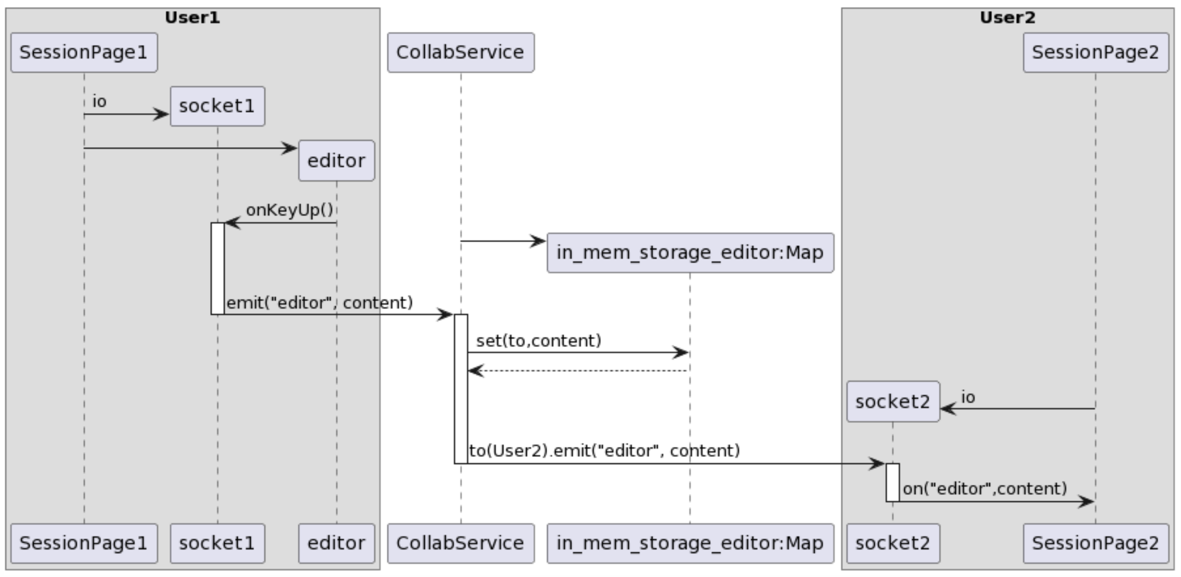
When the socket initially gets established, it first checks if the session ID still exists, and if so, fetches the content from the collab-service to the frontend. Any changes in the frontend code editor will then emit an event, indicating that a change has occurred, and notifies the other user. The flow of events can similarly be applied to the chat communication, but an event only gets emitted if the user sends the message via the Enter key.

Both the ‘editor’ and ‘chat’ functionalities subscribe to the Pub-Sub message design pattern to send messages to parties involved. The client sockets will emit a new message in a way such that the Server socket will receive this signal and redirect it to the appropriate socket channel. The client socket will then listen to this channel and update accordingly.

The ‘editor’ functionality allows users to type text in code syntax. That is, the ‘editor’ will function like a basic code text editor. The ‘editor’ is built using Microsoft Monoco Editor library which powers Visual Studio Code. This allows users to type in code syntax rather than just plaintexts which will boost user experience as the editor will have features such as indentation, line numbering, etc.

The contents of the ‘editor’ and the ‘chat’ are ‘persistent’ as well. Persistence in this case where if a user refreshes the SessionPage, the content will not be lost.

5.5.5.1 On Editor/Chat Changes



Editor/Chat Sequence Diagram

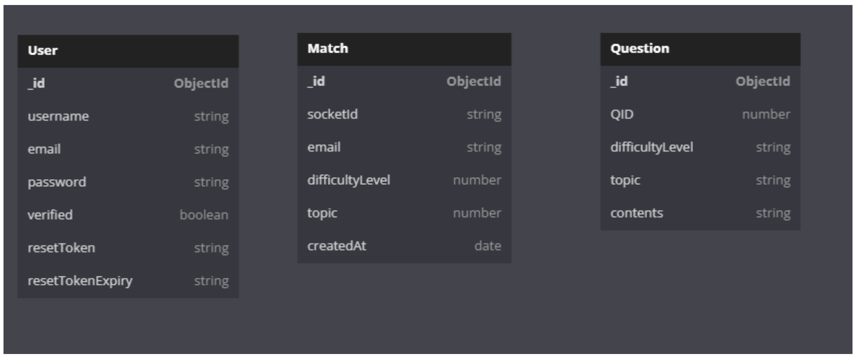
For further explanation on how the contents of the editor are stored and changed in real time, the flow of the sequence diagram explains this succinctly. The trigger event, i.e `onKeyUp()` will emit an event to the collaboration service which in turn will pass the event to the other user’s Session Page. It also stores the content in an in-memory storage in Collaboration Service. For now it exists as a Map object, but this concept can similarly be extended to other possible types of data storage as well.

The above flow can also be applied to the Chat functionality as well, with the trigger event being different i.e pressing on enter key while having the chat box active. A different Map object also houses the content of the chat for a given session.

5.6 Database Design

This section explains the design decisions of the database.

The team has adopted the database per service pattern to fit the microservice architecture of the varying services. Therefore, each service is assigned to a specific database as each database is relatively different from one another in terms of data stored. In this case, the user service is assigned to the user database, the matching service is assigned to the match database and the question service is assigned to the question database. The design of the database is given below.



User	
_id	ObjectId
username	string
email	string
password	string
verified	boolean
resetToken	string
resetTokenExpiry	string

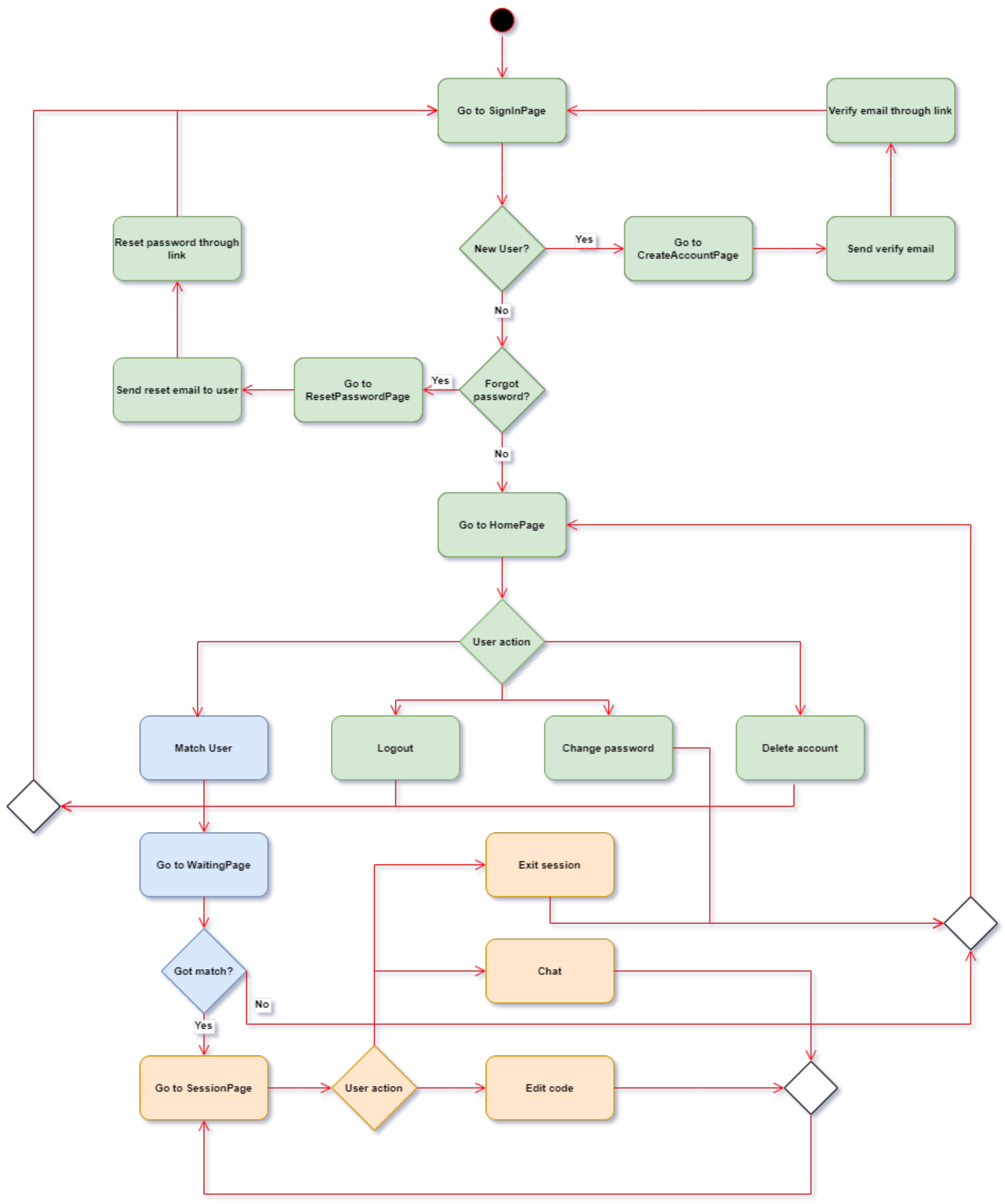
Match	
_id	ObjectId
socketId	string
email	string
difficultyLevel	number
topic	number
createdAt	date

Question	
_id	ObjectId
QID	number
difficultyLevel	string
topic	string
contents	string

Database Design

6. Application Flow

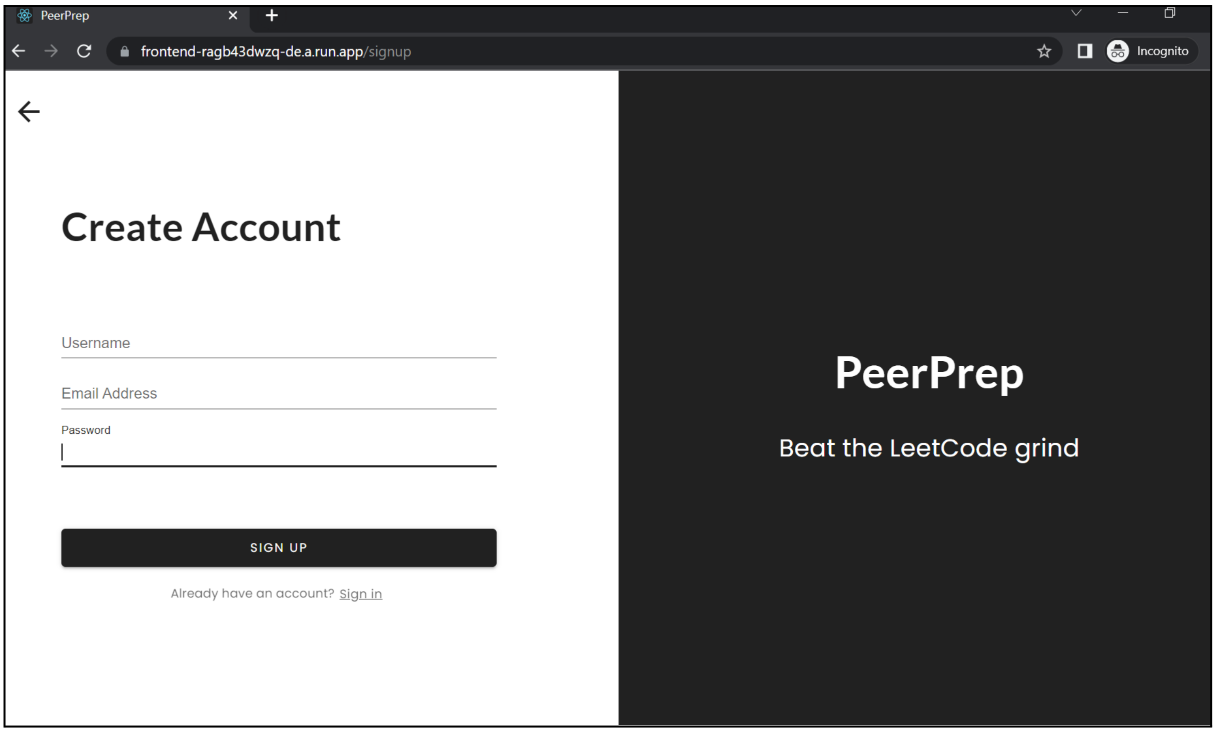
This section will detail the flow of how the user uses the application. The following diagram details the general flow of the application as a whole. The API for the respective domains can be found in the Appendix.



Application Flow Activity Diagram

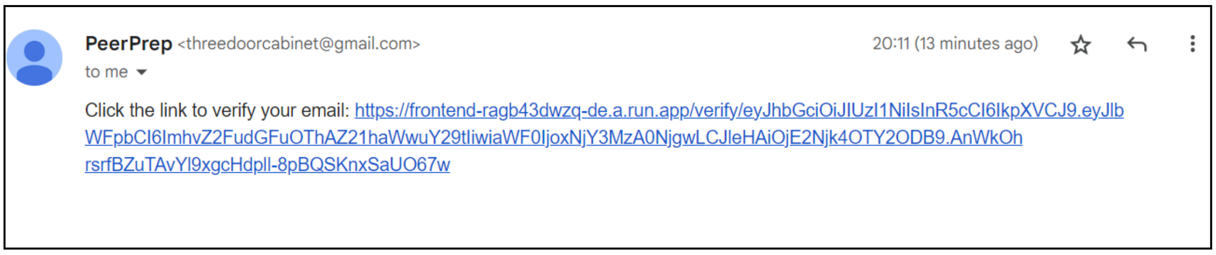
6.1 User Service UI

Firstly, the user can create a new account by hitting the Create Account button in the SignIn page. The user will then have to input a username, email address and password.



Create Account Page

Upon successfully creating an account, a verification email will be sent to the user.



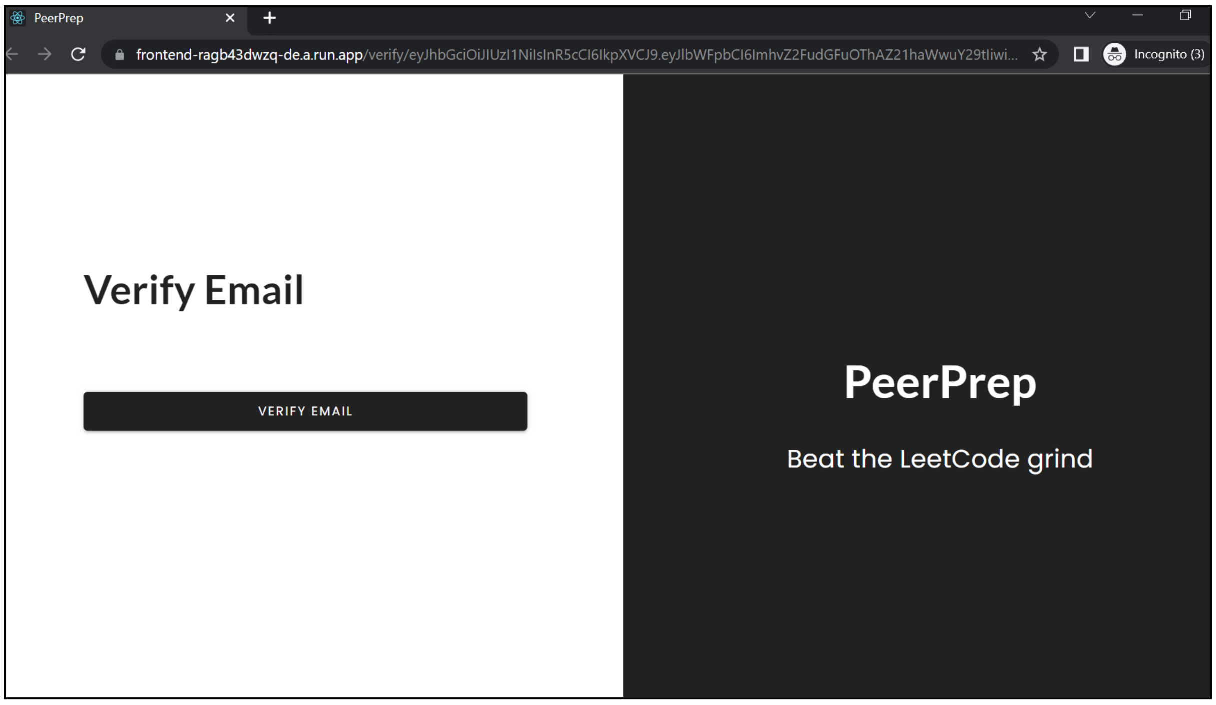
PeerPrep <threedoorcabinet@gmail.com>
to me

20:11 (13 minutes ago)

Click the link to verify your email: https://frontend-ragb43dwzq-de.a.run.app/verify/eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlb
WFpbCI6ImhvZ2FudGFuOThAZ21haWwuY29tIiwiaWF0IjoxNjY3MzA0NjgwLCJleHAiOjE2Njk4OTY2ODB9.AnWkOh
rsrfBZuTAvYl9xgcHdpll-8pBQSKnxSaUO67w

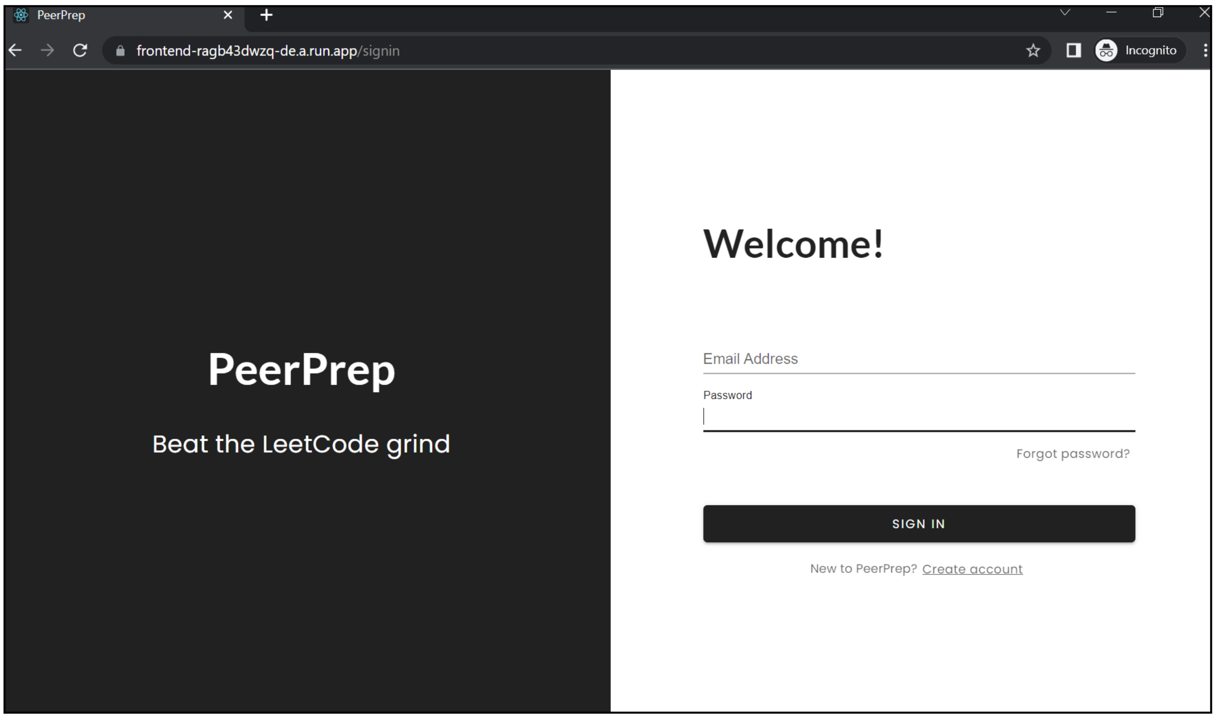
Create Account Verification Email

The user will then have to access the verification link in the email to verify the account.


PeerPrep
frontend-ragb43dwzq-de.a.run.app/verify/eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImhvZ2FudGFuOThAZ21haWwuY29tIiwi...
Incognito (3)
Verify Email
VERIFY EMAIL
PeerPrep
Beat the LeetCode grind

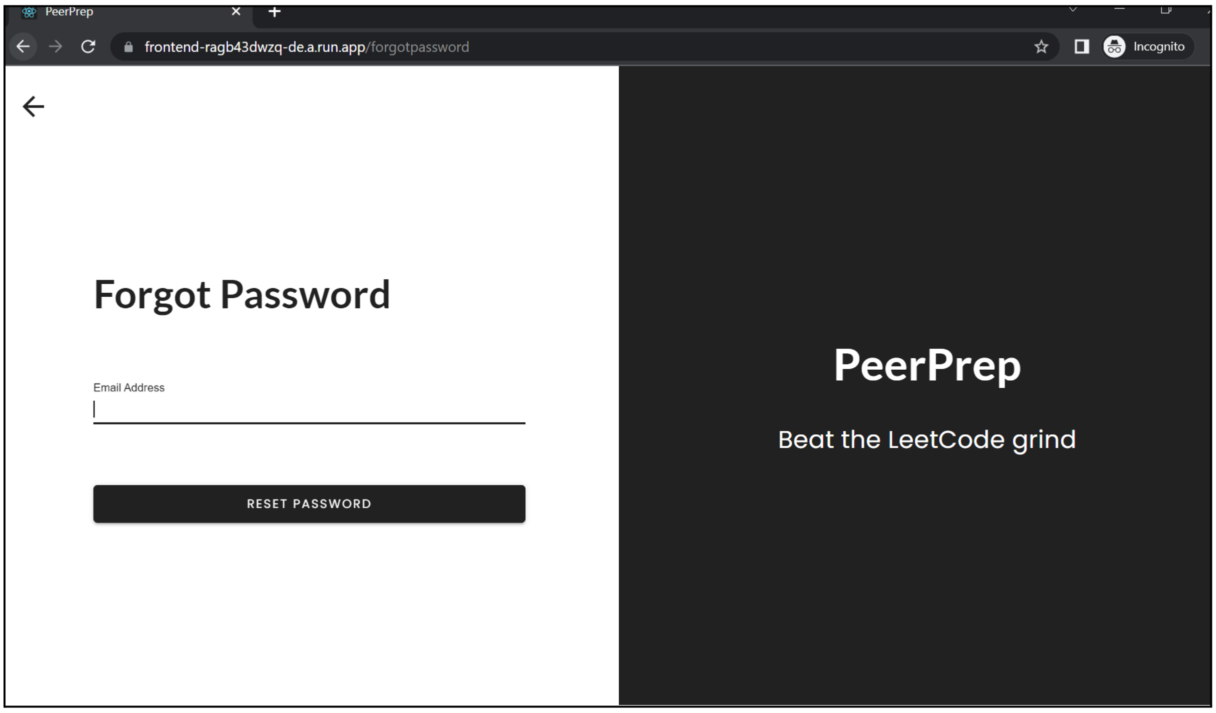
Verify Account Page

Once the account is verified, then the user will be able to login and enter the Home Page.



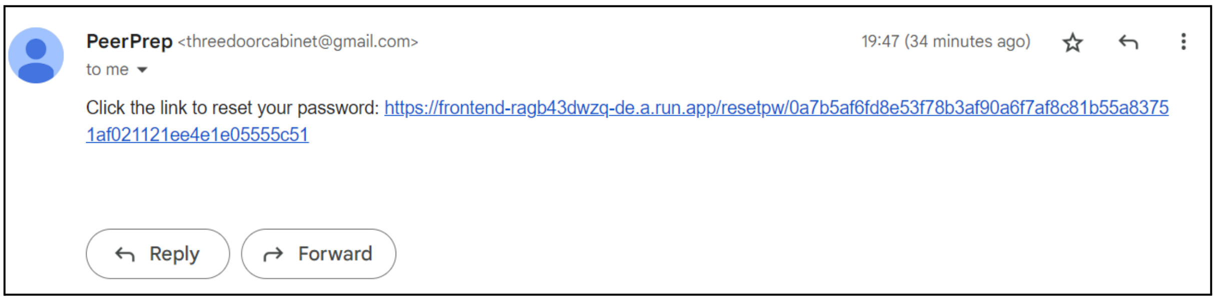
Sign In Page

Users can reset their password if they have forgotten their password by hitting the Forgot Password button in the SignIn page.



Forgot Password Page

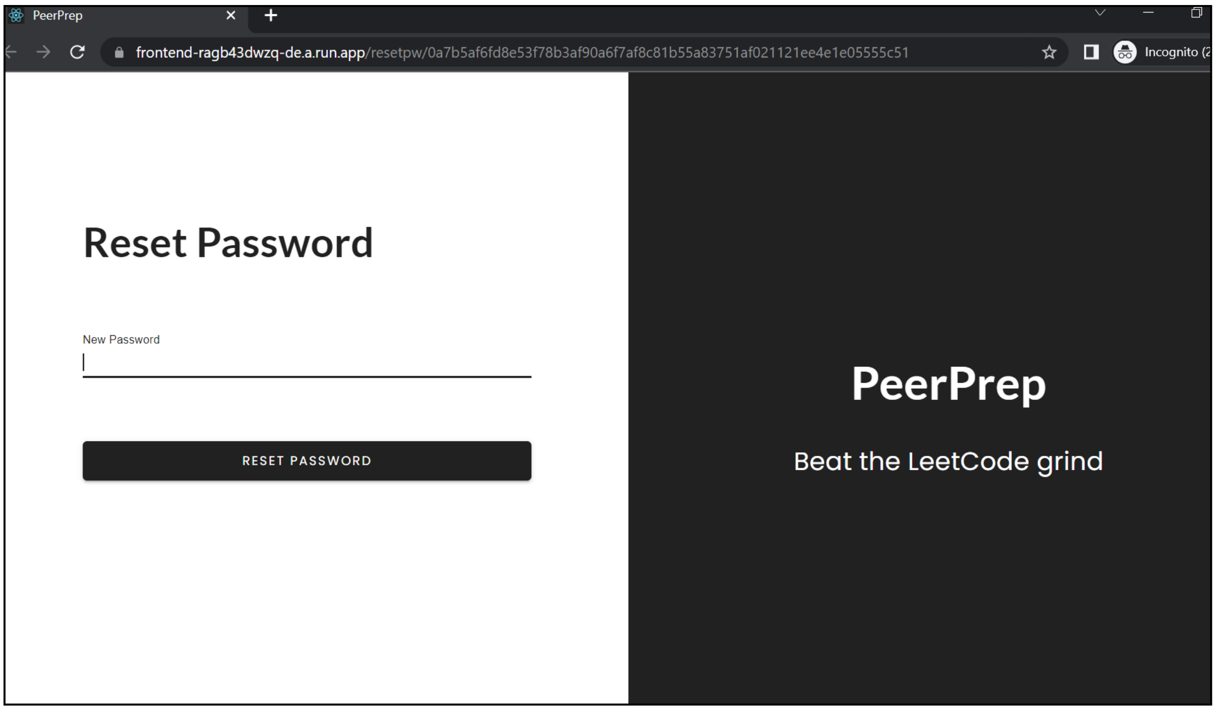
The user will then enter their registered email address and upon success, a password reset email link will be sent to the registered email.


PeerPrep <threedoorcabinet@gmail.com>
to me
19:47 (34 minutes ago)

Click the link to reset your password: https://frontend-ragb43dwzq-de.a.run.app/resetpw/0a7b5af6fd8e53f78b3af90a6f7af8c81b55a83751af021121ee4e1e05555c51

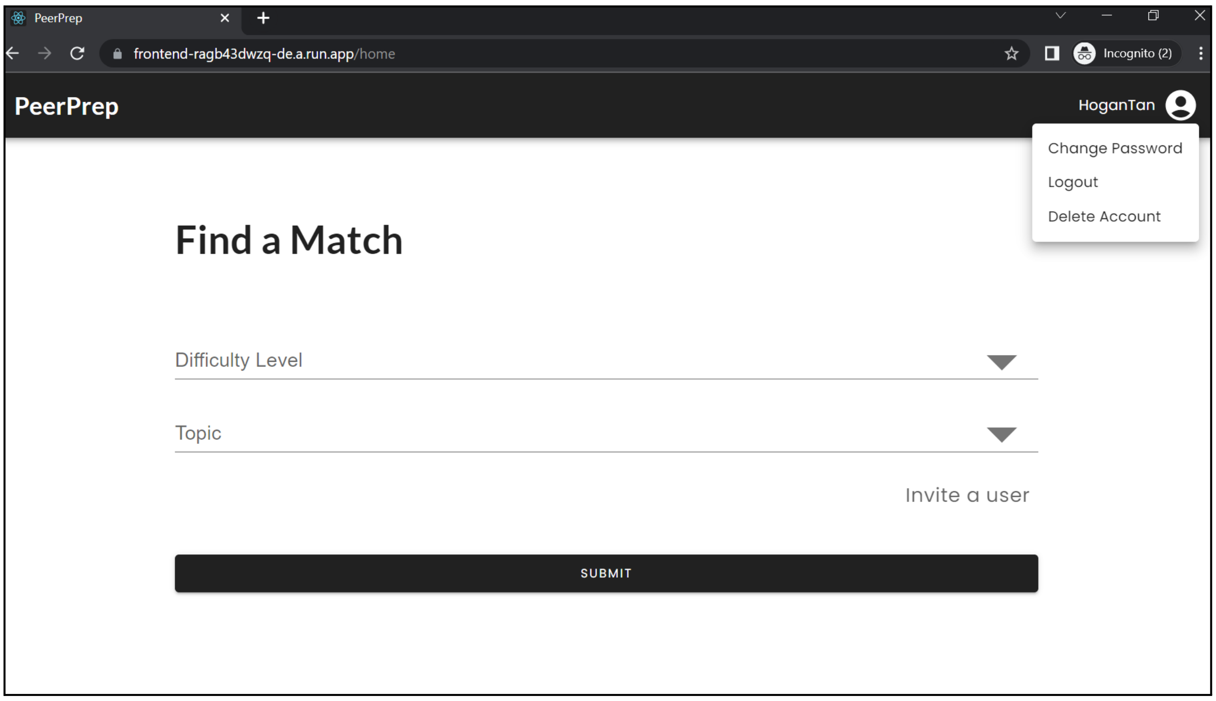
Reset Password Email

Users will then have to access the reset password link in the email to proceed to resetting their password. Users will then have to enter their new password and login if the reset is successful. A success pop up will result if the reset is successful.

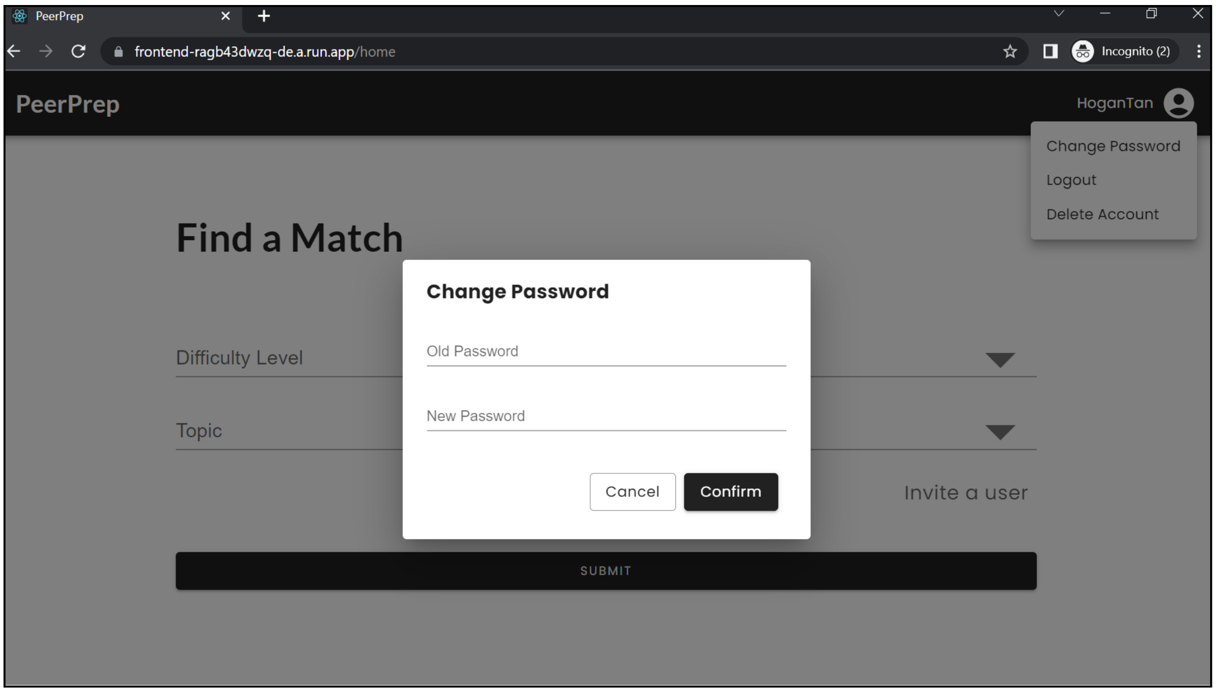

PeerPrep
frontend-ragb43dwzq-de.a.run.app/resetpw/0a7b5af6fd8e53f78b3af90a6f7af8c81b55a83751af021121ee4e1e05555c51
Incognito
Reset Password
New Password
RESET PASSWORD
PeerPrep
Beat the LeetCode grind

Reset Password Page

Users can change their password in the Home Page as well by hitting the Change Password button located in the menu at the top right of the Home Page. Users will then have to provide both valid old and new passwords to change their password. A success pop up will result if the change is successful.



Home Page

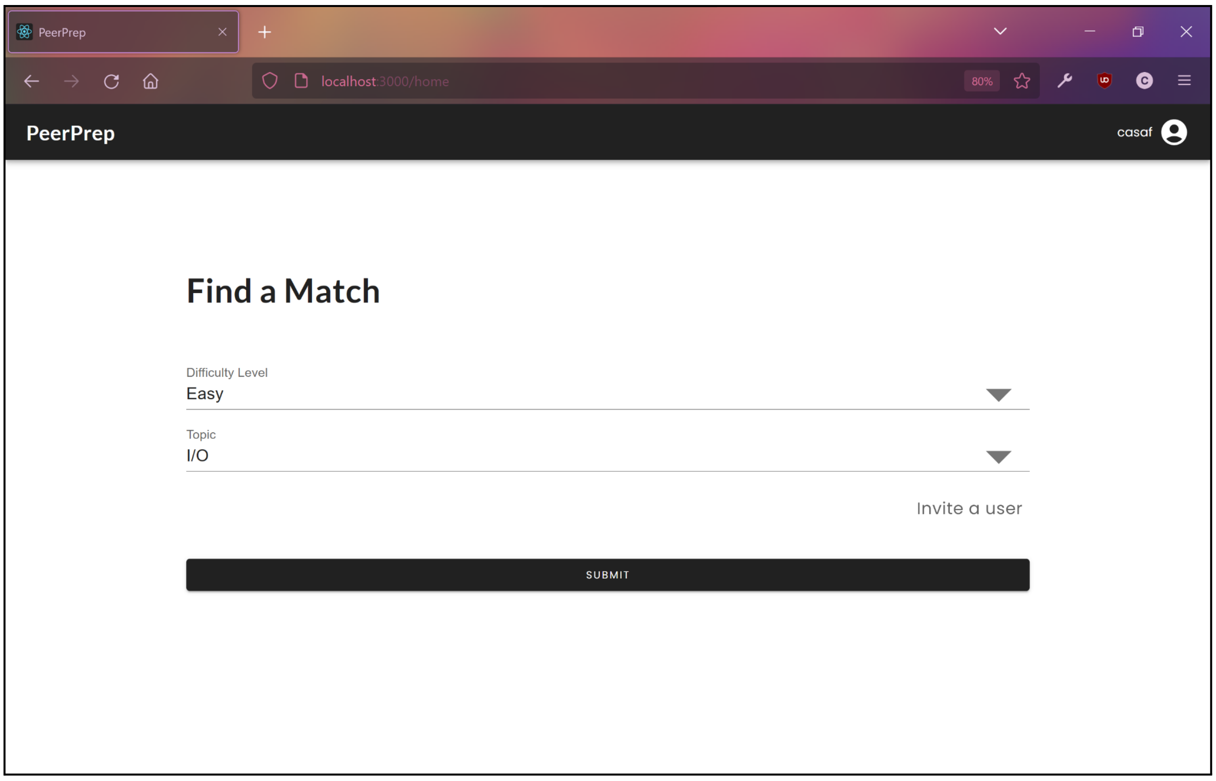


Change Password Dialog

Users can log out or delete their account in the Home Page as well by hitting the Logout and Delete Account buttons respectively located in the menu at the top right of the Home Page. Users will be prompted with a confirmation dialog when interacting with these buttons to confirm their decision.

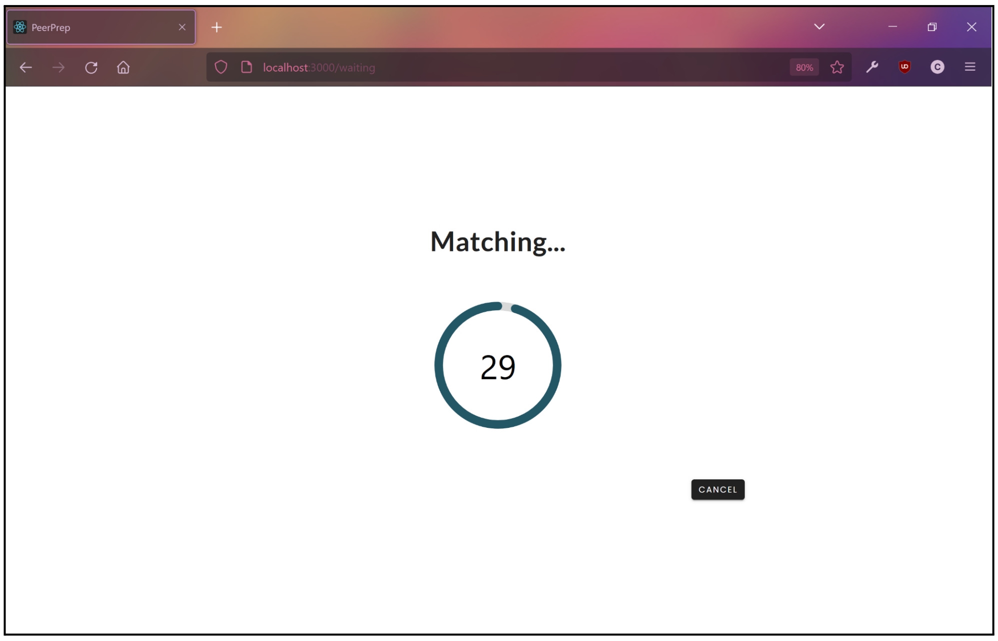
6.2 Matching Service

Users can find a matching by selecting desired difficulty level and topic.



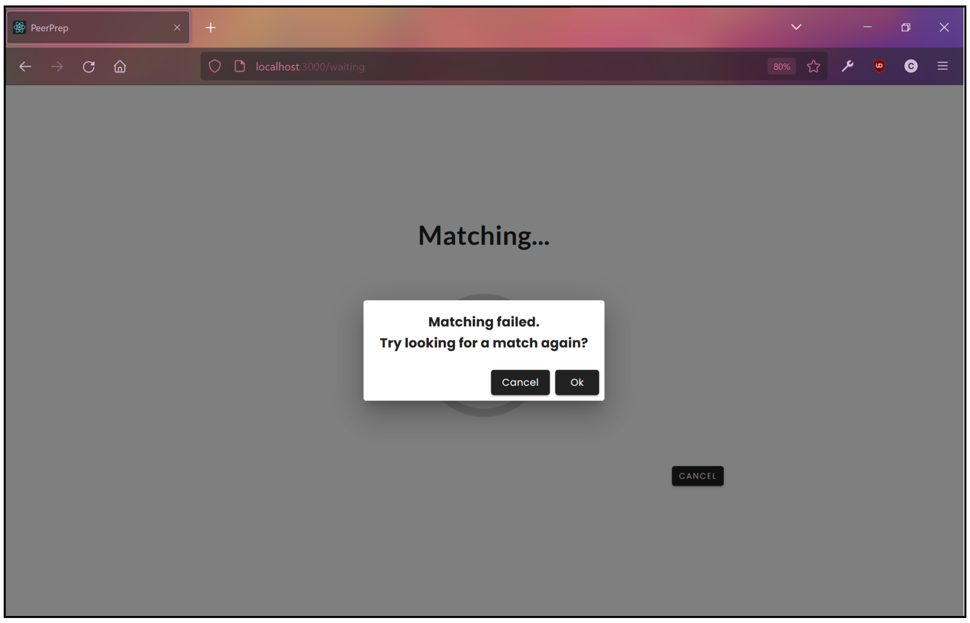
Home Page select Difficulty and Topic

Users will wait for up to 30 seconds to find a match. Users can choose to cancel the match and return to home.



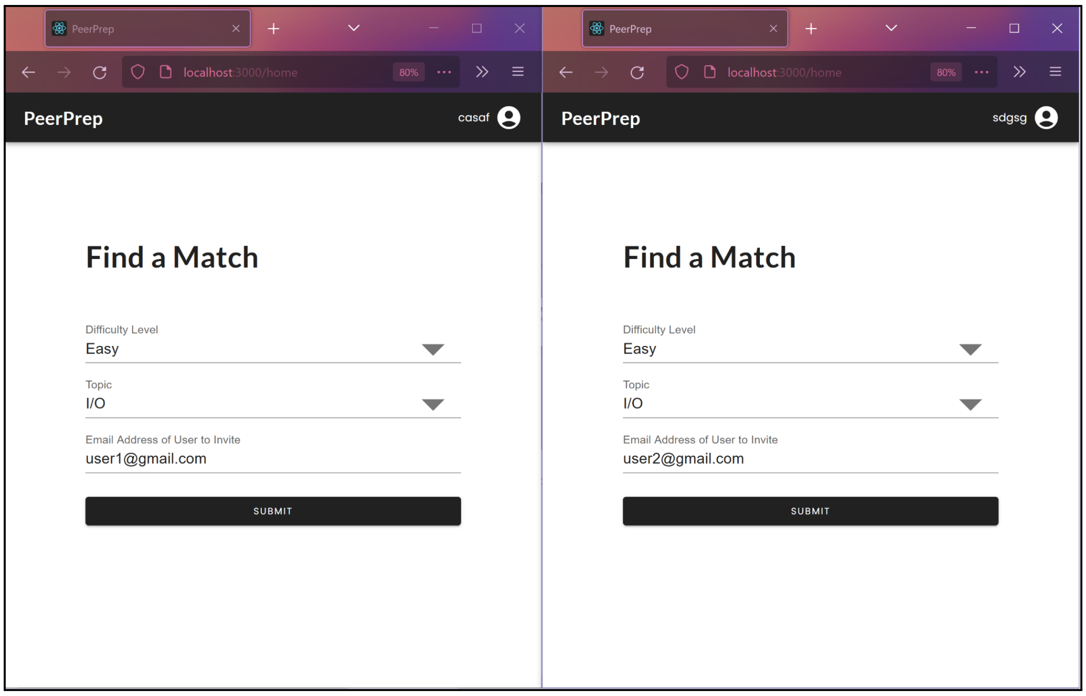
Waiting Page

If the matching had failed, users can go back to home or find a match again.



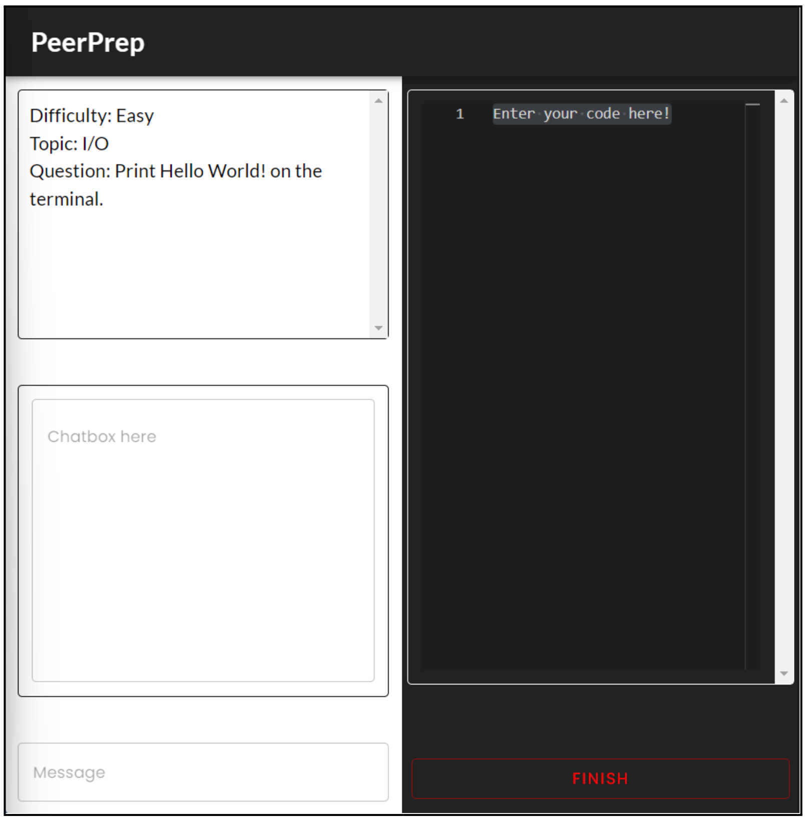
Waiting Page Fail Dialog

If the matching was successful, users will be redirected to a session with the matched user. Alternatively, users can invite each other by specifying the email of the user to invite.



Home Page select Invite User

6.3 Collaboration Service

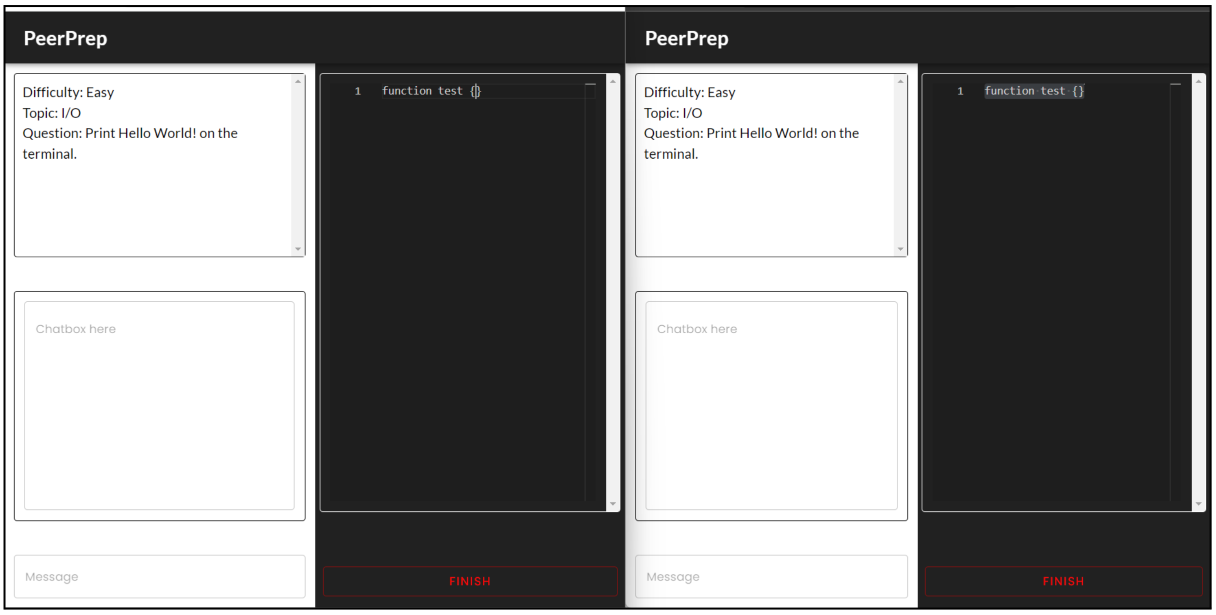


Difficulty: Easy
Topic: I/O
Question: Print Hello World! on the terminal.

```
Enter your code here!
```

Session Page For User

Once matched the two users will get into the session page, where they mainly interact with the chatbox and the code editor.



Difficulty: Easy
Topic: I/O
Question: Print Hello World! on the terminal.

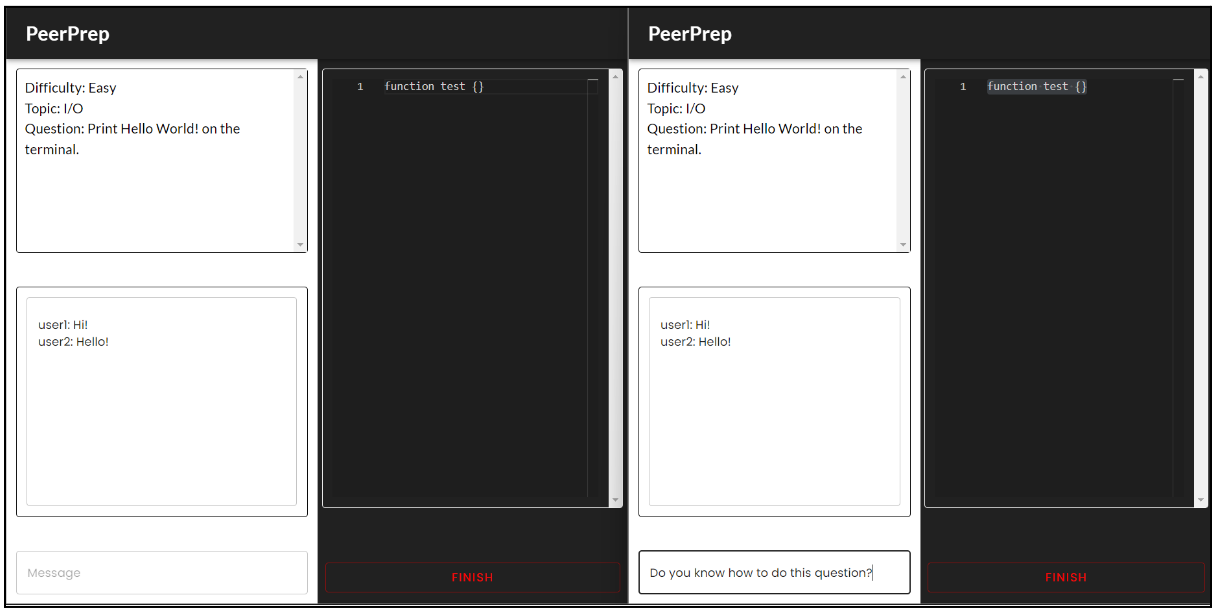
```
function test {}
```

Difficulty: Easy
Topic: I/O
Question: Print Hello World! on the terminal.

```
function test {}
```

Code Editor Changes

Anytime they type in the code editor, it will also get updated on the other user's session page as well.



Difficulty: Easy
Topic: I/O
Question: Print Hello World! on the terminal.

```
function test {}
```

user1: Hi!
user2: Hello!

Message

Difficulty: Easy
Topic: I/O
Question: Print Hello World! on the terminal.

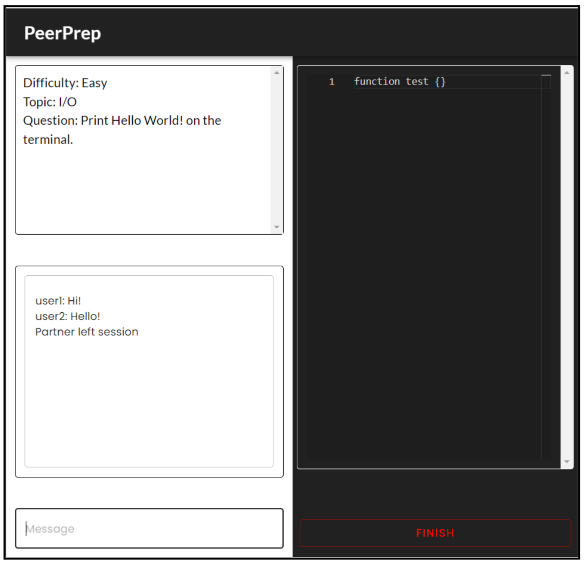
```
function test {}
```

user1: Hi!
user2: Hello!

Do you know how to do this question?

Chatbox Changes

They can send a message as well to the other user via the chat box by typing in the message and pressing the Enter key.



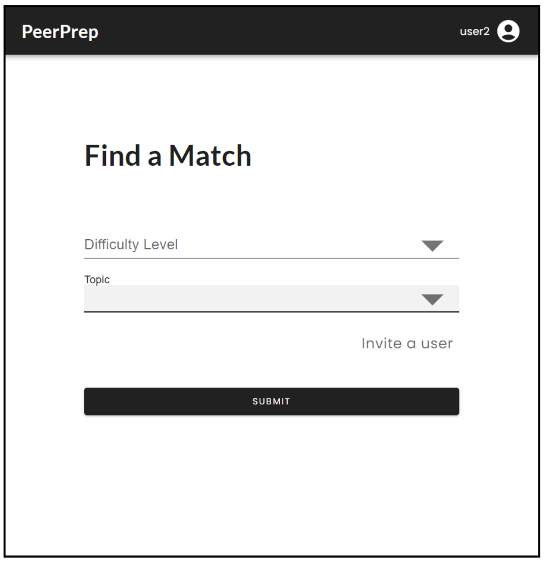
Difficulty: Easy
Topic: I/O
Question: Print Hello World! on the terminal.

```
function test {}
```

user1: Hi!
user2: Hello!
Partner left session

Message

User1 Still In Session Page



User2 Finishing The Session

Once the user is satisfied with the session, they can simply leave the session page by pressing the Finish button. They will go back to the Home Page where they can pick their difficulty level and topic again. The other user will be notified in the chat when their partner left the session.

7. Challenges Faced

This section covers some of the main challenges faced during development and deployment of the application.

1. Learning new technology such as Socket.io, MongoDB Atlas, React, Google Cloud, etc. Managed to overcome by taking some time researching and understanding how they work.
2. Integrating CD and deploying the application of a Microservice architecture.
    a. Management and setting of the environment variables were very different between deployment and local build.
    b. Locally building it using the HTTP protocol and then deploying it to an endpoint using the HTTPS protocol broke a couple of features initially.

8. Future Enhancements

This section covers some possible enhancements in the application.

Firstly, data persistence is not fully implemented where in the case if collaboration-service goes down, the data will not be saved. This is not a main priority in the application because sessions are often short-lived, and the user pool is not large enough to likely cause any downtime on our services. A further improvement however is to implement a message queue that supports persistence messages instead (like RabbitMQ).

Secondly, another useful feature to have might be the ability to compile and run code on the application itself. This will allow users to verify their code against some test cases immediately while coding. This will also open the possible extension of handling different code languages as well. That is, users will be able to select the language they would like to code in and the editor will handle the syntax based on the selected language.

Another possible extension would be to include voice communication features during coding sessions. Since one of the goals of the application is to help users explain their thought process while coding, a voice communication feature would enable users to do so more effectively as it can be quite tough to code and communicate ideas via just a text chat.

Next, a possible extension could be utilising a login API in the user service. For example, users can login via their Google account instead. This way, it would not only be easier for users to access the application, it will be easier for us to manage accounts and authentication as well since these accounts are handled externally.

9. Learning Points

Overall, the project has exposed us to many new technologies as well as the opportunity to apply the various design principles and patterns taught in the module. The workload of the project is manageable albeit being overwhelming at times because we were only given about 9 weeks to learn the various new technologies, apply, deploy them while still having to keep track of the project management side of things.

Creating a CI/CD pipeline for the first time, although was very difficult, the team can clearly see the benefits once it was implemented, seeing the code changes from Github being automatically being deployed to the end product.

Overall, the project was a fulfilling and meaningful one. Being able to work on a project and develop an application in a module usually turns out to be a rewarding experience.

Appendix

A. Use Cases

Title		**UCU-01 Register User**	
Goal		Register user account	
Description		The user registers for an account which includes username, email address and password.	
Actor(s)		User	
Triggering Event		User visits the PeerPrep and selects create account.	
Basic Flow			
Preconditions		Username and email address has not been registered before.	
1	System	Wait for the user to input username, password and email address.	
2	User	Input username and password.	Alternative Flow: 2b
3	System	Checks whether username and email address are already registered.	
4	System	Create an account based on input username, password and email address.	Alternative Flow: 4b
5	System	Check email address and verify account.	
6	System	Sign in user to new account with email address and password.	
Post Conditions(s)		New account successfully created and user is signed in to account.	
Abort Alternative Flow in case of abort: Alternative Flow 2b or 4b			
Alternative Flow 2b			
Precondition(s)		Username or email address already registered.	
2b1	System	Show create account fail.	

2b2	System	Prompt user to input new username and email address.	
2b3	User	Input another username or email address.	
Post Conditions(s)		-	
Alternative Flow 4b			
Precondition(s)		-	
4b1	User	User cancels create account.	
4b2	System	Shows create account has been cancelled and redirects the user to the sign in page.	

Title		**UCU-02 Sign In User**	
Goal		Sign in to registered account	
Description		The user signs into PeerPrep using a registered account.	
Actor(s)		User	
Triggering Event		User visits the PeerPrep.	
Basic Flow			
Preconditions		User has an existing verified PeerPrep account.	
1	System	Wait for the user to input email address and password.	
2	User	Input email address and password.	
3	System	Checks whether username and password are valid.	
4	System	Sign in user to account and generate authentication token.	Alternative Flow: 4b, 4c, 4d
Post Conditions(s)		User signed in successfully to account.	
Abort Alternative Flow in case of abort: Alternative Flow 4b, 4c and 4d.			

Alternative Flow 4b		
Precondition(s)		Email address not registered in the system.
4b1	System	Show email address or password invalid/unverified
4b2	System	Prompt user to retry
4b3	User	Input another email address and/or password
Post Conditions(s)		-
Alternative Flow 4c		
Precondition(s)		Password invalid
4c1	System	Show email address or password invalid/unverified
4c2	System	Prompt user to retry
4c3	User	Input another email address and/or password
Alternative Flow 4d		
Precondition(s)		Email address not verified
4d1	System	Show email address or password invalid/unverified
4d2	System	Prompt user to retry
4d3	User	Check email address for verification link and verify email
4d4	User	Input email address and password

Title	**UCU-03 Log out User**
Goal	Log out of account
Description	The user logs out from an account.
Actor(s)	User
Triggering Event	User selects log out when signed in to an account.

Basic Flow		
Preconditions		User is currently signed in to an account.
1	System	Wait for the user to select log out.
2	User	Selects log out.
3	System	Returns user to sign in page and blacklists authentication token.
Post Conditions(s)		User logged out successfully.

Title		**UCU-04 Delete Account**
Goal		Delete an existing account.
Description		The user deletes their account.
Actor(s)		User
Triggering Event		User selects delete account when signed in to an account.
Basic Flow		
Preconditions		User is currently signed in to an account.
1	System	Wait for the user to select delete account.
2	User	Selects delete account.
3	System	Returns user to sign in page and remove account from database.
Post Conditions(s)		User account deleted successfully.

Title	**UCU-05 Change Password**
Goal	Change password of account.
Description	The user changes their old password to a new password.

Actor(s)		User	
Triggering Event		User selects change password when signed in to an account.	
Basic Flow			
Preconditions		User is currently signed in to an account.	
1	System	Wait for the user to input old and new password.	
2	User	Input old and new password.	
3	System	Checks whether the old password is valid.	
4	System	Change password to new password	Alternative Flow: 4b
Post Conditions(s)		User change password successfully to new password.	
Abort Alternative Flow in case of abort: Alternative Flow 4b and 4c.			
Alternative Flow 4b			
Precondition(s)		Old password is not valid	
4b1	System	Show old password invalid	
4b2	System	Prompt user to retry	
4b3	User	Input another old password	
4b4	System	Jump back to step 3 of UCU-05	

Title	**UCU-06 Reset Password**
Goal	Reset password of account.
Description	The user resets their password via email.
Actor(s)	User
Triggering Event	User selects forgot password.

Basic Flow			
Preconditions		User has a registered account.	
1	System	Wait for the user to input registered account email address.	
2	User	Input email address.	
3	System	Sends reset password email	
4	User	Check email and click reset password link and input new password.	
5	System	Update account with new password.	
Post Conditions(s)		User reset password successfully.	

Title		UCM-01 Match User	
Goal		Match two waiting users.	
Description		The user waits for a matching using difficulty level and topics. Alternatively, the user can invite another user.	
Actor(s)		User	
Triggering Event		Guest visits the PeerPrep and selects matching.	
Basic Flow			
Preconditions		User is logged in to PeerPrep.	
1	System	Shows the selection of difficulty level (Easy, Medium, and Hard) and topics.	
2	User	Selects the difficulty level.	
3	User	Select the topics.	
4	User	Select search for a match.	Alternative Flow 4a
5	System	Find a waiting user that matches the information within 30s.	Alternative Flow 5a, 5b

6	System	Shows a valid match found and puts the users in the same room.	Alternative Flow 6a
Post Conditions(s)		Matching was done successfully.	
Abort Alternative Flow in case of abort: Alternative Flow 5b or 6a.			
Alternative Flow 4a			
Precondition(s)		**-**	
4a1	User	Select invite user.	
4a2	System	Shows email field.	
4a3	User	Enter email of user.	
4a4	User	Select send invitation.	
4a5	System	Sends the invitation and jumps back to step 5.	
Post Condition(s)		User invited successfully.	
Alternative Flow 5a			
Precondition(s)		No matching user was found.	
5a1	System	Shows matching failed.	
5a2	System	Shows rematching.	
5a3	User	Select rematching and jump back to step 1.	
Post Conditions(s)		-	
Alternative Flow 5b			
Precondition(s)		-	
5b1	User	User cancels matching.	
5b2	System	Shows matching has been cancelled and redirects the user to the home page.	

Post Condition(s)		No matching of users.	
Alternative Flow 6a			
Precondition(s)		-	
6a1	User	Selects leave the room.	
6a2	System	Shows room left and redirects the user to the home page.	
Post Condition(s)		Matching was successful but declined by the user.	

Title		UCM-02 Accept Invitation for Matching	
Goal		Match invited user.	
Description		The user is placed in the same room as the invitee.	
Actor(s)		User	
Triggering Event		Guest visits the PeerPrep and selects matching.	
Basic Flow			
Preconditions		User is logged in to PeerPrep. User has been invited by another user.	
1	System	Shows the selection of difficulty level (Easy, Medium, and Hard) and topics.	
2	User	Selects the difficulty level.	
3	User	Select the topics.	
4	User	Select search for a match.	
5	System	Find the inviter within 30s.	Alternative Flow 5a, 5b
6	System	Shows a valid match and puts the users in the same room.	Alternative Flow 6a
7	System	Shows that matching via invitation was successful.	
Post Conditions(s)		Matching was done successfully.	

Abort Alternative Flow in case of abort: Alternative Flow 3b, 4a		
Alternative Flow 5a		
Precondition(s)		**-**
5a1	User	User cancels matching.
5b2	System	Shows matching has been cancelled and redirects the user to the home page.
Post Condition(s)		No matching of users.
Alternative Flow 5b		
Precondition(s)		Inviter was not found.
5b	System	Shows matching failed.
5a2	System	Shows rematching.
5a3	User	Select rematching and jump back to step 1 of UCM-01.
Post Conditions(s)		-
Alternative Flow 6a		
Precondition(s)		-
6a1	User	Selects leave the room.
6a2	System	Shows room left and redirects the user to the home page.
Post Condition(s)		Matching was successful but declined by the user.

Title	UCQ-01 Get a question
Goal	Get a question’s contents including text, and other metadata.
Description	A random question’s contents are returned when provided with a difficulty level and topic.

		Alternatively, a specific question is returned.	
Actor(s)		User	
Triggering Event		Guests need a question.	
Basic Flow			
Preconditions		Guests have been matched and are ready to start collaborating.	
1	System	A random QID should be drawn from the questions available for a specific difficulty level and topic.	Alternative flow 1a
2	User	The question contents should be returned, given the QID found before.	
Post Conditions(s)		Some question’s contents have been returned correctly	
Abort Alternative Flow in case of abort: Alternative Flow 1a.			
Alternative Flow 1a			
Precondition(s)		**-**	
1a1	User	User selects to specify a question rather than get one at random.	
1a2	User	User is prompted and specifies the QID of the question to be used.	
Post Condition(s)		A QID has been specified.	

Title	UCQ-02 Get a list of questions
Goal	Get questions’ contents including text, and other metadata from several questions at once.
Description	A list of questions that satisfy a set of filters are returned when provided a set of filters.
Actor(s)	User
Triggering Event	Guests need a list question.

Basic Flow		
Preconditions		Guests have been matched and are ready to start collaborating.
1	System	Questions are filtered to get a list of questions that satisfy all filters that have been previously provided.
2	User	The questions' contents should be returned.
Post Conditions(s)		Some question's contents have been returned correctly

Title		UCC-03 Initiating a Session
Goal		Start the session between two matched Users
Description		A session will be initiated when two users are matched.
Actor(s)		User A and User B
Triggering Event		User A and B are matched
Basic Flow		
Preconditions		User A and B are logged in to PeerPrep. User A and B are matched.
1	System	Shows User A and B the session page.
Post Conditions(s)		Session between User A and B has started

Title	UCC-04 Finishing a Session
Goal	End the session between two matched users.
Description	One of the user presses the 'Finish' button, indicating the end of the session.
Actor(s)	User A (User who pressed the button) and User B

Triggering Event		User A presses the “Finish” Button
Basic Flow		
Preconditions		User A abd B are logged in to PeerPrep. User A and B are in an ongoing session together.
1	System	Shows the “Finish” button on both User A and B UI.
2	User A	Presses the “Finish” button
3	System	Shows User A the page to select difficulty level again.
4	System	Shows User B the session page with an indication User A has left the session.
5	User B	Presses the “Finish” button
6	System	Shows User A the page to select difficulty level again.
Post Conditions(s)		Session has ended between User A and User B

B. API Specifications

User Service

POST /api/user/createacc
Creates a new user account

Query Parameters:
username: username of account to be created
email: email address of account to be created
password: password of account to be created
 	
Returns (JSON):
If credentials do not make requirement, examples:
{
   "message": "Invalid credentials!",
},
 	
If creating account successful:
{
    "message": "Created new user ${username} successfully!",
},
 	
If input username or email address already in use, examples:
{
   "message": "Username or email address already in use!",
},
 	
If there are missing fields, examples:
{
   "message": "Please fill up all fields!",
},
 	
If unknown error, examples:
{
   "message": "Database failure when creating new user!",
}

POST /api/user/signin
Sign in with a registered account
Query Parameters: email: email address of account to be created

password: password of account to be created

Returns (JSON):
If credentials do not make requirement, examples:
```
{
   "message": “Invalid credentials!”,
},
```

If sign in successful, examples:
```
{
   "message": "Successfully signed in as user!”,
   “username”: username,
   “Token”: token
},
```

If sign in fields invalid, examples:
```
{
  “message”:  "Email address or password incorrect/unverified!”,
},
```

If there are missing fields, examples:
```
{
  “message”:  "Please fill up all fields!”,
},
```

If unknown error, examples:
```
{
   “message”: "Database failure when signing in!”,
}
```

DELETE /api/user/deleteacc/:email
Delete registered account
Query Parameters: email: email address of account to be created Returns (JSON): If delete successful, examples: {   "message": "Deleted email successfull!”, },

If email does not exist, examples:
```
{
  “message”:  "Deleted email unsuccessfull!”,
},
```

If there are missing fields, examples:
```
{
  “message”:  "Please fill up all fields!”,
},
```

If unknown error, examples:
```
{
   “message”: "Database failure when deleting user!”,
}
```

PUT /api/user/changepw/:email

Change password of account

Query Parameters:
email: email address of account to be created
oldPassword: current account password
newPassword: password to change to

Returns (JSON):
If old password incorrect, examples:
```
{
   "message": “Incorrect old password”,
},
```

If change password successful, examples:
```
{
   "message": "Change password successfull!”,
},
```

If there are missing fields, examples:
```
{
  “message”:  "Please fill up all fields!”,
},
```

If unknown error, examples:
```
{
   “message”: "Database failure when deleting user!”,
}
```

PUT /api/user/resetpw/:token
Reset password of account

Query Parameters:
token: reset token
newPassword: password to reset to
 	
Returns (JSON):
If reset password fail, examples:
{
   "message": “Reset password failed!”,
},
 	
If reset password successful, examples:
{
   "message": "Reset password successfull!”,
},
 	
If there are missing fields, examples:
{
  “message”: "Please fill up all fields!”,
},
 	
If unknown error, examples:
{
   “message”: "Unknown Error”,
}

POST /api/user/sendverify/:email
Send verification email
Query Parameters: email: email of account Returns (JSON): If email does not exist, examples: {   "message": “Invalid email”, },

If send email successful, examples:
```
{
   "message": "Email verification email sent!”,
   “token”: token
},
```

If send email unsuccessful, examples:
```
{
  “message”:  "Email verification email failed to sent!”,
},
```

If unknown error, examples:
```
{
   “message”: "Unknown Error”,
}
```

POST /api/user/verify/:token

Verify account

Query Parameters:
token: verification token

Returns (JSON):
If account verify successful, examples:
```
{
   "message": "Email verified successfull!”,
},
```

If account verify unsuccessful, examples:
```
{
  “message”:  "Email verified failed!”,
},
```

If there are missing fields, examples:
```
{
  “message”:  "Please fill up all fields!”,
},
```

If unknown error, examples:
```
{
   “message”: "Unknown Error”,
}
```

POST /api/user/logout

Verify account

Query Parameters:
token: authentication token

POST /api/user/forgotpw/:email

Verify account

Query Parameters:
token: verification token

Returns (JSON):
If reset email sent successfully, examples:
```
{
   "message": "Password reset email sent!”,
   “token”: token
},
```

If email does not exist, examples:
```
{
  “message”:  "Invalid email!”,
},
```

If unknown error, examples:
```
{
   “message”: "Unknown Error”,
}
```

Matching Service

POST / api/matching

Finds a match

Query Parameters:
socketId (String): ID of client socket
email (String): email of user
emailToInvite (String): email of user to invite
difficultyLevel (Number): difficulty level
topic (Number): topic
createdAt (Date): date of matching request

Returns (JSON):
If some error occurred during request validation, examples:

```
{
   "message": “Invalid request”,
},
```

If some error occurred during matching, examples:

```
{
   "message": “Matching failure”,
},
```

If the matching was done successfully, examples:

```
{
   "message": “New match request is added”,
},
{
   "message": “Matching success”,
}
```

Question Service

GET /api/question
Test the question-service connection
Query Parameters: <none> Returns (text): Hello World from question-service

GET /api/question/getQuestion
Fetch a new question
Query Parameters: difficulty (String): the difficulty of the question topic (String): the topic of the question qid (Number): the unique question identifier of the question Note that only (question and difficulty) OR (qid) are needed. If providing the difficulty and topic, a random question will be chosen from that subset of questions. If a qid is provided, that specific question will be returned. Returns (JSON):

If some error occurred, JSON with only one element ("message") will be returned containing the error message.
If the question was fetched successfully, example:

```
{
  "question": {
    "QID": 3,
    "difficulty": "Hard",
    "topic": "I/O",
    "contents": "This is a question."
  }
}
```

GET /api/question/getQuestionsByFilter

Fetch a new question

Query Parameters:
difficulty (String): the difficulty of the question
topic (String): the topic of the question
qid (Number): the unique question identifier of the question
Note that not all query parameters introduced above are needed - any combination of them or none at all is fine.

Returns (JSON):
If some error occurred, JSON with only one element ("message") will be returned containing the error message.
If the questions were fetched successfully, example:

```
{
  "questions": [
    {
      "QID": 3,
      "difficulty": "Hard",
      "topic": "I/O",
          "contents": "Prompt the user for two numbers, multiply them, and present the result on the terminal."
    }
  ]
}
```