CS3219 Final Report
Group 57

Tan Teong Yu, Owen (A0217932A)
He Shuimei (A0209260M)
Sim Wei Shian Benjamin (A0245856R)
Gibson Yip Wei Yang(A0240899M)
Daniel Lim Wei En (A0233730N)

School of Computing, National University of Singapore

CS3219 Software Engineering Principles and Patterns

15 November 2023

Table of Contents

Introduction

Background

In today's competitive job market, securing a desirable position often requires more than just academic achievements. Employers are increasingly emphasizing practical skills, especially in fields such as technology and software development. One crucial aspect of the hiring process is the technical interview, which serves as a direct evaluation of a candidate's problem-solving abilities, coding skills, and communication prowess.

Purpose

Recognizing the need for a comprehensive and collaborative solution to address the challenges of interview preparation, Peerprep was conceived. Peerprep is an innovative platform designed to empower students by providing a space for collaborative learning and skill enhancement. This collaborative app is specifically tailored to meet the unique demands of interview preparation, offering a dynamic environment where students can practice coding questions and receive valuable feedback from their peers.

The core idea behind Peerprep is to harness the collective knowledge and experiences of a community of learners, creating a supportive ecosystem that fosters growth and improvement. The traditional approach to interview preparation often involves individual study, where candidates work in isolation to refine their skills. However, Peerprep understands that the power of collaborative learning can significantly enhance the effectiveness of interview preparation

Project Scope

PeerPrep is a comprehensive web-based platform designed to empower penultimate and final year students from the School of Computing at NUS in their preparation for technical interviews. By focusing on penultimate and final year students, Peerprep aims to provide the most beneficial support through our platform during a critical period when students are actively seeking internship opportunities and transitioning into the workforce.

PeerPrep design encompasses several key functionalities. Firstly, users can effortlessly create accounts and log in, facilitating seamless engagement with the platform. The core feature involves a sophisticated matching system, enabling users to connect with peers on PeerPrep. Once matched, users gain access to a coding sandbox that presents potential technical interview questions. This interactive environment allows real-time collaboration, where users can collectively work on problem-solving, leveraging the opportunity to learn from each other during the process. PeerPrep thus provides a dynamic space for collaborative learning and skill enhancement in tackling technical interview challenges.

Work Distribution

Individual Contributions

Name	**Technical & Non-technical Contributions**
Tan Teong Yu Owen	**Technical Contributions** Question Service, User Service, Integration of Frontend & Backend, Continuous Integration, Deployment **Non-technical Contributions** Documentation of architecture, tech stack, question service, user service, CI/CD
He Shuimei	**Technical Contributions** Frontend design and implementation, Integration of Frontend & Backend, Propagation of questions **Non-technical Contributions** Project Report: Documentation of frontend and related technologies: frontend techstack, diagrams, and screenshots. Future enhancement, Group reflection
Sim Wei Shian Benjamin	**Technical Contributions** Matching service **Non-technical Contributions** Documentation of matching service
Gibson Yip	**Technical Contributions** Collaborative Code Editor, Execution of code in sandboxed environment and serverless leetcode question service on GCP **Non-technical Contributions** Documentation of technologies specified above
Daniel Lim	**Technical Contributions** Video Call for Communication, Automated Integration Testing using Cypress for CI pipeline **Non-technical Contributions** Documentation for video call and testing, Project Report

Sub-Group Contributions

Sub-Group	**Nice-to-have Features**

Tan Teong Yu Owen Gibson Yip	N5: Improved code editor N6: Enhance collaboration service N9: Deployment of the app on AWS
He Shuimei Sim Wei Shian Benjamin Daniel Lim	N1: Communication service N4: Enhance question service N8: Unit, integration, and system testing using CI

Application Requirements

This section details the functional and non-functional requirements of our application and its priority associated with them.

Priority	
High	High-priority requirements are **crucial for the app's core functionality** and are directly aligned with its primary objectives. They often include features that are essential for the app to function as intended or meet its primary goals.
Medium	Medium-priority requirements are **important for enhancing the app's functionality**, usability, or user experience. While not critical for the core functionality, they contribute significantly to the overall value of the app.
Low	Low-priority requirements are **considered nice-to-have features** that could enhance the app's overall appeal but are not critical for its core functionality or immediate success.

Functional Requirements

Tag	Description	Priority	Implemented
F1: User Service – Responsible for user profile management			
F1.1 User Registration			
F1.1.1	• Users should be able to create an account by providing their name, email, and password. • System should validate the uniqueness of email addresses. • Users should receive a confirmation email to verify their account.	**H**	✅
F1.1.2	Use OAuth to offer flexibility to users for authentication	**H**	✅

F1.2 User Login			
F1.2.1	Registered users should be able to log in with their email and password	**H**	✅
F1.2.2	Implement user authentication and session management.	**H**	✅
F1.3 User Password Recovery			
F1.3.1	Users should have a way to recover their password by using their emails in the case of loss of their password.	**M**	✅
F1.4 User Questions Track Record			
F1.4.1	Maintain a record of the questions attempted by the user e.g., maintain a list of questions attempted along with the date-time of attempt, the attempt itself and/or suggested solutions.	**L**	❌
F2: Matching Service – Responsible for matching users based on some criteria			
F2.1 Criteria Selection			
F2.1.1	User should be able to select their preferred question difficulty	**H**	✅
F2.1.2	User should be able to select their preferred question topic	**L**	❌
F2.2 Matching Algorithm			
F2.2.1	• Develop a matching algorithm that pairs users based on their selected difficulty level or/and topic. • Consider a reasonable timeout period for matching. • Implement a notification system to inform users of a successful match.	**H**	✅
F2.3 Timeout Handling			
F2.3.1	• If a user is not successfully matched within the specified duration, they should receive a timeout notification. • Allow users to retry or choose different preferences.	**M**	✅
F3: Question Service – Responsible for maintaining a question repository indexed by difficulty level (and any other indexing criteria e.g. specific topics)			
F3.1 Questions Serving			
F3.1.1	• A pair of users should be able to receive a question based on their preferred difficulty level upon matching. • System should be able to select a question from the	**H**	✅

	pair of users' preferred difficulty level and serve it to them.		
F3.1.2	• A pair of users should be able to receive a question based on their preferred question topic upon matching. • System should be able to select a question from the pair of users' preferred topic and serve it to them.	**L**	❌
F3.2 Questions Selection			
F3.2.1	System should be able to serve a pair of users a question matching their criteria.	**H**	✅
F3.1.2	System should be able to serve a pair of users a unique question each time they use the app.	**L**	❌
F3.3 Error Handling			
F3.3.1	System should be able to validate questions based on required fields such as description or title	**H**	✅
F3.3.2	System should be able to handle duplicate questions based on title	**H**	✅
F3.3.3	System should be able to handle the case when there are no valid questions in the questions repository to serve to the user. As in the case of: • User requesting for a topic that is not present. • User requisition for a difficulty level that does not exist. • User makes a request for a question of a particular criteria and they have already done all the questions of that criteria such that the system cannot serve them a unique question.	**L**	Not applicable as we do not store history of user currently
F3.4 Question Management (N4)			
F3.4.1	System should provide some way for developers to create, update and delete questions.	**H**	✅
F3.4.2	System should provide a way to get more questions from an online question database such as LeetCode.	**L**	✅
F3.4.3	System should provide a way to tag questions by some criteria such as topic or popularity, etc.	**L**	✅
F3.4.4	System should provide a way for a pair of users to request additional questions to solve after they have been matched	**M**	✅

	and put into the collaboration space.		
F4: Collaboration Service – Provides mechanism for real-time collaboration (concurrent code editing) between matched users			
F4.1 Collaborative Space			
F4.1.1	• System should provide a collaborative space (in the form of a code editor) in which two matched users can concurrently edit code. • Anything one user types should immediately be visible to the other user and vice versa.	**H**	✅
F4.1.2	System should have a mechanism for users to stop the collaboration at any time and return each user to the landing page.	**M**	✅
F4.1.3	System should provide a way for users to continue their collaboration and request new questions after completing a question.	**L**	✅
F4.2 Error Handling			
F4.2.1	System should be able to handle unexpected errors when two users are collaborating. As in the case of: • One or both of the users unexpectedly disconnecting or quitting. • One user being inactive for a certain period of time such that the active user cannot collaborate effectively. The system can handle this by returning both users to their respective landing page.	**H**	✅
F4.3 Communication (N1)			
F4.3.1	The system should provide a mechanism to facilitate real-time communication between the users in the collaborative space	**H**	✅
F4.3.2	The system should provide video/voice-calling service for collaboration.	**L**	✅
F4.4 Improved Code Editor (N5)			
F4.4.1	Collaboration space should include an enhanced code editor with code formatting, syntax highlighting for one main language.	**M**	✅

F4.4.2	Collaboration space should include an enhanced code editor with language support for multiple popular languages.	**L**	✅
F4.5 Code Execution (N3)			
F4.5.1	Collaboration space should include a mechanism for users to execute code in a sandboxed environment	**H**	✅
F4.5.2	Collaboration space should include a mechanism for users to execute code in a sandboxed environment against test cases	**M**	❌
F4.5.3	Collaboration space can display said results to the users.	**L**	✅
F5: App UI			
F5.1 Register/Login			
F5.1.1	Provide simple register interface	H	✅
F5.1.2	Provide simple login interface	H	✅
F5.1.3	Input validation for register/login interface	L	✅
F5.2 Code Editor			
F5.2.1	Provide a basic code editor for user	**H**	✅
F5.2.2	Improved code editor with code formatting, syntax highlighting for one language, syntax highlighting for multiple languages	**L**	✅
F6: Deployment			
F6.1 Testing			
F6.1.1	Unit Testing	M	✅ Testing of components
F6.1.2	Integration Testing	H	✅ End-to-end testing
F6.2 CI/CD			
F6.2.1	Add build pipeline with automated testing	M	✅
F6.2.2	Add Git hooks	L	✅
F6.3 Integrate with Cloud Native Solutions (AWS/GCP)			
F6.3.1	Deployment of app to cloud	H	✅

Non-functional Requirements

<table>
<tr><th colspan="4">Non-Functional Requirements</th></tr>
<tr><td colspan="4">NFR1 Performance</td></tr>
<tr><td>NFR1.1</td><td>The system should respond to user behavior on the website within 2s.</td><td>H</td><td>✅</td></tr>
<tr><td>NFR1.2</td><td>System should support a minimum of 50 active users.</td><td>M</td><td>✅</td></tr>
<tr><td colspan="4">NFR2 Security</td></tr>
<tr><td>NFR2.1</td><td>User data, including login credentials and collaborative session data should be encrypted in transmission and storage.</td><td>H</td><td>✅</td></tr>
<tr><td>NFR2.2</td><td>The system should have strong authentication and authorization mechanisms to ensure only authorized users can access the system.</td><td>H</td><td>✅

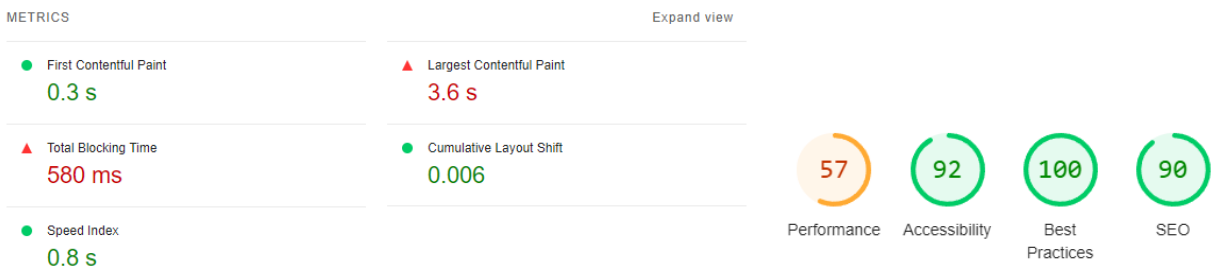
Session management in the form of cookies</td></tr>
<tr><td colspan="4">NF3 Scalability</td></tr>
<tr><td>NFR3.1</td><td>The platform should handle a growing number of users and questions without significant degradation in performance.</td><td>H</td><td>✅</td></tr>
<tr><td colspan="4">NFR4 Usability</td></tr>
<tr><td>NFR4.1</td><td>The platform should ensure that the interface to match with other users is intuitive such that a user can easily match with another user</td><td>H</td><td>✅</td></tr>
<tr><td>NFR4.2</td><td>The platform should provide some feedback on certain actions such as the following:
1. When joining queue to match
2. Disconnecting from queue
3. Add/Edit question form validation</td><td></td><td>✅</td></tr>
<tr><td>NFR4.3</td><td>As part of user experience, the system should respond to user behavior on the website within a certain threshold of time.

This is as according to the table below:
<table><tr><th>Behavior</th><th>Threshold (s)</th></tr><tr><td>Navigation, Clicking,</td><td>0.1</td></tr></table></td><td></td><td>✅</td></tr>
</table>

Typing, Chat, Typing in Code Editor	
Login, Registration, Logout	2
Matching	30
Serving Questions, Starting Collaboration, Ending Collaboration	2

Performance metrics

Lighthouse is an open-source tool for auditing the performance of web pages.



Metrics of Dashboard Page

We use the following metrics to measure our performance:

- **First Contentful Paint (FCP)** - measures the time from when the page starts loading to when any part of the page's content is rendered on the screen
- **Speed Index** - measures how quickly content is visually displayed during page load
- **Total Blocking Time** - measures the total amount of time between FCP and Time to Interactive (TTI)

	Landing Page	Dashboard	Collaboration Space
First Contentful Paint	0.2s	0.3s	0.4s
Speed Index	0.5s	0.8s	0.6s
Total Blocking Time	0.31s	0.58s	0.43s

These metrics show that our web app is able to render pages within 1s and meets NFR1.1 requirements.

In order to serve users with minimal lag, we had to ensure our APIs had low latency. Below are the various APIs we tested and the average response times. This is calculated by taking the average of the response time of 100 requests.

	Average Response Times
Fetch User profile	65ms
Fetch Question	49ms
Fetch All Questions	291ms

Since our APIs run on serverless functions, the functions can be scaled up or down to handle a larger number of requests, thereby meeting NFR1.2.

Tech Stack

Component	Technologies Used
Frontend	React, NextJS, Chakra-ui, Bulma
Backend	NodeJS, Express, Socket.IO, Yjs, WebRTC, Judge0, Agora
Database	DynamoDB
CI/CD	GitHub Actions
Testing	Cypress
Deployment	AWS CDK: Infrastructure as Code AWS Amplify: Frontend AWS Lambda: Question service, user Service, Video service AWS Elastic Beanstalk: Collaboration service, Matching service

Frontend

React, NextJS

When it comes to building single-page applications (SPAs), React is one of the best choices as it offers flexibility in creating components for a web application. Its popularity also meant that our team can benefit from its vast ecosystem as many libraries support React. Many of our team members are familiar with React for web development and it is also friendly for beginners to learn.

NextJS builds on top of React and provides additional features such as strategising rendering on a per-page basis. With server-side rendering, we can improve the user's experience as some pages can be rendered on the server. With content being rendered on the server, we can also optimize our website for Search Engine Optimization. NextJS also offers a file-based routing system and simplifies the creation of API routes. This makes it easy for us to organize and structure our application.

Chakra-UI, Bulma

Chakra UI is a versatile and highly customizable React component library which greatly streamlines the process of building responsive user interfaces. As it is built for React, it is easily integrated and compatible with NextJS and empowers developers to create visually appealing designs with its extensive component selection. It seamlessly combines style props and utility functions, making it an efficient and powerful library when working with user interfaces in React.

Bulma is a modern, open-source CSS framework that simplifies the process of styling user interfaces. It is simple, flexible and easy to use and learn. Bulma offers a comprehensive set of CSS classes and components. Bulma's modular structure and grid system provides a solid foundation for constructing responsive designs across various screen sizes. The comprehensive documentation also makes the CSS framework easily accessible and referenced for developers to create visually appealing websites. Containing only CSS classes, Bulma complements the existing component library by offering even more flexibility in styling selected components.

Backend

NodeJS

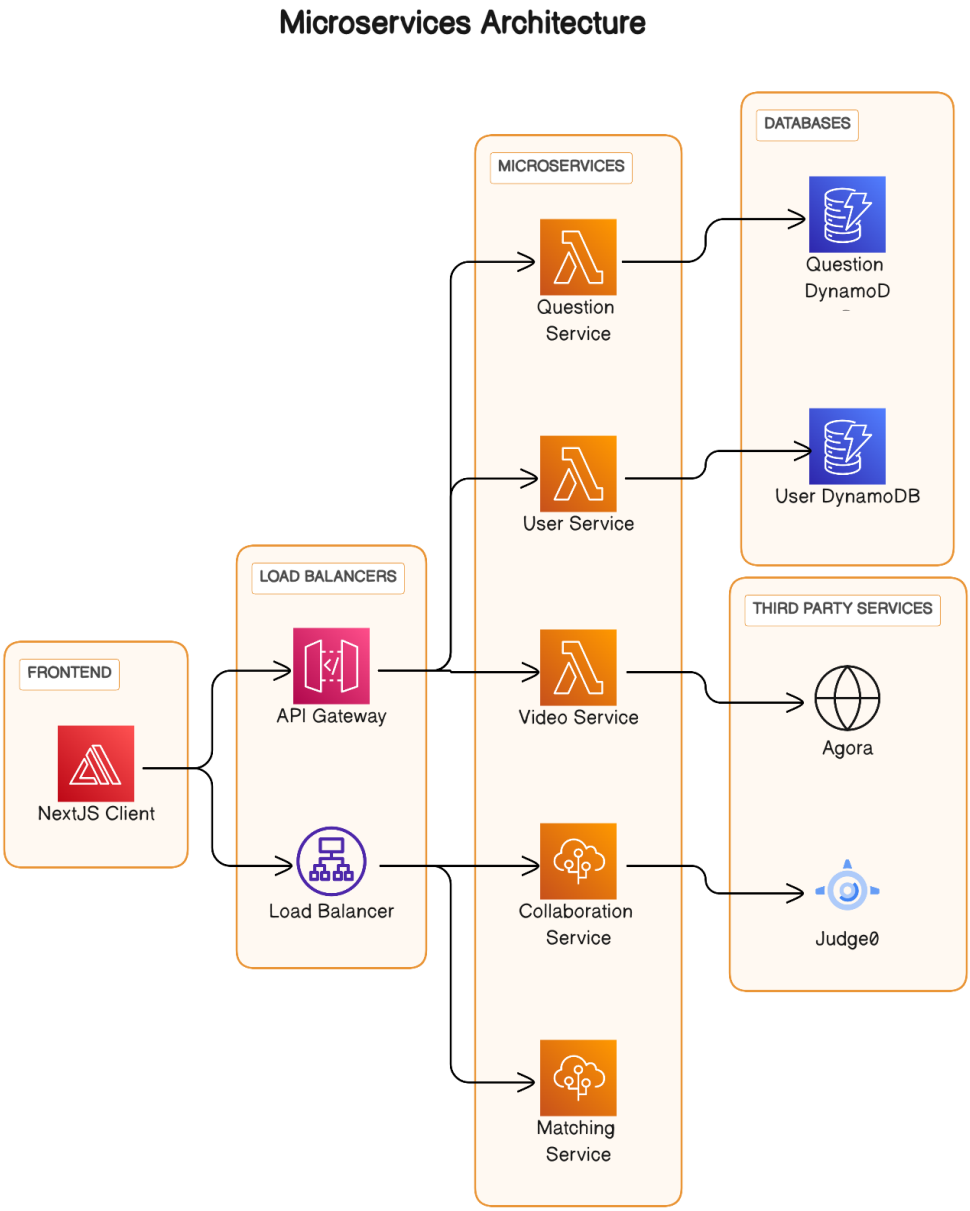
NodeJS provides us with the ability to write server-side code without having to learn a new language as it relies on JavaScript. It is built on the V8 JavaScript engine from Google, which compiles JavaScript directly into machine code, making it very performant. Since we will be using a microservices architecture where our application is composed of many small and independent services, its light-weight nature is a good fit for us.

Express

Express is a NodeJS web application framework that allows for simplicity when it comes to developing APIs for web applications. Express leverages on the benefits of NodeJS and is designed to be minimalistic and opinionated, allowing flexibility for us to structure our microservices. It has a robust middleware system that can handle authentication, logging, error handling etc.

Architecture

PeerPrep leverages a microservices architecture to deliver a scalable and efficient platform for users. This architectural approach divides the application into independent, modular services, each responsible for specific functionalities, which allows for flexibility for developers developing each microservice.



High Level Architecture Diagram

When designing the architecture of our application, we considered monolithic architecture and microservices architecture. Despite both architectures having their advantages and challenges, we decided to choose microservices over a monolithic architecture for the reasons below.

Firstly, microservices architecture allows for independent scaling of different services. In PeerPrep, various features may experience different levels of usage. Adopting microservices can allow us to scale specific services such as our collaboration service when there is a high number of concurrent users, optimizing resource utilization.

Secondly, microservices architecture aligns well with our agile methodology as it makes it easier to rapidly evolve and maintain the app over time. Adding new features or updating existing ones can be done more incrementally and team members can focus on specific services without having to touch the entire application. This ultimately reduces coupling and adheres to separation of concerns.

Lastly, microservices allow for fault isolation. If a microservice fails, the impact is limited to that specific service and the rest of the application continues to function. This contributes to higher fault tolerance and resilience.

Despite mentioning these advantages, we do acknowledge the downsides of employing a microservices architecture. Microservices may not provide immediate development speed benefits, especially in the initial stages of a project. It requires proper setup and coordination, which can be quite an overhead. Microservices introduce a distributed system architecture, which can be complex to design, implement and manage. It can be challenging to coordinate multiple services. However, we believe that as the app evolves over time, we are able to reap the benefits of scaling our services independently and rapidly develop new services.

In the following sections, we go into details of each microservice and the various technologies and developments it employs.

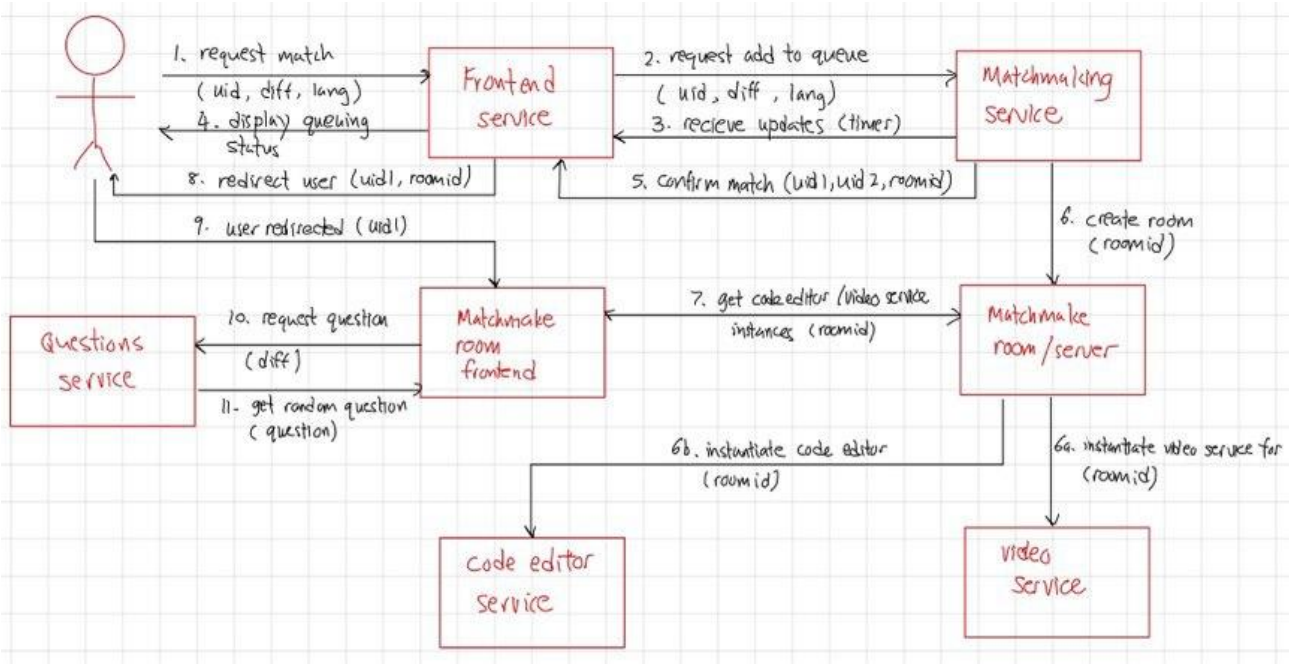
Frontend

User Interface development

The frontend user interface serves as the central hub for orchestrating interactions between users and the various other microservices in the PeerPrep.

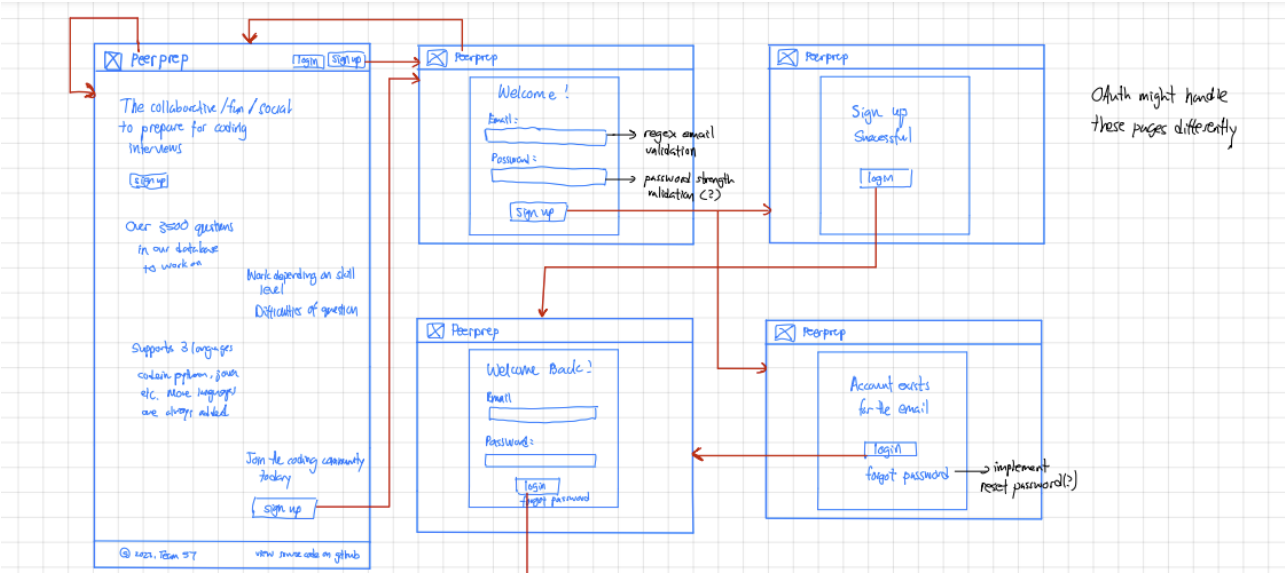
To design the sitemap and determine the page flow, it is crucial to grasp how each microservice interacts with one another and how they integrate together on the Frontend. This necessitates a comprehensive grasp of both the user journey and the data flow across the microservices

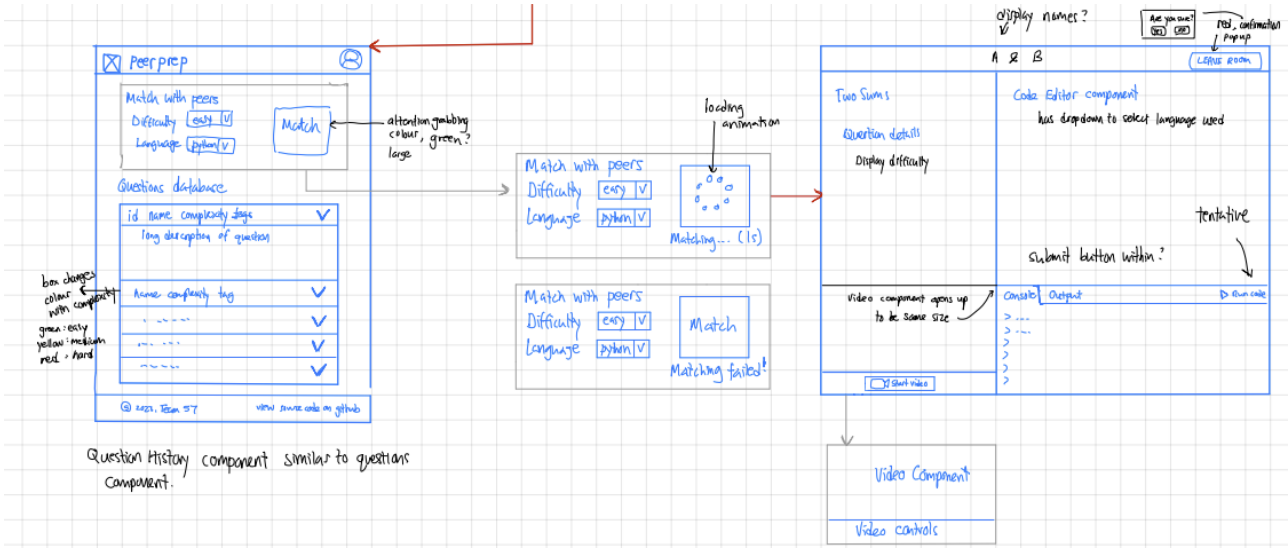
To achieve this understanding, a low-fidelity mockup of how a user would interact with PeerPrep and the subsequent data flow.



User & Data Flow Diagram

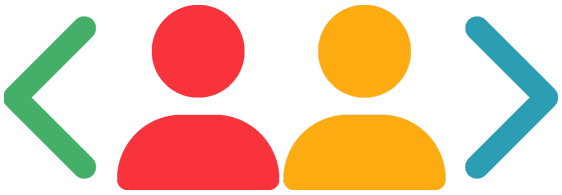
After achieving a baseline understanding with the group on how the overall PeerPrep project would be like, low-fidelity wireframes were then drawn up. The wireframes were crafted based on the insights gathered from different team members, who contributed their expertise to delineate how components within their microservices converge on the Frontend.





Low-fidelity hand-drawn wireframes

Leveraging the wireframes, we expedited the development of the frontend user interface, swiftly prototyping it with HTML and Bulma CSS to provide easy to use styling libraries. As development progressed, Chakra UI played an instrumental role in furnishing versatile and customizable components.



PeerPrep Logo

A simple, yet thoughtfully crafted logo was conceptualized for the application portraying two users representing team collaboration, sandwiched between two angled brackets which signifies the coding elements. The theme of the colors chosen is Energy, making use of bright and vibrant colors to signify fun and excitement.

Screenshots of PeerPrep can be found in the Appendix:Frontend

Code Editor

The Monaco Editor is a powerful and flexible code editor that can be embedded into web applications. Developed by Microsoft, it provides features like syntax highlighting, autocompletion, error checking, and more, making it an excellent choice for building custom code editors.

Video Call

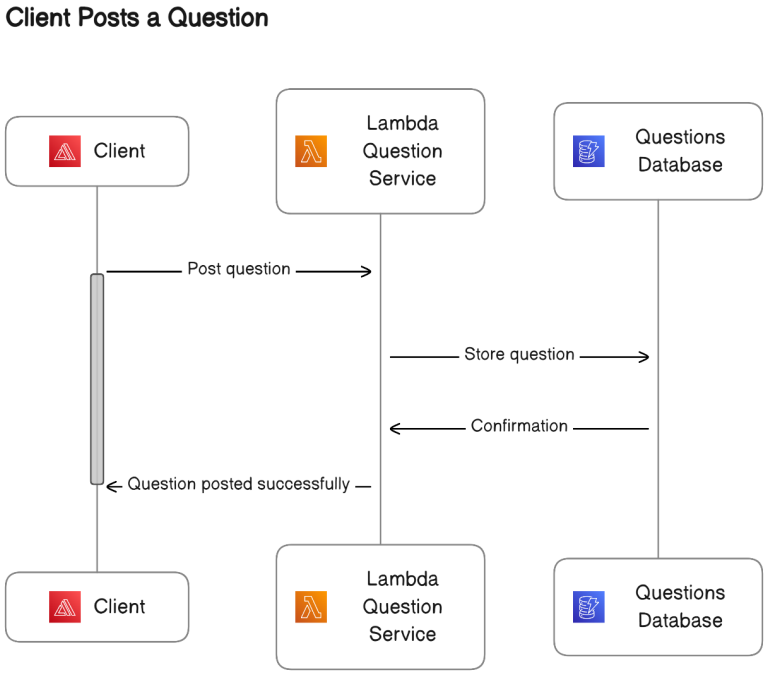
We made use of the Agora platform to enable video conferencing in our web application. The Agora Video SDK makes it easy to embed real-time video chat into web, mobile and native apps. And thanks to Agora's intelligent and global Software Defined Real-time Network, we can integrate high-quality, low-latency video calling features into our application. Moreover, we used the Agora Web UIKit which is built on top of the Agora Video SDK using React in order to streamline the addition of video conferencing into our application.

Backend

Question service

Question service is a fundamental component responsible for managing and providing access to a collection of programming and algorithmic questions for users. Questions are categorized based on their difficulty levels (easy, medium, hard) and tagged with relevant topics, algorithms or data structures. This helps users find questions that match their skill levels and areas of interest.

Question service exposes a RESTful API that our web client can interact with. The apis support CRUD operations with questions and also fetching random questions of a specified difficulty that is used by collaboration service.



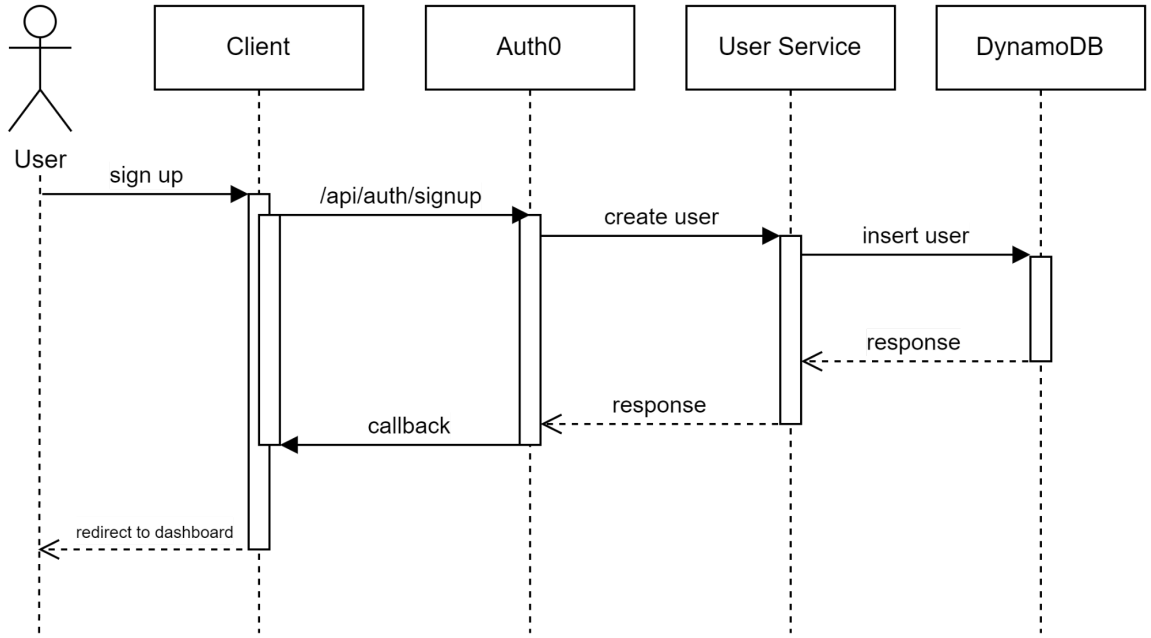
Sequence Diagram of Posting a Question

The sequence diagram above encapsulates the flow of events when a user creates a question.

1. User posts a question on the web client.
2. Question service receives the request and validates the request.
3. The question is stored in the database and the database provides confirmation.
4. Question service returns a response to inform the user that the question has been created successfully.

This flow is similar when a user reads, updates and deletes a question.

User service



Sequence Diagram for user signup

The user service is responsible for user profile management. We decided to use Auth0 to handle authentication and authorization for our web application. Auth0 supports various authentication methods including username/password and social identity providers such as Google and Github. This provides flexibility for our users to choose how they would like to authenticate with our application. Auth0 follows industry best practices for security and we feel that it is much more secure to use Auth0 to store passwords. Auth0 apis are incorporated into our user service. While Auth0 handles storing of credentials, user service stores user profile data in DynamoDB, such as display names.

The sequence diagram above encapsulates the flow of events when a user signs up.

1. User clicks sign up on the client.
2. User is redirected to the Auth0 Login page.
3. After successfully creating a user, Auth0 sends a request to user service to create a user.
4. User service validates the user profile data and inserts user data into DynamoDB.

5. User service confirms with Auth0 that the user has been created successfully and the callback from Auth0 redirects user to dashboard.

Matching service

The matching service supports matching users based on the question difficulty level selected. Two users will be matched if the question difficulty level they selected is the same.

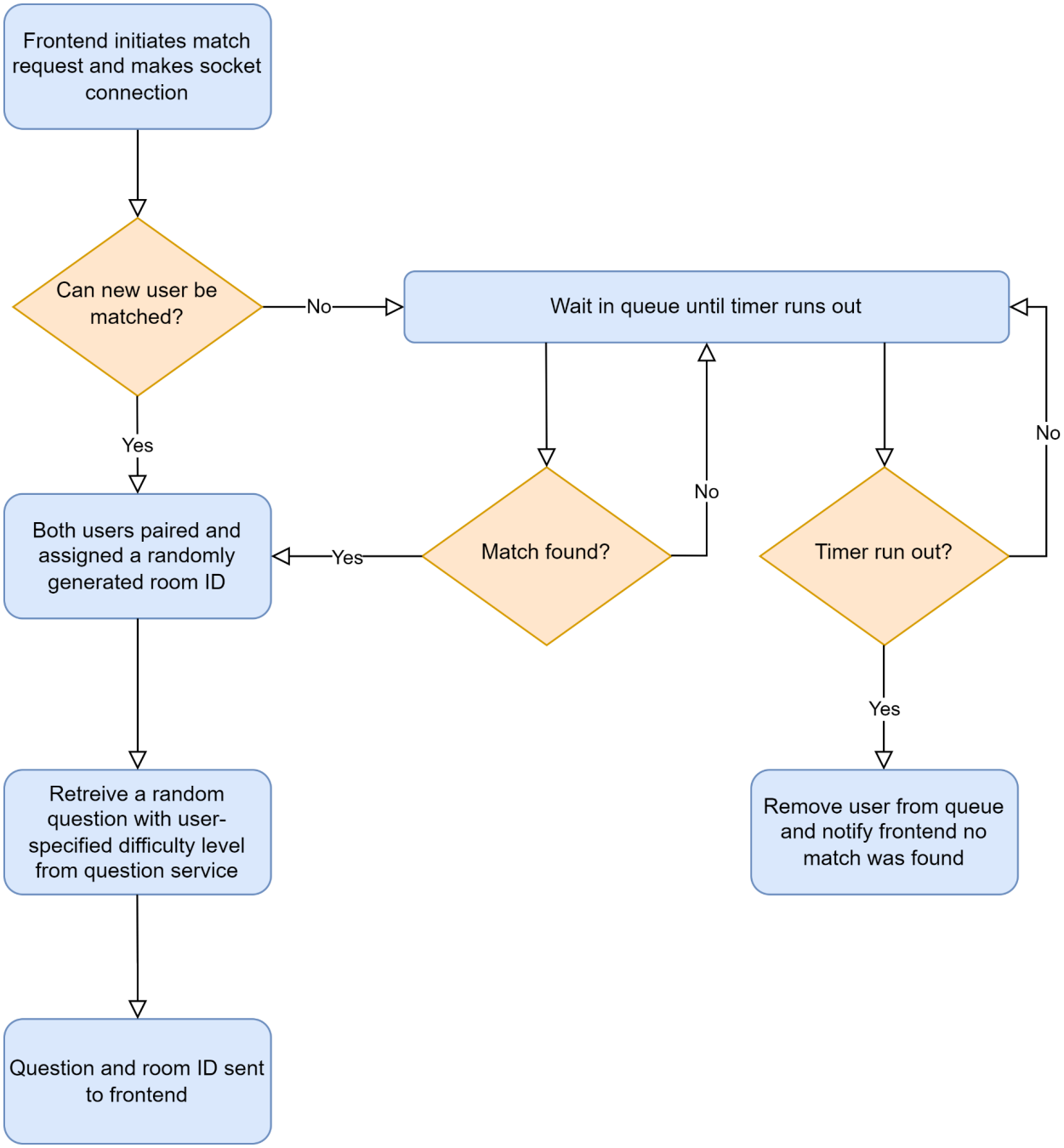
To facilitate the matching process, we used Socket.IO, a NodeJS library that provides bi-directional and low-latency communication using WebSocket. The matching service consists of a single NodeJS server that uses the Socket.IO library. The matching server maintains an internal queue using JavaScript to keep track of the users who are waiting for a match. To ensure users are not kept waiting for a match indefinitely, the matching server will abort the matching process for a user if the user has been waiting in the queue for 30 seconds.

A typical usage scenario is the following: Upon initiating a match request, the frontend makes a socket connection to the matching server. The matching server then checks if the new user can be matched with a user who is already waiting in the queue.

If there is a match, both users are logically paired and a randomly generated room ID is assigned to them. The matching server also retrieves a random question with the user-specified difficulty level from the question service. Both the question and room ID are sent to the frontend, which will then redirect both users to the collaborative room.

If there is no match, the server places the new user in the queue. A timer of 30 seconds is assigned to the new user who then waits in the queue until a match is found or until the timer runs out. Once the timer runs out, the server removes the user from the queue and notifies the frontend that no match was found.

The following flowchart shows the above logic:

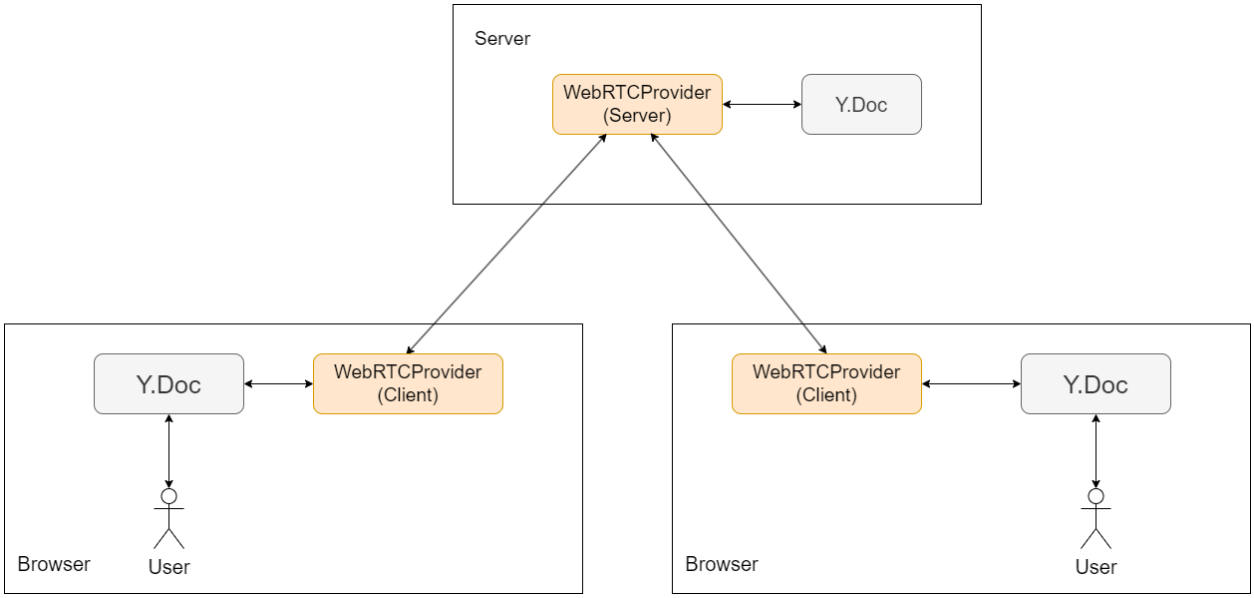

Frontend initiates match request and makes socket connection
Can new user be matched?
No
Wait in queue until timer runs out
Yes
Both users paired and assigned a randomly generated room ID
Yes
Match found?
No
Timer run out?
No
Yes
Retreive a random question with user-specified difficulty level from question service
Remove user from queue and notify frontend no match was found
Question and room ID sent to frontend

Collaboration service

The collaboration service provides a real-time code editor, allowing both users within the ‘room’/interview to code together and solve the given questions.

To achieve this, we relied on Yjs. Yjs is a CRDT implementation that exposes its internal data structure as shared types that can be manipulated concurrently. Furthermore, Yjs has built a rich ecosystem of extensions around Yjs such as ready-to-use editor integrations for many popular text editors(y-monaco), adapters to different network technologies(WebRTC and WebSocket) and persistence providers that store document updates in a database. Undeniably, Yjs offers modular building blocks for building collaborative applications which perfectly serves our needs for the collaborative code editor.

Also, Yjs boasts of being the fastest CRDT implementation by far.



Here is a simple overview on how Yjs works to ensure Yjs documents are created, modified and synced across clients. Providers replicate data across different Y.Docs and are necessary for all persistence and network transport.

Since each client possesses their own Y.Doc, this helps to act as an implicit cache in the event a particular user disconnects from the room. As the WebRTCProvider will propagate the updated changes from the user still in the room onto the code editor for the user who reconnects to the same room.

Code Execution

Ensuring that code written on the code editor is executed correctly and securely is a priority. To achieve this, we decided to use a Judge0 which is a robust, scalable, and open-source code execution system.

We deployed a docker image of Judge0 onto a Google Cloud Platform virtual machine using Compute Engine. This ensures that the code submitted for execution is compiled and executed within a sandboxed environment, preventing any malicious code from being executed on our local environment. Furthermore, this method allows us to scale up readily as we can just spin up more virtual machines to handle the increased load.

`POST /submissions/{?base64_encoded,wait}`

Status Code	Response

201	{
	"stdout": "SEVMTE8hIQo=\n",
	"time": "0.018",
	"memory": 3112,
	"stderr": null,
	"token": "30868c2d-2f2c-4611-b31a-aa5f1709fc9d",
	"compile_output": null,
	"message": null,
	"status": {
	"id": 1,
	"description": "Accepted"
	}
	}

`GET /submissions/{token}{?base64_encoded,fields}`

Status Code	Response

200	{
	"stdout": "HELLO!!\n",
	"time": "0.018",
	"memory": 3112,
	"stderr": null,
	"token": "30868c2d-2f2c-4611-b31a-aa5f1709fc9d",
	"compile_output": null,
	"message": null,
	"status": {
	"id": 1,
	"description": "Accepted"
	}
	}

A more detailed table is available in the Appendix

Video Service

The video service handles user authentication through the use of tokens whenever a user joins a channel for a video call. The Agora platform which we are using uses tokens to authenticate users. It is a microservice that serves to generate tokens for users on our application's Frontend to connect to the Agora platform in order to enable video conferencing.

When a user has been matched through our matching microservice, they are redirected into a room that we use as our collaborative space. Once here, the Frontend makes an API call to the video service to generate a token specific to the current user and their room. Following that, the video service generates the unique token and returns it to the Frontend. Thereafter, the user now has an authentication token and they can start a call, which will cause the Frontend to connect to the Agora platform using the authentication token and enable the user to join a video call channel and enable video conferencing.

A more detailed look at the interactions between the user, the video service, the Frontend and the Agora platform is shown in the sequence diagram below.

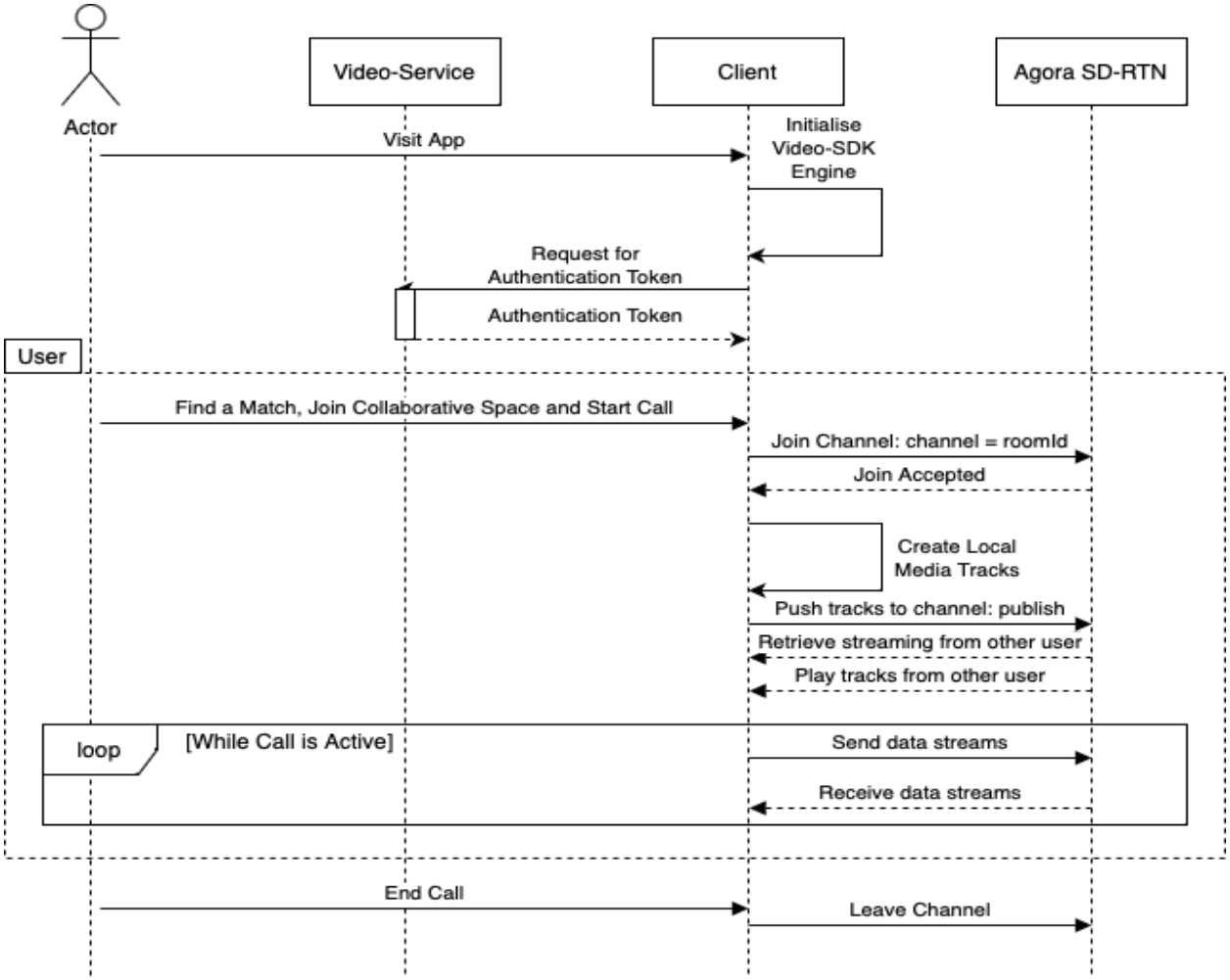


Figure: Sequence Diagram for Video Chat

Leetcode Question Service

The development of the LeetCode question retrieval and database storage code stems from a need to streamline the process of accessing and managing a curated set of coding challenges. LeetCode, a popular platform for honing algorithmic and coding skills, provides a wealth of questions crucial for interview preparation and skill enhancement. Manually selecting, organizing, and updating this set of questions can be time-consuming and error-prone.

To address this, the code automates the retrieval of LeetCode questions through its internal API, ensuring an up-to-date and diverse set of challenges. By leveraging this automated approach, the project aims to improve efficiency, reduce manual workload, and maintain the relevance of the question set over time.

Deploying the LeetCode question retrieval and database storage code on Google Cloud Run offers several advantages, and here's our justification for choosing this deployment option:

1. Serverless Architecture:

Google Cloud Run provides a serverless environment, eliminating the need to manage infrastructure. This allows you to focus solely on your code and its functionality. Serverless architectures are inherently scalable, automatically handling the scaling of resources based on demand.

2. Cost Efficiency:

Cloud Run follows a pay-as-you-go pricing model, where you are billed based on the actual compute resources consumed. This is cost-effective, especially for sporadically used services like a LeetCode question retrieval script. The serverless model ensures that you are not paying for idle resources.

3. Containerization with Docker:

Google Cloud Run supports containerized applications, making it convenient to package your code and its dependencies into a Docker container. This ensures consistency between development and production environments, reducing the likelihood of issues arising due to differences in configurations.

4. Ease of Deployment:

Deploying on Cloud Run is straightforward. The platform automatically handles the deployment, scaling, and load balancing of your containerized application. This simplicity is beneficial for quick iterations, updates, and hassle-free management.

Testing

This section details the testing that our group did in order to ensure quality throughout our entire pipeline. We configured our tests to be automated and to be triggered by GitHub Actions anytime we push to our remote repository so that they are part of our Continuous Integration Pipeline.

Testing With Cypress

Cypress is an end-to-end testing framework built for modern web browsers. It allows developers to test anything that runs within a browser. It also enables developers to write, automate, run and debug tests directly within a browser. We chose to use Cypress as our Frontend testing tool as it allows us to simulate how a user would use our web application from their own browser in real-user scenarios. This lets us write more reliable tests as we can closely simulate how real users will use our application. It also allows us to have more accuracy when testing as we can discover bugs that may appear when real users are using our application and fix these issues before the code is integrated. And finally, allowing us to automate our testing lets us ensure quality throughout our entire pipeline. All these made Cypress a good choice for our testing purposes which is why we decided to use it.

End-to-End Tests

We made use of Cypress and end-to-end testing to simulate user actions and flows throughout our application. The table below details the test specification, the purpose of each test and what we are testing in each test specification for our End-to-End tests.

Test Specification	**Purpose**	**What We Test**
`login_spec.cy.ts`	To simulate a user logging in to our application.	To test that a user is able to login to our application. To test that a user is able to logout from our application. To test that upon logging in, a user is given a session cookie to enable session management.
`login_spec_error.cy.ts`	To simulate a user making mistakes while trying to log in.	To test that a user is shown an error message specifying a wrong username upon typing an invalid username. To test that a user is shown

		an error message specifying a wrong email upon typing an invalid email. To test that a user is shown an error message specifying a wrong password upon typing an invalid password.
`register_error_spec.cy.ts`	1. To simulate a user making mistakes while trying to register for a new account.	To test that in the case where a user does not enter a username, an email and a password, the user is prompted using a validation message to fill in all the fields. To test that a user is unable to register for an account without filling in all the necessary fields of username, email and password. To test that a user cannot register for an account with a username that is already taken and is reminded if he/she tries to do so. To test that a user cannot register for an account with an email that is already taken and is reminded if he/she tries to do so. To test that a user cannot register for an account with a username that does not meet the requirements and is reminded if he/she tries to do so.
	2. To simulate a user entering a password for their account when registering.	To test that a user cannot register for an account with a password that does not meet the password complexity requirements. To test that the password

		complexity requirements are shown to the user who is entering a password and that the password complexity messages are sufficiently detailed so that the user knows the requirements he/she has to meet.
`forget_password_spec.cy.ts`	To simulate a user forgetting their password and wanting to change it.	To test that our application has a page for users to change their password. To test that a user is able to change their password via their email and username that they signed up with and that the page displays sufficient prompts that the user can follow along to change their password.
`home_page_spec.cy.ts`	To simulate a user visiting the home page of the application.	To test that a user is able to visit the url that the application is hosted at.
`profile_page_spec.cy.ts`	1. To simulate a user logging in to our application and visiting the profile page.	To test that once a user has logged in, they are able to visit the profile page which is a protected route. To test that the profile page displays the current user's information correctly, namely their username/display name and their email.
	2. To simulate a user editing their display name and deleting their account on the profile page.	To test that a user is able to edit their display name via a form submission in a modal and that the modal disappears after the user updates their display name. To test that upon submitting the form, the current user's display name is updated to

		their new display name. To test that a user is able to delete their account.
`matching_spec.cy.ts`	To simulate a user selecting a difficulty and then looking for a match to carry out an interview with.	To test that once a user has logged in, they are able to start the matching process which is a protected functionality. To test that a user has access to the connect button to start the matching process and the disconnect button to stop the matching process while it is being carried out. To test that a user is able to select the difficulty of a question that they want for the matching process. To test that once a difficulty has been selected and a user clicks the connect button, the user is shown a countdown timer indicating the progress of their matching and that the matching process begins. To test that the matching process stops automatically if no match is found within the countdown timer time. To test that if a user clicks the disconnect button, the matching process stops and that the user is able to start the matching process again if they would like to.
`question_display_spec.cy.ts`	To simulate a user accessing the landing page and viewing the questions display table.	To test that once a user has logged in, they are able to view the landing page that is protected. To test that the application is able to retrieve the questions

		from the question service and display the questions in the question table for the user to view. To test that the user is able to see questions in the questions table with their necessary titles, categories and complexity tagging. To test that a user is able to view additional details of each question in a modal by clicking the view details button of each question.

Component Tests

The matching process of our application matches two users into a collaborative space which we call a room where they can start a video call, view programming questions to work on collaboratively and code on an interactive code editor. However, matching two users in a room would require us to have two separate accounts logged in at the same time to have both users be matched. A limitation of Cypress is that it does not have this functionality, and as specified in their documentation, there is no reason for them to expose the browser automation APIs.

Therefore, we opted for component tests to test the video conferencing and the code editor components of our application as that would allow us to still test the functionality of these key components of our application. The table below details the test specification, the purpose of each test and what we are testing in each test specification for our Component tests.

Test Specification	**Purpose**	**What We Test**
`CodeEditor_spec.cy.tsx`	1. To simulate a user getting matched into a room and having access to the code editor.	To test that the Code Editor component is able to mount properly. To test that the Code Editor component is displayed to the user and that the user is able to interact with it.
	2. To simulate a user interacting with the code editor.	To test that a user is able to select a language that they want to code in and that the

		syntax highlighting works for the different languages that the user has selected. To test that a user is able to submit the code they have typed into the editor to be run. To test that the Code Editor is displayed to the user and they are able to type on it and whatever is typed is displayed in the code editor. To test that a user is able to submit the code that they have typed into the editor to be run by the code execution section of the application and that the code is run and then the results are returned to the Code Editor and then displayed to the user in the terminal section of the Code Editor.
`UIKitVideoPlayer_spec.cy.tsx`	1. To simulate a user getting matched into a room and having access to the video conferencing functionality.	To test that the Video Conferencing component is able to mount properly. To test that the Video Conferencing component is displayed to the user and that the user is able to interact with it.
	2. To simulate a user interacting with the video conferencing functionality.	To test that a user is able to start a call using the Video Conferencing component. To test that upon starting a call, the Video Conferencing component appears and displays the video and audio feed to the user. To test that a user has access to the control bar while in a Video Conference, where the

		control bar allows a user to start and stop the camera, start and stop the microphone and end the call. To test that a user is able to end a call using the Video Conferencing component.

Test Case Design

The test cases were designed using a black box testing approach that aimed to test our software exclusively from its specified external behavior. We chose to design the tests this way as it would more closely mimic a real user interacting with our application. A real user would not have access to the underlying code or logic of our application. They can only interact with the system through its external behavior and through the exposed functionalities of the system. This, combined with Cypress allowing us to mimic a real user interacting with our application, allows us to design test cases that as closely replicate how a human would interact with our system as possible without having to manually test the system ourselves. So we chose a black box approach to testing to further make our testing replicate a real user's interactions.

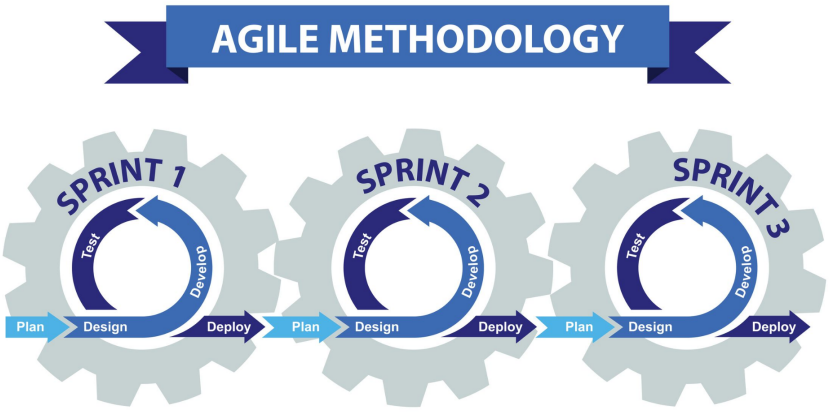
We decided to do end-to-end testing because it was the most efficient way for us to cover as much of the functionality as possible given the time constraints that we had. This is opposed to doing pure component testing for the Frontend and pure unit testing for our microservices. Having the tools to test the application directly in a browser also ensured we could test the application in as real conditions as possible and with all the microservices and the Frontend running. We could test the interaction of the microservices and the Frontend, as well as all the API calls and the Frontend interactions from the browser itself which saves time and gives us more coverage.

We should design tests to maximize their effectiveness, which is how effective the tests are at detecting bugs, and their efficiency, which is how much we can minimize the tests required to maximize the number of bugs detected or in other words the rate of success of a test which is (number of bugs detected / number of test). So the test cases above were designed such that we only tested each major user interaction once in order to minimize the number of tests. We also designed the tests to have sufficient breadth as we tried to get the tests to cover as many main user interactions as possible.

Finally, I used a mix of exploratory and scripted testing to design the test cases. I started with scripted testing by referring to the external specifications of how the application should behave and started designing tests based on that. Thereafter, I explored the application myself and interacted with it and found new flows and interactions to test as well.

CI/CD

Development Process



Agile Methodology Sprint Cycles: credits

Our team decided to adopt an Agile approach to guide our development processes. Sprint cycles is a key component of the Agile methodology. Milestones were broken up into sprints. At the beginning of each sprint, we discuss priorities of backlog items. The team proceeds to work on the selected items and integrate code incrementally. Bugs are also identified and fixed in the subsequent sprint.

For milestone 1, we had to scope the minimum viable product (MVP). It was important that we only allocated what was necessary in the MVP as we were starting from scratch and many of us were unfamiliar with microservice architecture.

For milestone 2, multiple enhancements to services were prioritized. One enhancement would be to add code execution to our collaboration space, allowing users to run their code and visualize the output. It was important that we maximized this milestone to develop various features so that more testing can be done in milestone 3.

By milestone 3, everyone was familiar with the project and could better estimate the efforts of remaining features. As there was not much time left, certain features were decided to be not implemented. Priority was given to fixing bugs and enhancing the usability of the app. We focused more on testing our application and improving the user interface of the app.

Continuous Integration/Continuous Delivery

For styling, we adopted the **Airbnb JavaScript Style guide**. It is one of the most starred JavaScript style guides on GitHub. Every developer writes code differently. As a team, this can reduce code readability and potentially introduce bugs. A style guide ensures that everyone is adhering to the same set of rules. It incorporates the best practices into our code, increasing code quality.

For code quality, we used **Husky**, **Prettier** and **ESLint** to create a powerful combination for maintaining code quality, enforcing consistent coding styles and preventing common issues in a development workflow. ESLint is a static code analysis tool that identifies and enforces patterns in TypeScript. Using ESLint helps to identify potential issues, bugs and code smells. Prettier is a code formatting tool that supports JavaScript and TypeScript. We used Prettier to ensure consistency and adherence to the Airbnb style guide. Husky was used to configure git hooks so that Prettier would run before commits.

GitHub Actions is a powerful CI/CD tool. Using GitHub actions, we defined workflows to trigger upon pull requests and when pushing to master branch. When integrating our code changes into our master branch, a build job is triggered. This job ensures that our code is able to build. Style checks and linting is also performed to ensure that there are no code violations. Lastly, a test job is also triggered to automate testing. This serves as a form of regression testing and informs us when someone commits a code-breaking change and prevents the bug from being integrated into the master branch.

Below is an example of some steps run in our build job:

```yaml
defaults:
  run:
    working-directory: Frontend

steps:
  - name: Checkout
    uses: actions/checkout@master

  - name: Setup node env
    uses: actions/setup-node@v2.1.2
    with:
      node-version: ${{ matrix.node }}
      registry-url: 'https://registry.yarnpkg.com'

  - name: Cache node_modules
    uses: actions/cache@v2
    with:
      path: ~/.cache/yarn
      key: ${{ runner.os }}-node-${{ hashFiles('**/yarn.lock') }}
      restore-keys: |
        ${{ runner.os }}-yarn-

  - name: Install dependencies
    run: yarn install --frozen-lockfile

  - name: Build Taiwind CSS
    run: npx tailwindcss -i ./src/styles.css -o ./dist/output.css

  - name: Build
    run: yarn run build

  - name: Run code style check
    run: yarn run style:all
```

Github Actions Build Workflow

Continuous Deployment

For backend services that are deployed on AWS, we use AWS Cloud Development Kit (AWS CDK) to provision our resources on the cloud. AWS CDK follows the Infrastructure as Code (IaC) paradigm, allowing us to define and manage our infrastructure using code. Although we can directly deploy services on AWS Console, we decided to use IaC as it allows us to document our provision of cloud resources using familiar programming languages such as TypeScript.

Currently, we decided to continue with manual deployment as not all services are always being changed. Since we are rapidly developing, automated deployment would cause unchanged microservices to go down. We wanted to prioritize our app's availability to users and schedule deployments in a way that minimizes impact on users.

Future Enhancement

Given the constrained time frame for completing PeerPrep, there are specific features that not only align with the goals and purpose of Peerprep, and are also suitable candidates for future enhancements.

1. **Enhanced matching options**: Offer users with expanded matching options through: Preferred Coding Language, Spoken Language, Category of Questions, providing users with a more tailored experience, ensuring a better fit with their preferences
2. **More question enhancements**: Enhance usability of the platform with the ability to sort, search and filter questions. Furthermore, streamlining the matching process by removing questions that users have previously completed from the question pool, minimizing redundancy.
3. **History service**: Introducing a dedicated history service that allows users to review their past question attempts and submissions, allowing users to track their development progress over time.
4. **Interactive learning paths**: Introducing a new feature where there are structured goals and paths which encourages users to complete a set of questions.
5. **Gamification**: Incorporate gamification elements like badges, leaderboards and achievements to make the learning experience more engaging and competitive, and motivating users to consistently use the platform.

Group Reflection

Throughout the semester, the project process and our collaborative group work, there were several key reflections and learning points that emerged.

One notable aspect has been the importance of effective communication within the team. Clear and open lines of communication along with our team's dedication to engage and reply to messages and enquiries within the group promptly which has proven instrumental in aligning our individual efforts, ensuring a unified approach to project tasks and the project as a whole.

Additionally the diversity of skills within our group has been a strength, allowing us to leverage a wide range of expertise, embracing varied perspectives. This has been crucial in enriching our problem-solving and decision-making processes.

Moreover, the project timeline emphasized the value of careful planning and time management, given that we were only allowed around 7 weeks to work on the entire PeerPrep project: which included the assignments and its submission videos/documentation, project report, PeerPrep, and preparation for the project presentation.

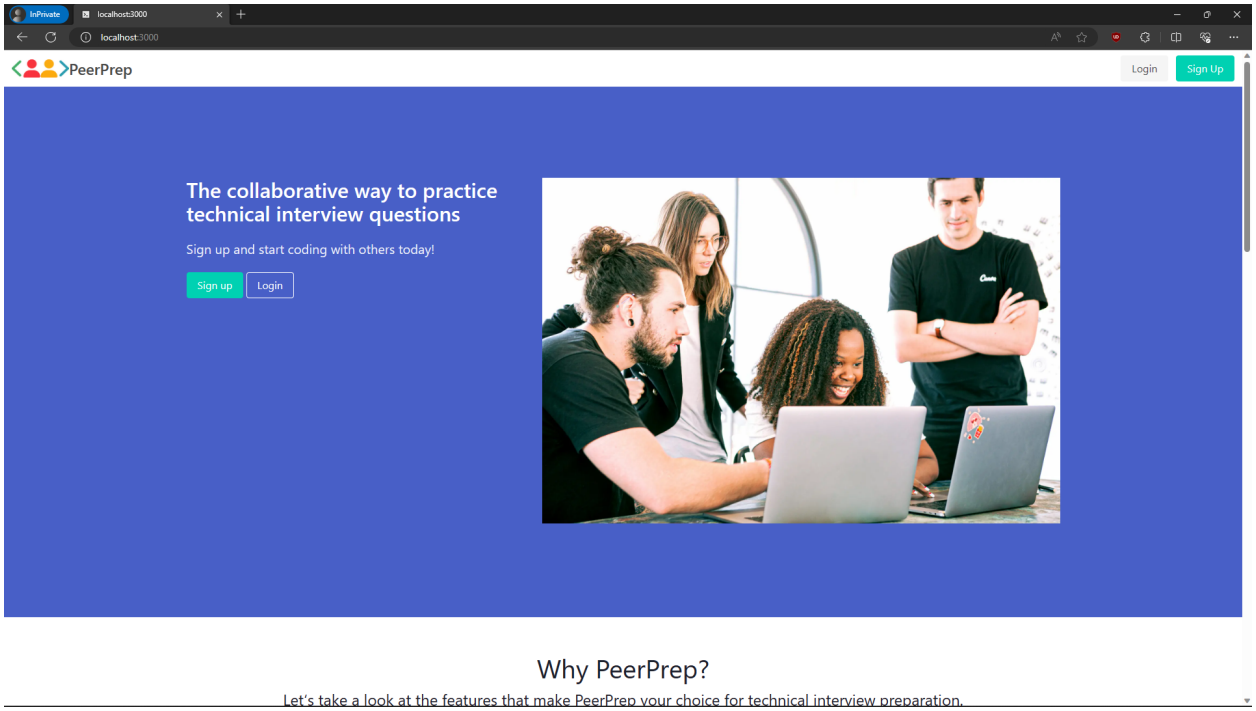
Despite having a product backlog, there were times when requirements of a feature were unclear. This led to confusion among the team when features implemented were not what they expected. In hindsight, this could be avoided by emphasizing on details when creating user stories in order to be clear on what the feature is supposed to achieve. These user stories could be further supported by diagrams or references to existing implementations so that everyone would be on the same page.

Overall, these challenges we encountered served as valuable lessons, highlighting improvements that we can make within ourselves and our workflow as a group. This experience would surely prove valuable in the future when we work in software development within a team.

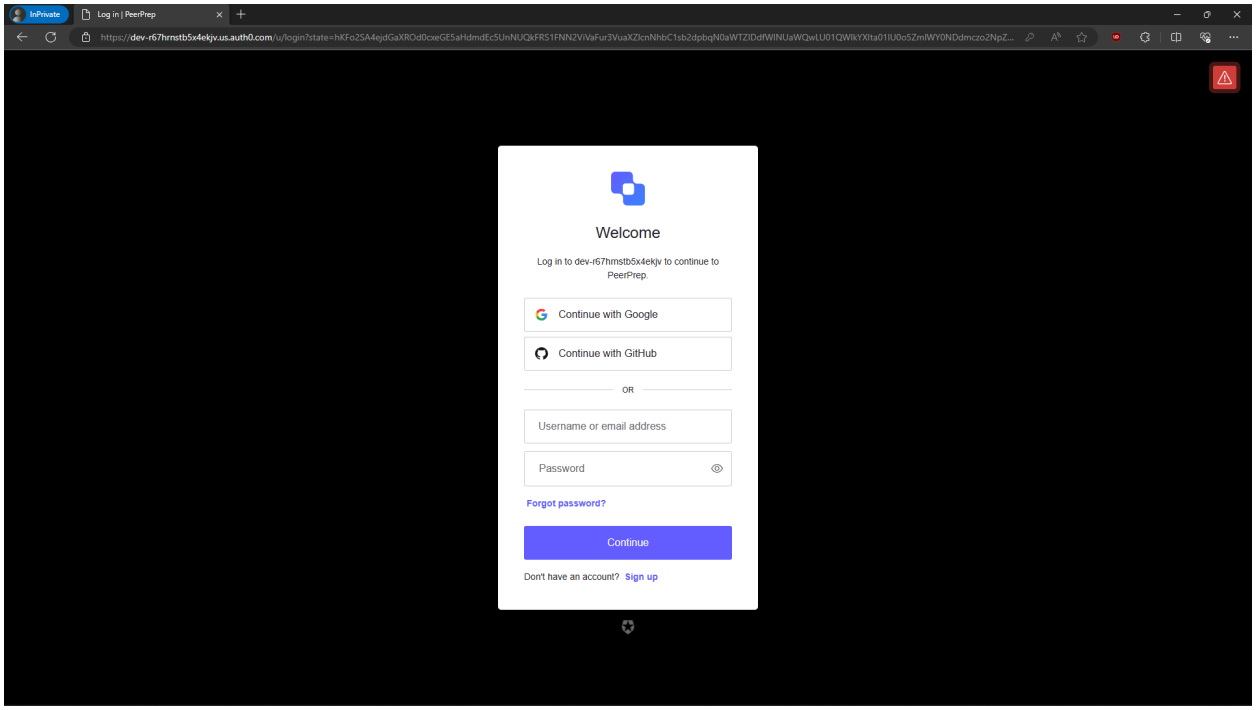
Appendix

Product Snapshots

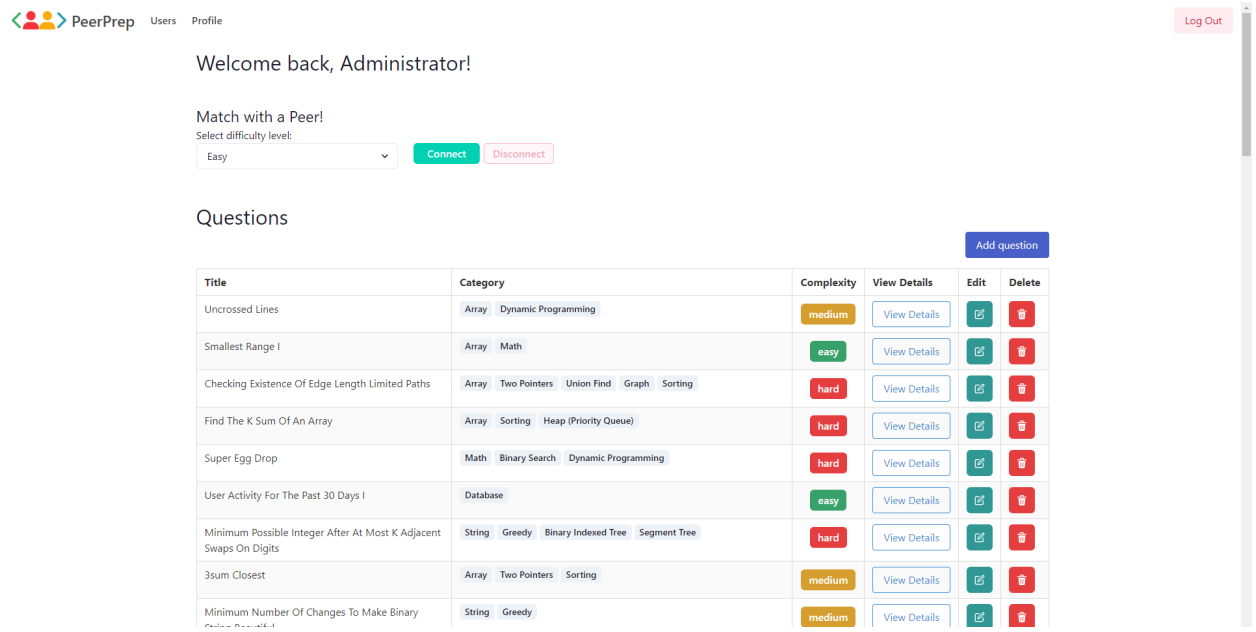
This section includes screenshots of PeerPrep.



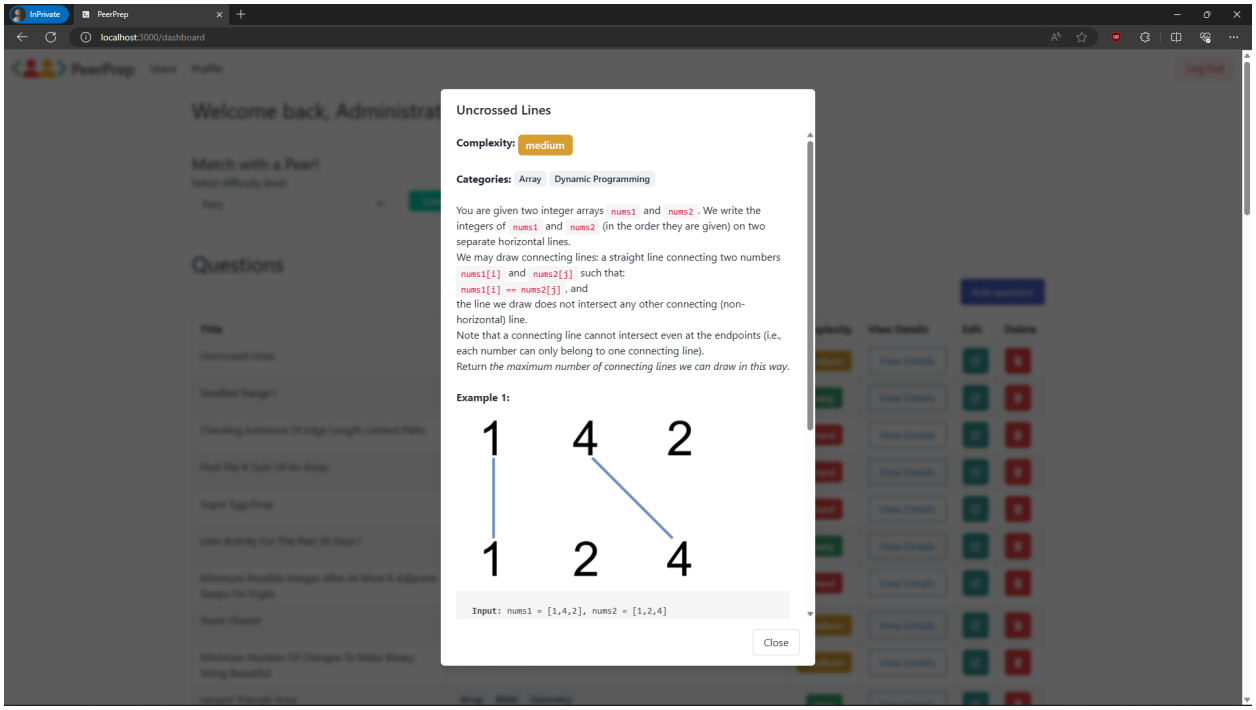
Landing Page (/) For guests and users not logged in



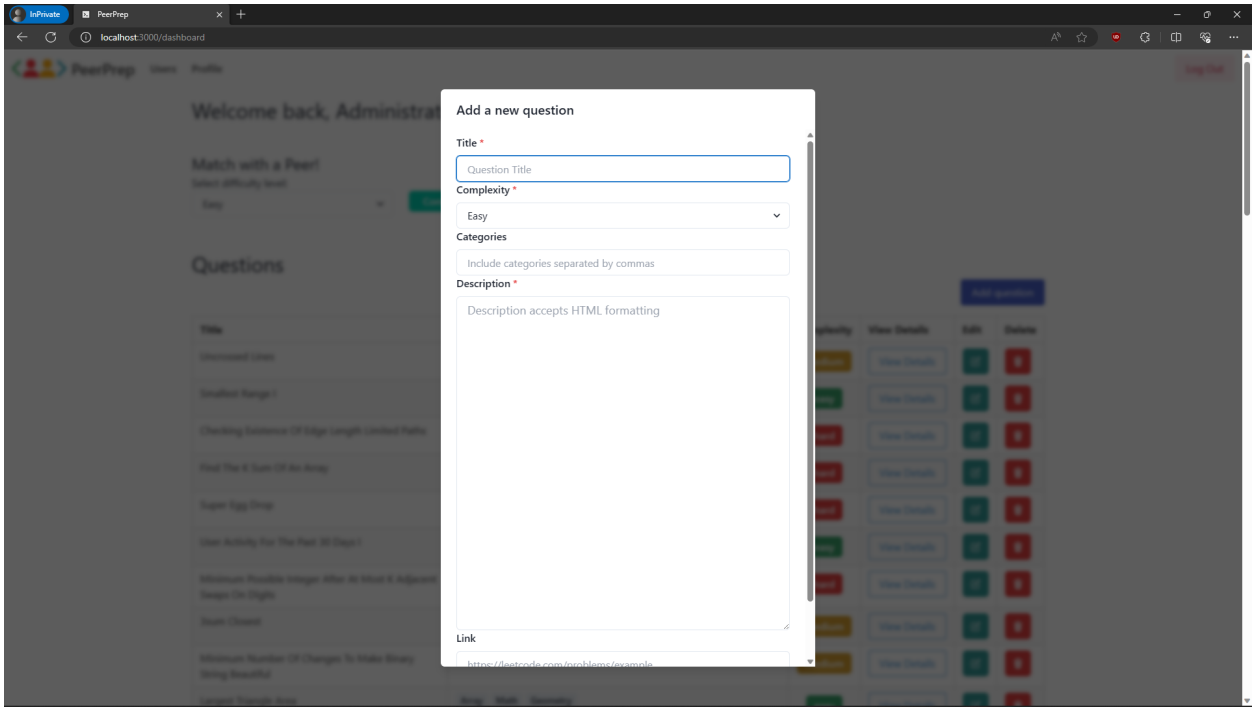
Login/Signup page



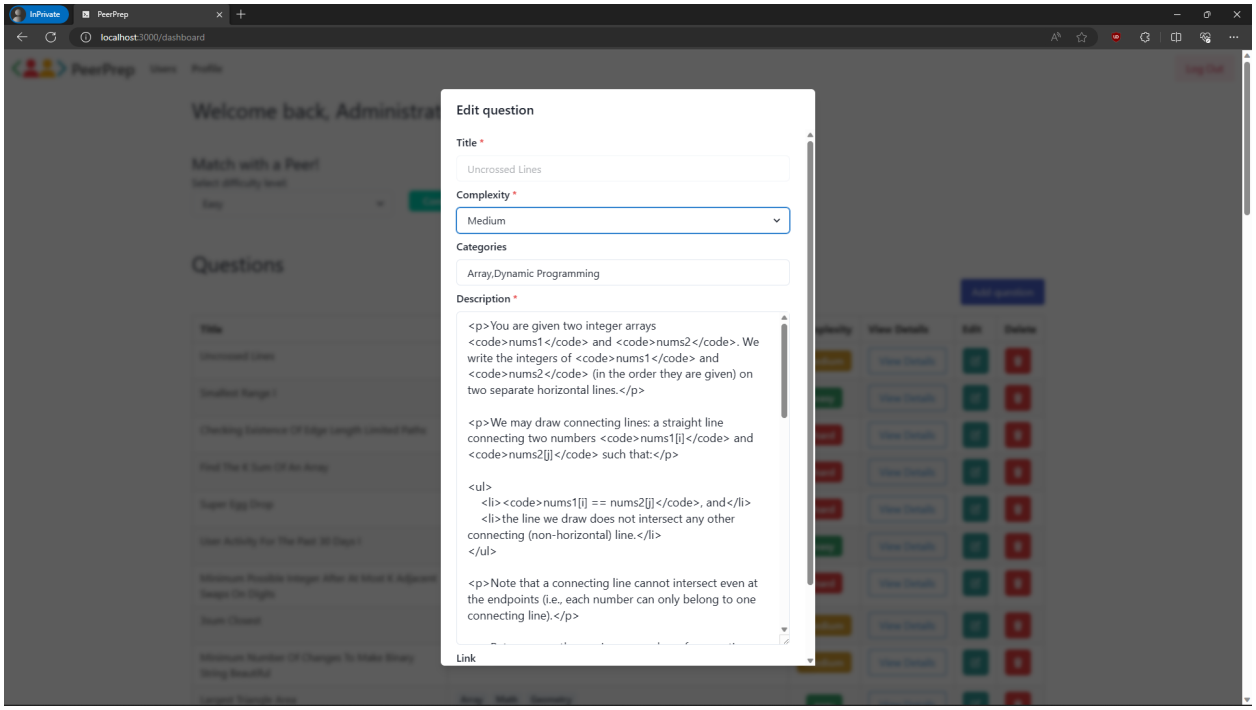
Dashboard page (/dashboard) for users that are logged in



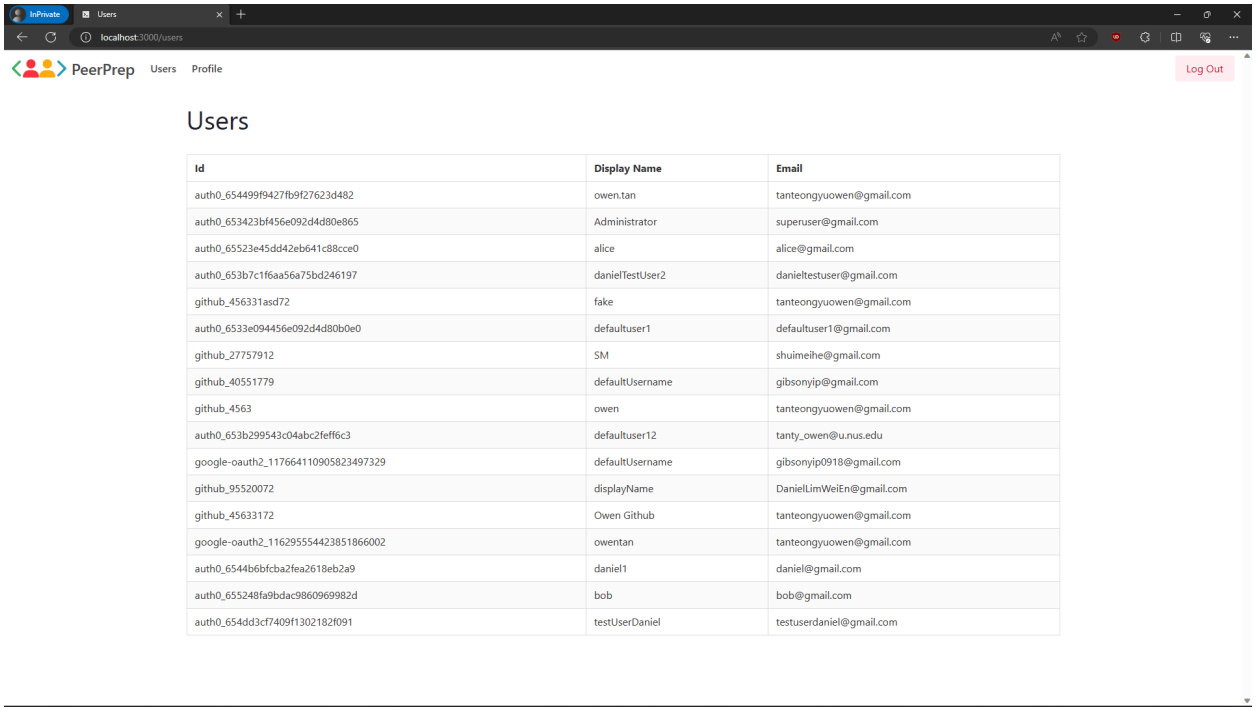
View question details



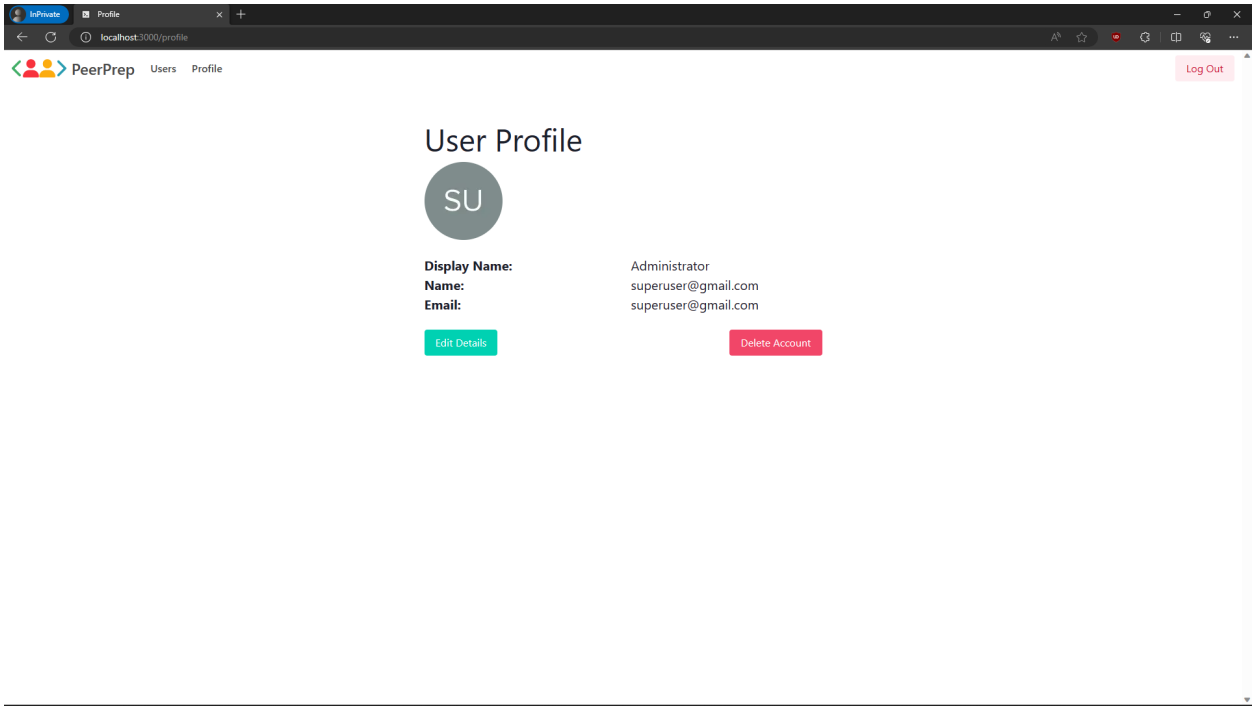
Add Question modal (only available to administrator accounts)



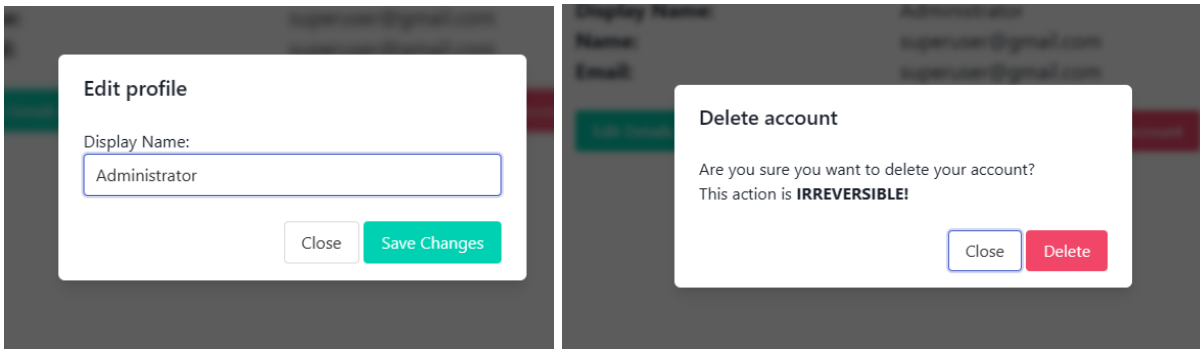
Edit Question modal (only available to administrator accounts)



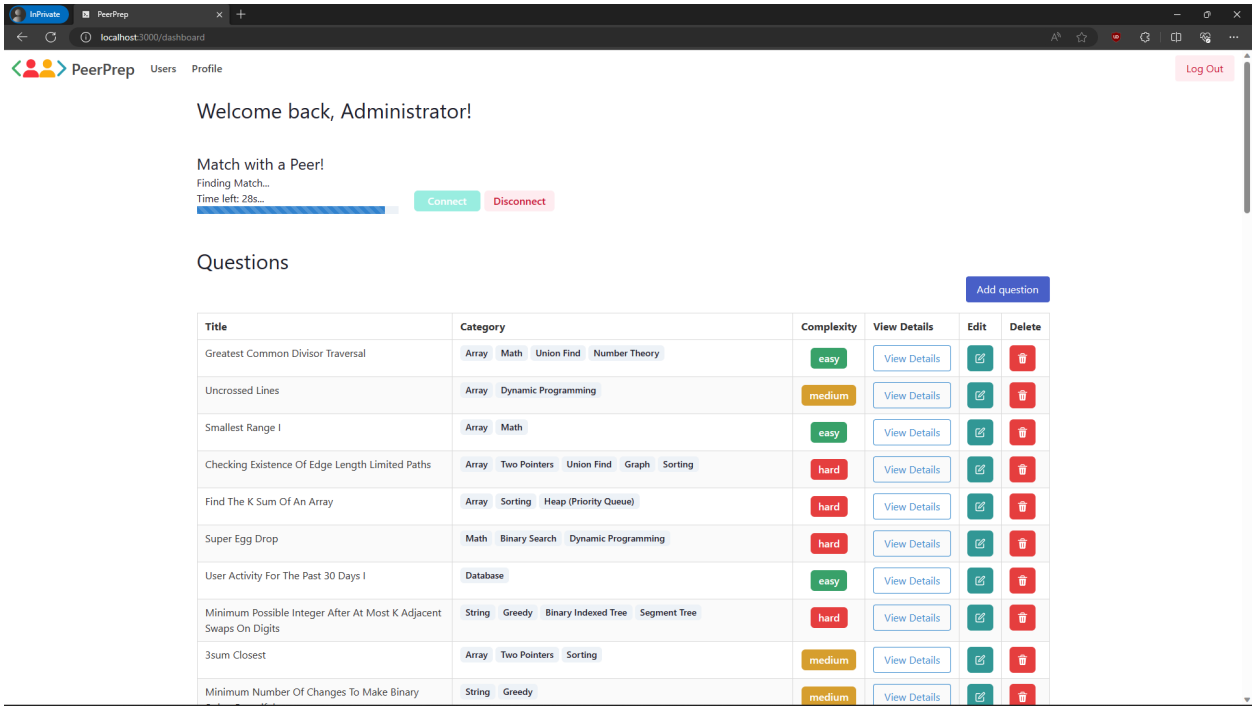
View all users page (/users) only available to administrator accounts



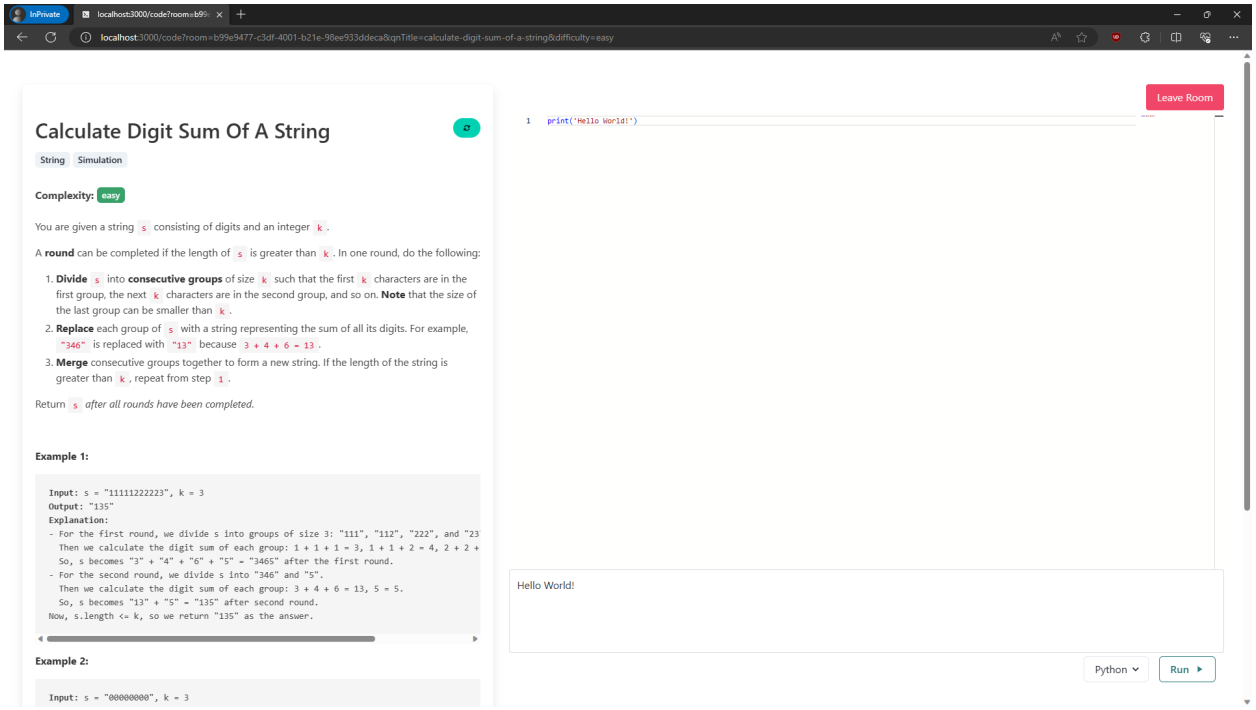
User profile page (/profile). Users can edit their display name in this page or delete their account



Edit display name modal and delete user modal in the profile page



User in queue



Collaborative code editor page (/code)

- 2 logged in users are matched according to the question difficulty chosen
- Users can choose the coding language, and execute the code
- Users can start a video call with each other

Sample APIs

Code execution

`POST /submissions/{?base64_encoded,wait}`

Status Code	Response

201	<pre>{
	  "stdout": "SEVMTE8hIQo=\n",
	  "time": "0.018",
	  "memory": 3112,
	  "stderr": **null**,
	  "token": "30868c2d-2f2c-4611-b31a-aa5f1709fc9d",
	  "compile_output": **null**,
	  "message": **null**,
	  "status": {
	  "id": 1,
	  "description": "Accepted"
	  }
	}</pre>

422	<pre>{
	  "language_id": [
	  "can't be blank"
	  ]
	}</pre>

422	<pre>{
	  "language_id": [
	  "language with id 150000 doesn't exist"
	  ]
	}</pre>

503	<pre>{
	  "error": "queue is full"
	}</pre>

`GET /submissions/{token}{?base64_encoded,fields}`

Status Code	Response

200	{
	"stdout": "HELLO!!\n",
	"time": "0.018",
	"memory": 3112,
	"stderr": **null**,
	"token": "30868c2d-2f2c-4611-b31a-aa5f1709fc9d",
	"compile_output": **null**,
	"message": **null**,
	"status": {
	"id": 1,
	"description": "Accepted"
	}
	}