

PeerPrep

Group 08

CS3219 Software Engineering Principles and Patterns

Name	Student number
Elgin Lee Yiheng	A0233556A
Ezra Tio Shu Rong	A0233377B
Goh Yi Jie, Jonathan	A0239659J
Gregg Chan	A0238929L
Quah Kian Seong	A0235337E

Github Repository:

<https://github.com/CS3219-AY2324S1/ay2324s1-course-assessment-q08>

Deployment:

<https://peer-prep-ywhzo.ondigitalocean.app/>

I. Table of Contents

I. Table of Contents.....	2
II. Project Contributions.....	7
III. Subgroups.....	9
IV. Introduction.....	10
Background.....	10
Purpose.....	10
1. Requirements Specification.....	11
1.1 User Service.....	11
1.1.1 Functional Requirements.....	11
1.1.2 Non-Functional Requirements.....	12
1.2 Matching Service.....	13
1.2.1 Functional Requirements.....	13
1.2.2 Non-Functional Requirements.....	14
1.3 Question Service.....	15
1.3.1 Functional Requirements.....	15
1.3.2 Non-Functional Requirements.....	16
1.4 Collaboration Service.....	17
1.4.1 Functional Requirements.....	17
1.4.2 Non-Functional Requirements.....	18
1.5 Code Execution Service.....	19
1.5.1 Functional Requirements.....	19
1.5.2 Non-Functional Requirements.....	20
1.6 Communication Service.....	21
1.6.1 Functional Requirements.....	21
1.6.2 Non-Functional Requirements.....	22
1.7 History Service.....	23
1.7.1 Functional Requirements.....	23
1.7.2 Non-Functional Requirements.....	24
1.8 AI Service.....	25
1.8.1 Functional Requirements.....	25
1.8.2 Non-Functional Requirements.....	26
2. Architectural Design.....	27
2.1 Architectural Diagram Description.....	28
2.2 Architecture Diagram Tool Choices.....	29
2.3 Alternative architectural diagram design.....	31

2.4 Architectural Design Thought Process.....	31
3. Database Schema Design.....	32
3.1 Question Service Database Design (MongoDB).....	32
3.1.1 Question Schema.....	32
3.1.2 Code Template Schema.....	34
3.1.3 Test Case Schema.....	34
3.1.4 Possible Alternative Schemas.....	35
3.2 Matching Service Database Design.....	37
3.2.1 WaitingList Schema.....	37
3.3 User Service Database Design.....	38
3.3.1 User Schema.....	38
3.4 Collaboration Database Design.....	39
3.4.1 Room Schema.....	39
3.5 History Database Design.....	40
3.5.1 AnsweredQuestion Schema.....	40
4. Service-Specific Design.....	41
4.1 AI Service.....	41
4.1.1 Front End Design.....	41
4.1.2 Key Features.....	42
4.1.3 Implementation.....	42
4.1.4 Back End Design (AI-Service microservice).....	42
4.1.5 Model Choice.....	42
4.2 Code Execution Service.....	43
4.2.1 Front End Design.....	43
4.2.2 Judge0 API Call.....	44
4.2.3 Design rationale.....	45
4.3 Collaboration Service.....	46
4.3.1 CodeMirror.....	46
4.3.2 WebSocket Design.....	46
4.3.3 Database Setup.....	47
4.3.4 Websocket Setup.....	47
4.3.5 Stateful Component Setup.....	47
4.3.6 Action Codes.....	47
4.4 Communication Service.....	53
4.4.1 Video Calling Feature.....	53
4.4.2 Chat Feature.....	55
4.5 History Service.....	57
4.5.1 Front End Design.....	57
4.5.2 Key Features.....	57

4.6 Matching Service.....	58
4.7 Question Service.....	59
4.8 User Service.....	59
5. Frontend.....	60
5.1 React.....	60
5.2 Architecture Diagram.....	60
5.2.1 Legend.....	61
5.2.2 Design Decision-Making Process.....	61
5.3 Key Libraries Utilized.....	62
6. UI.....	63
6.1 Login Page.....	63
6.2 Sign Up Page.....	64
6.3 Home Page.....	65
6.4 User Profile Page.....	66
6.5 Queuing State.....	68
6.6 Custom Room Portal.....	69
6.7 Programming Room.....	70
6.7.1 Room Manipulation.....	70
6.7.2 Communication Tools.....	71
6.7.3 Interactive Development Environment (IDE).....	73
6.7.4 General Features.....	73
7. UX.....	73
7.1 Login/Sign Up.....	74
7.2 User Profile Options.....	74
7.3 Matching.....	77
8. Authentication & Security.....	78
8.1 Process of authentication.....	78
8.2 Process of Registration.....	79
8.3 Bcrypt.....	80
8.4 JsonWebToken (JWT).....	80
8.5 Securing REST Endpoints.....	81
8.5.1 User Service.....	81
8.5.2 Question Service.....	82
8.5.3 Collaboration Service.....	82
8.5.4 Code Execution Service.....	82
8.6 Securing Websockets.....	83
8.7 Security of traffic through HTTPS.....	84
8.7.1 Frontend HTTPS.....	84
8.7.2 Backend HTTPS.....	85

9. Continuous Deployment.....	87
9.1 Kubernetes Cluster Automated Deployment.....	87
9.1.1 Snapshot of digital ocean container registry.....	89
9.1.2 Efficiency of continuous deployment.....	90
9.2 Frontend Automated Deployment.....	91
10. Code Organization.....	92
10.1 MVC architectural pattern.....	93
10.1.1 Model.....	93
10.1.2 View.....	93
10.1.3 Controller.....	94
11. ORM framework.....	95
11.1 Sequelize.....	95
11.2 Mongoose.....	95
12. Kubernetes Horizontal Pod Autoscaler.....	96
12.1 Load Testing.....	97
13. API Endpoints.....	98
13.1 Question Service.....	99
13.1.1 Fetching a question by ID.....	99
13.1.2 Fetch all questions.....	101
13.1.3 Add a question.....	102
13.1.4 Update a question.....	104
13.1.5 Delete a question.....	104
13.1.6 Get random question.....	106
13.2 User Service.....	108
13.2.1 Register.....	108
13.2.2 Login.....	109
13.2.3 Get username.....	110
13.2.4 Update password.....	111
13.2.5 Deregister.....	112
13.3 Collaboration Service.....	113
13.3.1 Create Room.....	113
13.3.2 Get Room.....	114
13.3.3 Update room question.....	115
13.4 Code Execution Service.....	116
13.4.1 Execute all test cases.....	116
13.5 AI Service.....	118
13.6 History Service.....	119
13.6.1 Add answered question.....	119
13.6.2 Get all answered question by username.....	119

14. Crucial API Latency Test.....	120
15. Software Development Process.....	124
15.1 Sprint System and Work Allocation.....	124
15.2 Pair Programming.....	125
15.3 Source control and collaboration.....	125
16. Improvements and Enhancements.....	126
16.1 Lack of cohesion for services.....	126
16.2 Scalability of Stateful Services.....	127
16.2.1 Problem.....	128
16.2.2 Possible Fix.....	128
16.3 JWT Token Handling.....	129
16.4 Logic handling for Add Question, Update Question.....	129
16.5 Reasoning behind using database for the Matchmaking Service.....	129
17. Reflections and learning points.....	130

II. Project Contributions

Member	Technical Contributions	Non-Technical Contributions
Elgin Lee Yiheng	<ul style="list-style-type: none"> Deployed frontend and kubernetes cluster Code execution for 3 languages Continuous deployment (CD) for frontend and K8s cluster configuration Change question inside of a room Adding new question with code template and test cases Code / language / question synchronization for code editor on leave / join room 	
Goh Yijie, Jonathan	<ul style="list-style-type: none"> Dockerized the application to make it usable in any environment Implemented the video calling Implemented the real-time chatting logic and interface Implemented the AI service chat bot 	Self-documented contributions
Ezra Tio Shu Rong	<ul style="list-style-type: none"> Implemented login and signup system UI/UX design and implementation for login, sign up, and custom room pages Implemented custom room functionality Implemented collaborative real-time IDE Implemented dynamic syntax highlighting for real-time IDE All assignment videos (Voiceover, compiling, polishing) Art for login page 	
Gregg Chan	<ul style="list-style-type: none"> Updating of question for user with manager roles Authentication of users in order to view question and interact with the application Addition of a random question whenever a room is created for user referencing Some UX features 	
Quah Kian Seong	<ul style="list-style-type: none"> Implemented history service with graphs and tables, making use of different libraries 	

	<ul style="list-style-type: none">• User's profile page• Some UX features	
--	--	--

Side note: Everyone has an equal share for the must-haves (i.e. assignments), and there are too many overlaps to split who has done what.

III. Subgroups

No.	Members	Nice to haves
1	Jonathan, Kian Seong	<ul style="list-style-type: none"> • N1: Communication: Implement a mechanism to facilitate communication among the participants in the collaborative space (other than the shared workspace) e.g., text-based chat service and/or video (+voice) calling service. • N2: History: 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. • N7: Incorporate generative AI to assist during the preparation. For example, users can seek help from ChatGPT, Copilot (or such other services) to seek explanation of code written by the other participant. Users can seek help from gen AI to solve the puzzle by providing prompts from within the interface.
2	Elgin, Ezra, Gregg	<ul style="list-style-type: none"> • N3. Code execution: Implement a mechanism to execute attempted solution/code in a sandboxed environment, and retrieve + present the results in the collaborative workspace • N4. Enhance question service by allowing questions to be managed via complexity, and changing questions during a session. • N5. Enhance collaborative service by providing an improved code editor with code formatting, syntax highlighting for one language, syntax highlighting for multiple languages • N9. Deployment of the application on the production system (Digital Ocean Platform) • N10. Scalability: The deployed application should demonstrate easy scalability of some form • N11: The application should have an API gateway of some kind that redirects requests to the relevant microservices. • N13: Custom room that allows users to code with their friends instead of a stranger. • N14: Continuous deployment for frontend and backend

IV. Introduction

Background

The technical interview is an initial hurdle that most people aspiring to be a part of the technology industry have to go through. Despite the existence of coding platforms such as Leetcode to practice data structure and algorithm questions, there is a lack of collaboration features. This hinders the users from being able to practice conveying their ideas verbally to someone else, which is vital to pass a technical interview.

Purpose

We aim to create a platform that fills the gap of existing coding platforms, by providing collaborative and matchmaking features. By utilizing the platform, people will become more prepared and confident to tackle technical interviews.

1. Requirements Specification

1.1 User Service

The user service refers to the authentication features that we provide for login and signup. It ensures that only authenticated users have access to the site.

1.1.1 Functional Requirements

Responsible for user profile management, persisting data in an SQL database.

No.	Requirement	Priority	Sprint
F-U-1	The user service should allow users to register for an account with a username and password	High	1
F-U-2	The user service should provide an interface for users to login with username and password	High	1
F-U-3	The user service should allow only users who input the correct username and password to log in successfully	High	1
F-U-4	The user service should allow logged in users to change their password	High	1
F-U-5	The user service should allow logged-in users to deregister their own personal account	High	1
F-U-6	The user service should allow logged-in users to log-out	High	1
F-U-7	The user service should be able to assign roles to users (normal user / manager)	High	1
F-U-8	The user service should provide an interface for logged in user to retrieve their profile information (username)	High	1
F-U-9	The user service should provide a way for users to remain logged in on page refreshes	High	1

1.1.2 Non-Functional Requirements

Prioritized NFRs for user service:

Priority Rank	Quality Attribute	Rationale
1	Security	This service plays a vital role in protecting the whole application. If not prioritized, the whole application's security will be compromised
2	Availability	This service should not have any down time at all. If this service becomes unavailable, users will be unable to be authenticated and use other application features
3	Usability	Since this service will be the first that users will interact with, it should be user friendly so that it will not deter new users away.

No.	Requirement	Priority	Sprint
NF-U-1	Passwords saved in the SQL database should be hashed to protect against malicious actors	High	1
NF-U-2	On registration, passwords must have a minimum length of 8 to protect against brute-force attacks	High	1
NF-U-3	Service should have 100% uptime, and no downtime, with the help of having replicas of this service's pods.	High	1
NF-U-3	There should be user friendly messages on login / registration attempts so that users will know what they have done wrongly and amend it	High	1
NF-U-4	Duplicate username should not be allowed because it may compromise a person's identity	High	1

1.2 Matching Service

The bread and butter of PeerPrep. Users are able to “match” with each other via a queueing system based on selectable difficulty. This allows users to perform mock interviews without needing to find someone to interview with offline.

1.2.1 Functional Requirements

Responsible for matching users based on the complexity of questions the user chooses to do.

No.	Requirement	Priority	Sprint
F-M-1	This service should allow users to join a queue based on question complexity	High	2
F-M-2	This service should match two users if there is a valid match (2 person queuing for the same complexity)	High	2
F-M-3	This service should remove users who are in the queue for more than 30 seconds, but have not found a valid match	High	2
F-M-4	On failing to find a match, users should be able to retry matching or cancel the matching attempt	High	2
F-M-5	Users should be notified with the progress of the matching via a timer countdown	High	2

1.2.2 Non-Functional Requirements

Prioritized NFRs for matching-service:

Priority Rank	Quality Attribute	Rationale
1	Reliability	This service plays a vital role in matching users so that they can collaborate on solving a coding question. They must be able to match with another person whenever there is a valid match, and should match up with the person queuing for the same complexity.
2	Robustness	This service should handle failed matches gracefully, and not crash the service, because bringing down the service will prevent matches for a period of time
3	Testability	This service should be designed such that there is a way to test that there is always a correct matching

No.	Requirement	Priority	Sprint
NF-M-1	The matching service should have 100% accuracy in matching users who are a valid match	High	2
NF-M-2	The matching service should not have any false matches, and should only match users who are queuing for the same difficulty	High	2
NF-M-3	The matching service should handle fail matches gracefully, and not crash the service	Medium	2
NF-M-4	The matching service should provide interfaces for testers to simulate 2 different users trying to join a queue	Medium	2

1.3 Question Service

The question service houses the question repository. Each question comes with a full description, test cases and code boilerplate in three supported languages.

1.3.1 Functional Requirements

Responsible for maintaining a question repository persisted in a MongoDB database

In this section, **attributes** refer to the following schema:

(Question Id, Question Title, Question Description, Question Category, Question Complexity, Code Templates, Test Cases)

No.	Requirement	Priority	Sprint
F-Q-1	The question service should support adding new questions with the appropriate attributes	High	1
F-Q-2	The question service should support fetching of all questions stored in the database	High	1
F-Q-3	The question service should support the deleting of specified questions from the database	High	1
F-Q-4	The question service should support updating of questions	High	1
F-Q-4	All questions must have a question ID, question description, question title, question category, question complexity, code templates, and test cases	High	1
F-Q-5	A list of questions must be displayed with basic information (Question Title, Question category, Question Complexity) to the users	High	1
F-Q-6	A question when clicked, should display details on the question (All attributes except for Question Id)	High	1
F-Q-7	Adding of duplicate questions is disallowed, the question title and question description must be unique	High	1
F-Q-8	Unauthenticated and unauthorized users is not allowed to access questions either for reading or updating	High	2
F-Q-9	Logged in users without a manager role is not allowed to update or delete any questions from the question repository, but only view the questions	High	2

F-Q-10	Logged in users with the manager role can read, update, and delete the questions	High	2
F-Q-11	A question should be able to support images	High	1

1.3.2 Non-Functional Requirements

Prioritized NFRs for question service:

Priority Rank	Quality Attribute	Rationale
1	Performance	Questions are an integral part of the application, and failure to fetch and display questions to the user will result in extremely poor user experience
2	Scalability	This service should be scalable because it is the service that will have the highest traffic. It needs to be scalable to prevent overloading, and still serve users with low latency
3	Efficiency	There are a lot of attributes within a question, and not all are required for certain parts of the application. There should be some form of optimization such as selectively returning attributes of a question required to reduce data transferred within the traffic

No.	Requirement	Priority	Sprint
NF-Q-1	Adding a question should take less than 0.5 seconds	High	1
NF-Q-2	Deleting a question should take less than 0.5 seconds	High	1
NF-Q-3	Updating a question should take less than 0.5 seconds	High	1
NF-Q-4	Getting all questions should take less than 0.5 seconds	High	1
NF-Q-5	This service should be scalable with the help of tools such as Kubernetes horizontal pod auto scaler (HPA)	High	1

1.4 Collaboration Service

The collaboration service simulates an environment similar to that of a real live programming interview. This is in line with the main design goals of PeerPrep.

1.4.1 Functional Requirements

In this section, we refer pair as the two users who are matched and is in a shared room

No.	Requirement	Priority	Sprint
F-C-1	The pair should have a shared code editor that shows live updates on changes	High	3
F-C-2	This service should allow more than 1 room, so that when there is more than 1 pair, all pairs can have their own private room	High	3
F-C-3	The collaboration service should restrict only at most two users in a room at any given time	High	3
F-C-4	The collaboration service should provide essential question attributes for the pair to reference in the room	High	3
F-C-5	The system should provide a code editor with syntax highlighting	High	3
F-C-6	The collaboration service should provide multiple languages that the pair can choose to code in, with the correct syntax highlighting	High	3
F-C-7	This service should allow the pair to change to another random question, provided that both of them agree	High	3
F-C-8	Users should be able to leave a room whenever they want, and the person remaining in the room can stay in it	High	3
F-C-9	Users should be able to join a room with the room's ID	High	3
F-C-10	The room's state should be deleted when the pair leaves the room	High	3

1.4.2 Non-Functional Requirements

Prioritized NFRs for collaboration service:

Priority Rank	Quality Attribute	Rationale
1	Availability	This is a core service for the application, and it should not have any downtime. If not, it will render the application useless
2	Usability	The interface should be user friendly since this is the service that most users will interact with the most. A bad user experience will prevent users from using the application
3	Performance	There should be low latency, and code changes by one user should be immediately reflected by the other. If not, they might not be on the same code editor state and lead to poor user experience

No.	Requirement	Priority	Sprint
NF-C-1	The service should have low latency, ensuring that code changes made by one user are quickly reflected to the other, within 0.1 seconds	High	3
NF-C-2	The service should be automatically restarted whenever there is a failure, to give high availability	High	3
NF-C-3	The service should provide useful widgets that are useful for users, such as being able to see who is in the room.	High	3
NF-C-4	The service should provide a good flow of collaboration, from the point of entry until the end, giving good feedback to the user on what they should do	Medium	3
NF-C-4	The collaboration service should be able to support at least 1000 rooms at once.	High	3

1.5 Code Execution Service

Allows users to execute their written code by sending the code to Judge0, an API for code execution. This allows users to verify their solutions.

1.5.1 Functional Requirements

No.	Requirement	Priority	Sprint
F-CE-1	Users should be able to run their code and be able to know how many test cases have passed / failed	High	4
F-CE-2	Users should be able to view the output of their code execution based on the input and the expected output for the test case	High	4
F-CE-3	Users should be able to rerun the code as many times as they want	High	4
F-CE-4	Users should receive helpful messages when their code fails to run, telling them why it has a runtime error	High	4
F-CE-5	Users should receive TLE if their code takes too long to run	High	4
F-CE-6	This service should support code execution for multiple languages	High	4

1.5.2 Non-Functional Requirements

Prioritized NFRs for code execution service:

Priority Rank	Quality Attribute	Rationale
1	Performance	We should expect that the code execution results to return as soon as possible, because a long latency will lead to poor user experience
2	Reliability	We should not expect a code execution to return back a wrong result, if not it will lead to confusion to the user on which of the test cases actually passed, and which has failed.
3	Usability	Providing code execution will be useless if it doesn't provide useful feedback to users, giving them reasons for errors, their code output, input and the expected output
4	Scalability	Scaling this service will provide a capability to support more requests per second by users, as well as reducing the downtime. Hence, improving user experience

No.	Requirement	Priority	Sprint
NF-CE-1	There should not be any failed test case that should have passed	High	4
NF-CE-2	There should not be any test cases that passed but should have failed	High	4
NF-CE-3	The code execution and result returning should not take more than 5 seconds	High	4
NF-CE-4	There should be styling to denote a passed test case and failed test case, and ability to see the test case's input, expected output, and code execution output	High	4
NF-CE-5	This service should be scalable	High	4

1.6 Communication Service

Allows users to communicate while in the same room. This facilitates better discussion and collaboration.

1.6.1 Functional Requirements

No.	Requirement	Priority	Sprint
F-CO MM-1	Two matched users should have a chat-box that allows real time communication	High	1
F-CO MM-2	Users should be able to communicate via video call.	High	2
F-CO MM-3	Users should be able to communicate via voice call.	High	2

1.6.2 Non-Functional Requirements

Prioritized NFRs for communication service:

Priority Rank	Quality Attribute	Rationale
1	Performance	Ensure minimal latency in message delivery and call setup.
2	Reliability	Ensure high availability and error-free operation during calls and chats.
3	Usability	User-friendly interface, clear error messages, and intuitive navigation.
4	Security	Secure data transmission and storage, with privacy controls for users.

No.	Requirement	Priority	Sprint
NF-CO MM-1	Users should be able to see their message history throughout the entire match.	High	2
NF-CO MM-2	Users should be able to turn their video camera off.	High	2
NF-CO MM-3	There should not be any conflict of messages and audio sent at the same time	High	3
NF-CO MM-4	Users should only be able to use the video calling features under Https	High	3

1.7 History Service

Allows users to track their past interviews and solves. This provides a more immersive experience and retains users in the long-term.

1.7.1 Functional Requirements

Tracks the questions that are completed by the user

No.	Requirement	Priority	Sprint
F-H-1	History service should be able to return user's completed questions	High	3
F-H-2	History service should have an endpoint for adding completed question to an SQL/ NoSQL database	High	3
F-H-3	History service should show the question completed, question complexity, date, and who he is paired with	High	3
F-H-4	The history addition endpoint should be protected. Only authenticated users will have their history logged	High	3
F-H-5	The history tracking endpoint should be protected. Logged in users can view their own history, but not others' history	High	3

1.7.2 Non-Functional Requirements

Prioritized NFRs for history service:

Priority Rank	Quality Attribute	Rationale
1	Performance	We should expect that the history service results to return as soon as possible, because a long latency will lead to poor user experience
2	Usability	The interface should be user friendly since this is the service that most users will interact with the most. A bad user experience will prevent users from using the application
3	Reliability	The user should be able to access their history throughout the lifetime of the account. If the history gets erased at any point, this defeats the purpose of a history service

No.	Requirement	Priority	Sprint
NF-H-1	Loading the history should not take longer than 1 second	High	3
NF-H-2	Adding a history entry should not take more than 1 second	High	3
NF-H-3	History should be persisted throughout the lifetime of the user account	High	3

1.8 AI Service

A novelty that allows users to quickly access OpenAI's ChatGPT-4 to clarify any doubts.

1.8.1 Functional Requirements

Allows the user to interact with an interactive AI helpbot during the coding session.

No.	Requirement	Priority	Sprint
F-AI-1	The application should have a user-friendly interface that allows users to input queries and receive responses from ChatGPT.	High	2
F-AI-2	Generate accurate, relevant, and context-appropriate responses to user queries.	High	2
F-AI-3	Capability to understand and process user input in natural language.	High	2

1.8.2 Non-Functional Requirements

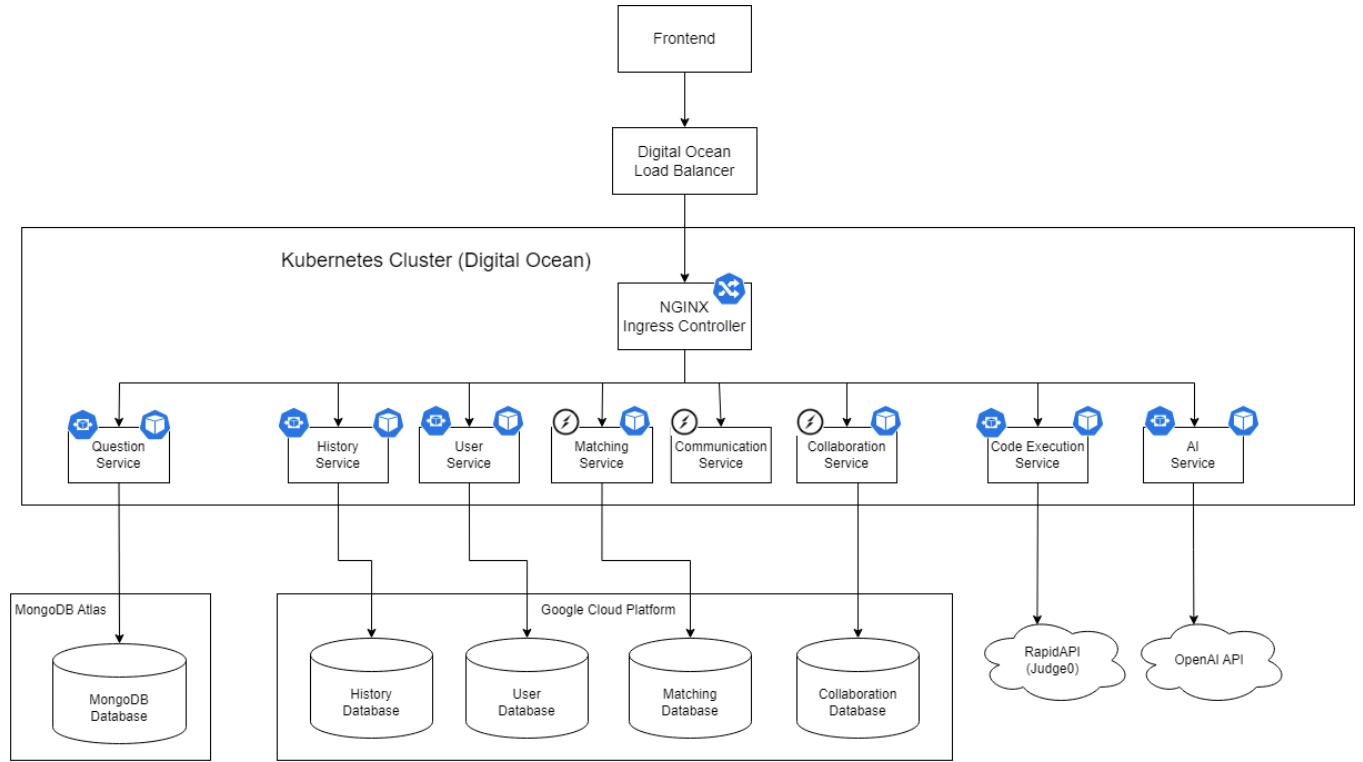
Prioritized NFRs for AI service:

Priority Rank	Quality Attribute	Rationale
1	Reliability	High availability and consistent performance, with minimal downtime.
2	Responsiveness	Quick response times to user queries, ensuring a fluid interaction.
3	Usability	The interface should be easy to understand and use

No.	Requirement	Priority	Sprint
NF-AI-1	More than one user should be able to use the AI service at any time	High	2
NF-AI-2	The chatbot should return a response to a query under 1 minute	High	3
NF-AI-3	The interface should be Intuitive and interactive	High	3

2. Architectural Design

PeerPrep Microservice Design



Architectural Diagram of Peerprep

Icons Legend:

Icon	Definition
	Kubernetes Pod
	Kubernetes Horizontal Pod Autoscaler Any service that has this symbol denotes that it has horizontal pod autoscaler enabled. Whenever the metrics server detects that 50% of the pod's CPU is used, it will scale horizontally as long as it does not exceed the maximum number that is specified.
	Websocket. Any service with this symbol denotes that websocket is utilized
	Kubernetes Ingress

2.1 Architectural Diagram Description

The typical workflow begins with a user initiating a request from the frontend. This request is first directed to the Kubernetes load balancer, which acts as the entry point to the Kubernetes cluster. From there, the request is forwarded to the NGINX Ingress Controller running within the Kubernetes Cluster.

Within the NGINX Ingress controller, the routing logic is determined by the rules defined in the ingress.yaml file. This configuration is used to analyze the user's request URL, specifically looking at its prefix. Based on this analysis, the Ingress Controller then directs the request to the appropriate service associated with the matching prefix. The service then sends a response back to the user.

2.2 Architecture Diagram Tool Choices

Kubernetes Choice

We hosted our Kubernetes cluster on DigitalOcean, and interacted with it through kubectl commands and configured it with the help of deployment, service, and ingress yaml files. The reason why we decided to make use of Kubernetes is because there are some services where scalability is an important NFR, and Kubernetes is our first choice because of the robustness and popularity of the tool. Furthermore, we can harness the functionalities provided such as horizontal pod autoscaler, and NGINX ingress controller that gives us the logic to route requests to the correct services.

MongoDB Choice

We wanted to use a NoSQL database for our Question Service because of the unpredictability of its test cases, and code boilerplates. Some form of flexibility to store our question data is required and a NoSQL database provides just that, and not an SQL database. Since MongoDB is the most popular NoSQL database, we decided to go for it.

SQL Database Choice

We hosted our SQL databases in Google Cloud Platform, and utilized it for user service, matching service, and collaboration service. Unlike questions, the data that these services store is predictable and does not require flexibility. With the ACID properties of an SQL database, and structured property of SQL, we decided to use it to ensure we get reliable and correct results.

WebSockets

For some services, we require bidirectional communication between the frontend and the service. Using pure REST API will not work because it is stateless, and after the request is received and response is sent, the server no longer remembers the client. Hence, we make use of one of the most popular tools, Socket.io that provides bidirectional communication.

RapidAPI (Judge0)

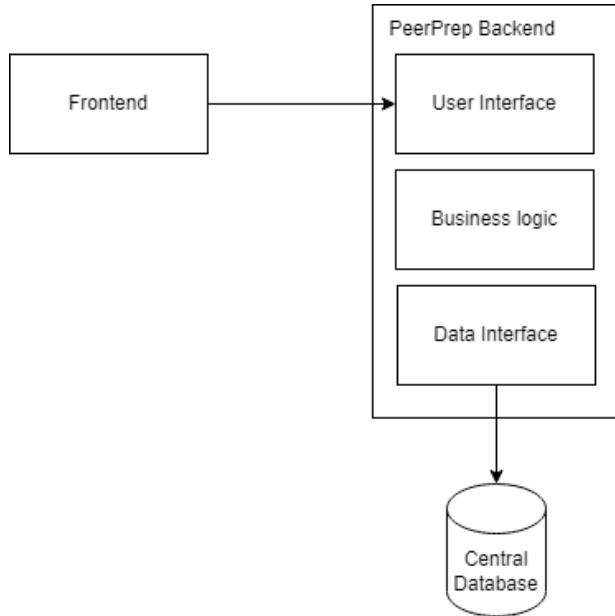
It is one of the more popular code execution tools that is very cheap (\$7/month), and used by many coding competition platforms. It provides high availability and execution of a myriad of programming languages as well and thus, we decided to employ it in our application.

Database per service pattern

It helps us by providing isolation, since each microservice has its own database. It protects the integrity of data by reducing the risk of data corruption and interference from other services. We can also scale individual services independently, without fearing that scaling a service will starve out other services in terms of database access. It also provides fault isolation, so if one database fails, it minimally impacts the rest of the application.

2.3 Alternative architectural diagram design

Monolithic architecture (Not used by us)



In this alternative design, it involves a monolithic architecture for the PeerPrep application. All requests from the frontend are sent to the user interface, which communicates with the business logic that handles the request. All logic is kept within the same service.

2.4 Architectural Design Thought Process

We decided to use the microservice architecture over the monolithic architecture for a number of reasons. Firstly, the microservice architecture allows us to scale the service that we want easily compared to the monolithic architecture. Since we have some NFRs that have the scalability requirement, the microservice architecture is better suited for our use case. Secondly, since we are working in a team of 5, a microservice makes maintenance and collaboration simpler because each subgroup will be responsible for different services, reducing code conflicts compared to monolithic architecture. Thirdly, it provides technology stack diversity, and different subgroups can choose different technology that they are better in to implement the service. Hence, it will lead to an overall higher quality application.

3. Database Schema Design

3.1 Question Service Database Design (MongoDB)

In our NoSQL database, we created 3 schemas to handle the storage of questions.

To skip through the schema description that we have defined, click [here](#) to navigate to the rationale of this design.

3.1.1 Question Schema

Code Snippet for Question Schema

```
5  const questionSchema = new Schema({
6    title: {
7      type: String,
8      required: true
9    },
10   categories: [
11     {
12       type: String,
13       required: true
14     }
15   ],
16   complexity: {
17     type: String,
18     enum: ['Easy', 'Medium', 'Hard'],
19     default: 'Easy',
20     required: true
21   },
22   description: {
23     type: String,
24     required: true
25   },
26   codeTemplate: {
27     type: Schema.Types.ObjectId,
28     ref: 'CodeTemplate'
29   },
30   testCases: [
31     {
32       type: Schema.Types.ObjectId,
33       ref: 'TestCase'
34     }
35   ]
36 });


```

Question schema description

Attributes	Properties and rationale
title	Type is of string which makes sense because it is a title
categories	An array of strings, because a question may have multiple categories, e.g. Array, Trees, Dynamic Programming
complexity	Type is a string which is constrained to 'Easy', 'Medium' and 'Hard'. There are no other complexity we want and this ensures that the database does not populate an erroneous value
description	Type is string
codeTemplate	A reference to CodeTemplate Schema
testCases	An array of references to TestCase Schema

Question document from MongoDB

```
_id: ObjectId('6545edd74ba1f46197f41f89')
title: "Longest Valid Parentheses"
- categories: Array
  0: "Strings"
  1: "Linked Lists"
complexity: "Hard"
description: "<p style='text-align:start;'><span style='color: rgb(255,255,255);background-color: black; font-weight: bold; font-size: 1.5em; padding: 0 2px;'>Longest Valid Parentheses</span></p>
<p>Given a string containing just two types of characters: '(', ')' and '*', write a function to check if the input string is valid. The '*' character here can represent either a '(' or ')'. A valid string must satisfy these constraints:</p>
<ul style='list-style-type: none; padding-left: 0; margin: 0; margin-top: 10px;'>
- Open brackets must be closed by the same type of brackets.
- An open bracket must be closed in the correct order.
- An asterisk can represent either a left parenthesis '(' or a right parenthesis ')'.</ul>
<p>Example 1:<br/>Input: s = "(*)<br/>Output: true<br/>Explanation: The single character '*' can be interpreted as either an opening or closing parenthesis.<br/>Example 2:<br/>Input: s = "(*))<br/>Output: false<br/>Explanation: This is not a valid expression because the second open parenthesis does not have a corresponding close parenthesis.<br/>Example 3:<br/>Input: s = ")<br/>Output: false<br/>Explanation: This is not a valid expression because there is an open parenthesis without a corresponding close parenthesis.<br/>Example 4:<br/>Input: s = "((*)*)<br/>Output: true<br/>Explanation: The expression is valid because the asterisks can be interpreted as opening parentheses, and the final ')' is a closing parenthesis that matches the first opening parenthesis. The intermediate ')' is also a valid interpretation of an asterisk as a closing parenthesis.<br/>Example 5:<br/>Input: s = "((*)*<br/>Output: false<br/>Explanation: This is not a valid expression because the final ')' is a closing parenthesis that does not have a corresponding opening parenthesis."'
codeTemplate: ObjectId('6545edd74ba1f46197f41f83')
- testCases: Array
  0: ObjectId('6545edd74ba1f46197f41f85')
  1: ObjectId('6545edd74ba1f46197f41f86')
__v: 0

```

3.1.2 Code Template Schema

Code snippet for Code Template Schema

```
5  const codeTemplateSchema = new Schema({
6    templates: {
7      type: Object,
8      required: true
9    }
10});
```

The code template schema consists of a template attribute that is of type object. This enables key value pairs of **language : code boiler plate** to be stored.

Code Template document from MongoDB

```
_id: ObjectId('65316ce0a2cf36e8d34e65e4')
templates: Object
  Python: "def search(nums, target):
            # Code here"
  Javascript: "function search(nums, target) {
                # Code here
              }"
  Ruby: "def search(nums, target)"
__v: 0
```

3.1.3 Test Case Schema

Code snippet for Test Case Schema

```
5  const testCaseSchema = new Schema({
6    input: {
7      type: Object,
8      required: true
9    },
10   output: {
11     type: Schema.Types.Mixed,
12     required: true
13   }
14});
```

The test case schema consists of an input which is of type object, and an output which is of type Mixed because it can be an integer, float, string, etc. This enables flexibility in the type of the output for our test case.

Test case document from MongoDB

```
_id: ObjectId('652785810654ccb905f3464f')
{
  input: Object {
    nums: Array [
      0: 2
      1: 7
      2: 9
      3: 3
      4: 1
    ]
    output: 12
  }
  __v: 0
}
```

3.1.4 Possible Alternative Schemas

There is an alternative design that requires just one schema to be defined, which contains all of the attributes stored in the Questions schema. However, this leads to a serious issue of increasing data flow across the network traffic between the frontend and the backend unnecessarily.

For example, in our home page, our application will be fetching all questions and displaying the question details to the user. However, it does not require the test cases or code templates to populate this page. Hence, if we were to implement the one schema method, we will be fetching a lot of unnecessary data.

The screenshot shows a dark-themed web application interface. At the top, there is a button labeled "Match with another user". Below it are three buttons: "Queue Easy" (green), "Queue Medium" (orange), and "Queue Hard" (red). In the center, there is a search bar with the placeholder "Search by title or category". Below the search bar, there are filter buttons: "All", "Easy", "Medium", and "Hard". To the right of these filters are "Add" and "Delete" buttons. The main content area displays a table with the following data:

Title	Category	Complexity
House Robber	Dynamic Programming,	Easy
3Sum	Arrays, Depth-First Search (DFS),	Medium
Two Sum	Arrays,	Easy
Longest Palindromic Substring	Strings,	Hard
Substring with concatenation of all words	Trees, Dynamic Programming,	Hard
Container with most water	Backtracking,	Medium
Search in rotated sorted array	Strings, Arrays,	Easy
Divisible and Non-divisible Sums Difference	Math,	Easy
ZigZag Conversion	Strings,	Medium
Longest Valid Parentheses	Strings, Linked Lists,	Hard

We should only populate the question's test cases and code templates only when we need them, like for example, the room page, which will save network bandwidth.

E ElginLeeChange QuestionCopy Room IDLeave Room

Container with most water

Medium

You are given an integer array `height` of length n . There are n vertical lines drawn such that the two endpoints of the i^{th} line are $(i, 0)$ and $(i, \text{height}[i])$.

Find two lines together with the x-axis form a container, such that the container contains the most water.

Return the maximum amount of water a container can store.

Notice that you may not slant the container.

Example 1:

Input: `height = [1,8,6,2,5,4,8,3,7]`
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example 2:

Input: `height = [1,1]`
Output: 1

Constraints:

- $n == \text{height.length}$
- $2 \leq n \leq 10^5$
- $0 \leq \text{height}[i] \leq 10^4$

Testcase	Result
Case 0	Case 1

height=

```
[1,8,6,2,5,4,8,3,7]
```

Console ▾

Run

Therefore, we came up with the 3 schema designs for our question service database, namely, [question schema](#), [code template schema](#), and [test case schema](#). It will definitely help in the question service's NFR of [performance and efficiency](#).

3.2 Matching Service Database Design

In this service, we only make use of one schema that defines the WaitingList to handle the logic of matching users. The data is stored in a SQL database.

3.2.1 WaitingList Schema

```
4  const WaitingList = sq.define("WaitingList", {
5    username: {
6      type: DataTypes.STRING,
7      allowNull: false,
8      primaryKey: true
9    },
10   complexity: {
11     type: DataTypes.ENUM(['Easy', 'Medium', 'Hard']),
12     allowNull: false
13   },
14   roomId: {
15     type: DataTypes.STRING,
16     allowNull: false,
17   },
18 });

});
```

Attribute	Description and rationale
Username	Type is a string, and represents the username of the person who is waiting to be matched
Complexity	Type is an ENUM that can only hold values ‘Easy’, ‘Medium’, and ‘Hard’. We do not want other values to be allowed because these are the only valid complexity for our application
roomId	Type is of type string, which is a UUIDV4 generated ID. This represents the room id that the user will be in after he gets matched

3.3 User Service Database Design

In this service, we create a schema that stores the details of a user with the help of a SQL database.

3.3.1 User Schema

```
4  const User = sq.define("user", {
5    username: {
6      type: DataTypes.STRING,
7      allowNull: false,
8      primaryKey: true
9    },
10   password: {
11     type: DataTypes.STRING,
12     allowNull: false
13   },
14   isManager: {
15     type: DataTypes.BOOLEAN,
16     defaultValue: false
17 }
18 })
```

Attribute	Description and rationale
Username	Type is a string, and represents the username of the user
password	Type is a string, and represents the hashed value of the user
isManager	Type is a boolean. If it is true, then the user is a manager, else, he is just a normal user

3.4 Collaboration Database Design

In this service, we have a schema that stores the details of the room with the help of a SQL database.

3.4.1 Room Schema

```
4  const Room = sq.define("Room", {
5    roomId: {
6      type: DataTypes.STRING,
7      allowNull: false,
8      primaryKey: true,
9    },
10   questionId: {
11     type: DataTypes.STRING,
12     allowNull: false,
13   },
14 });


```

Attribute	Description and rationale
roomId	Stores the room id (UUIDv4) of the room, of type string
questionId	Represents the question id for the room, of type string. This assists the user to get know what question to get when joining the room

3.5 History Database Design

The history service uses an SQL database to hold the history of all users.

3.5.1 AnsweredQuestion Schema

```
id: {
    type: DataTypes.INTEGER,
    autoIncrement: true,
    primaryKey: true
},
questionId: {
    type: DataTypes.STRING
},
questionTitle: {
    type: DataTypes.STRING
},
complexity: {
    type: DataTypes.STRING
},
username: {
    type: DataTypes.STRING
},
username2: {
    type: DataTypes.STRING
},
answeredAt: {
    type: DataTypes.DATE
},
isSolved: {
    type: DataTypes.BOOLEAN
},
roomId: {
    type: DataTypes.UUID
}
```

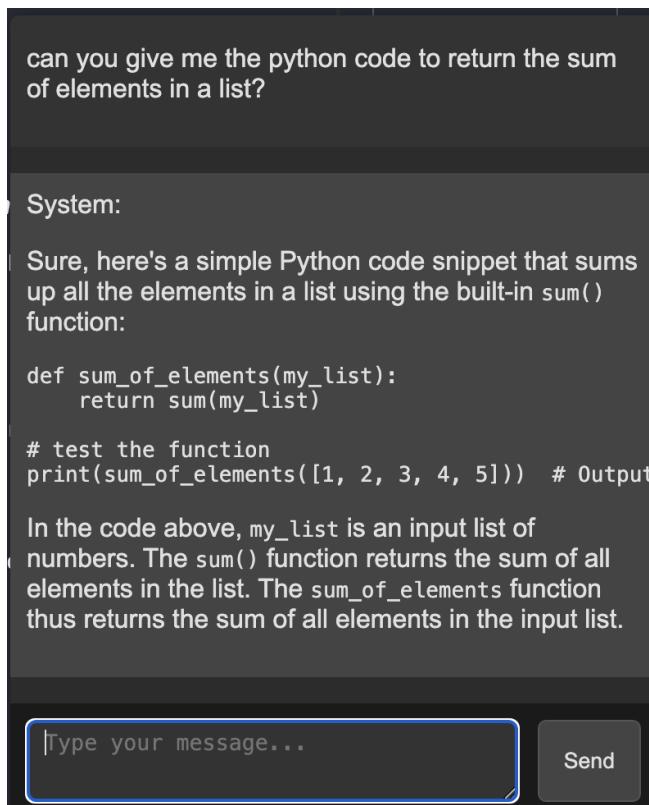
Attribute	Description & Rationale
id	Automatic numbering system handled by the database.
questionId	Represents the question id of the question solved, for easy retrieval of question details in the future should we wish to store details in other databases
questionTitle	Question title is stored for now so that loading a user's history is fast because there is no need to query our question repository.
complexity	Same reasoning as per question title.
username	Represents the username of the user who executed the code.
username2	Represents the name of the other user who answered the question (if present)
answeredAt	Represents the timestamp of user/users' attempt to answer the question.
isSolved	An answered question represents an attempt at answering and the user might not have necessarily solved the question. This indicates whether all the test cases passed, which by our standards is solving the question.
roomId	Represents a unique ID akin to a session ID, so only 1 solved and 1 unsolved attempt is recorded in the database.

4. Service-Specific Design

In this section, we discuss specific implementations and design decisions behind each microservice.

4.1 AI Service

The AI Chat Service is an interactive chatbot feature, integrating OpenAI's GPT-4 model in a React application. This provides a convenient and seamless AI-powered coding assistance for the users.



4.1.1 Front End Design

The main component, Chat, features a draggable chatbox where users can input queries, and the response will be fetched from ChatGPT's OpenAI API. This chatbox can be closed and moved around the page at any time, making sure that it does not block the view of the question or the coding panel.

4.1.2 Key Features

- Real-Time AI Conversations: Users interact with an AI chatbot, receiving immediate responses.
- Draggable Chat Interface: Enhances user experience with a movable chat window.
- Message History: Displays an ongoing conversation with scrolling to the latest messages.

4.1.3 Implementation

- React States: Manages user input, chat visibility, and message history.
- Axios for API Calls: Sends user messages to the backend ai-service and receives AI responses.
- Event Handling: Includes functionality for sending messages on Enter key press.

4.1.4 Back End Design (AI-Service microservice)

The backend consists of a Node.js express-based server integrating OpenAI's GPT-4 model to process and respond to user inputs.

The key functionalities are as follow:

- REST API Endpoint: Handles POST requests to /ai-service/chatgpt.
- OpenAI Integration: Utilizes the OpenAI API to generate chatbot responses.

4.1.5 Model Choice

The main reason why we chose to integrate OpenAI's GPT-4 model as it has excellent natural language processing capabilities, and provides useful responses to queries related to coding. This is suitable for our application as users will be able to get an answer to their queries relating to the current question they are solving, or for coding in general.

4.2 Code Execution Service

The code execution service handles a user's request to execute the code that is present in their shared code editor. This provides a way for users to test the code that they have written, so that they will have a way to know whether they've solved the question correctly or wrongly. This code execution results will be tested against our own test cases.

4.2.1 Front End Design

We have 2 tabs within our code execution UI. The “Testcase” tab will show the user the test case input, without the expected output. This helps to prevent spoilers for the user.

This screenshot shows the 'Testcase' tab of a dark-themed application. At the top, there are two tabs: 'Testcase' and 'Result'. Below the tabs, there are two buttons: 'Case 0' (which is selected) and 'Case 1'. A text input field labeled 'n=' contains the value '3'. In the bottom right corner of the main area, there is a yellow 'RUN' button.

The Result tab shows the user the test case's expected output, and their output on running the application

This screenshot shows the 'Result' tab of the same application. It has the same layout as the 'Testcase' tab, with 'Testcase' and 'Result' tabs at the top, 'Case 0' selected, and a 'n=' input field containing '3'. However, the 'Output=' section is now visible, showing the text 'Run to see output'. The 'Expected=' section also shows the value '3'. The 'Console' and 'RUN' buttons are at the bottom.

UI for failed test case

Testcase	Result
Case 0	Case 1
s=	
ACBBD	
Output=	
10	
Expected=	
5	

Console ▾ RUN

UI for passed test case

Testcase	Result
Case 0	Case 1
s=	
ABFCACDB	
Output=	
2	
Expected=	
2	

Console ▾ RUN

4.2.2 Judge0 API Call

In our application, we utilize the Judge0 API for code execution. To efficiently handle multiple test cases, we aggregate the expected inputs and outputs, creating a batch. This batch is then submitted via a POST request to the specified URL (/submissions/batch) using our private key. The API responds with a token, which we employ to query the API again, checking the status at the URL (/submissions/batch).

Given that results can be in various states such as "Processing" or "In Queue," we implement a polling mechanism, making up to 10 attempts to retrieve results before signaling an error indicating server overload.

Following retrieval, we parse the results and conduct a comparison of the standard output (stdout) with our expected output for each test case. This enables us to inform the user about which test cases have passed and failed.

4.2.3 Design rationale

By making use of batch submission, we can reduce the network traffic between our service and the Judge0 server. If we were to send single submissions, if the user executes the code for a question with multiple test cases, there will be multiple requests sent and multiple polling processes.

4.3 Collaboration Service

The collaboration service specifically refers to the real-time IDE within the room component. The functionality of the service comes from two main components; the CodeMirror library, and WebSockets. We discuss the implementation of each component below.

4.3.1 CodeMirror

CodeMirror is a powerful JavaScript library that offers IDE-related components. Its main export is a fully-fledged IDE which can be created via a dynamic binding to some “textarea” component on the HTML page. The IDE supports language-specific syntax highlighting, as well as dynamic code-language changing.

```
editorRef.current = Codemirror.fromTextArea(
  document.getElementById("realtimeEditor"),
  {
    mode: { name: "python", json: true },
    theme: "dracula",
    autoCloseTags: true,
    autoCloseBrackets: true,
    lineNumbers: true,
  }
);
```

Code for the IDE

As described above, we simply create a HTML “textarea” component on our page, then call a function from the CodeMirror library. The IDE will then be rendered in place of the textarea component. We can also pass in parameters such as default language, theme and other syntactical choices.

4.3.2 WebSocket Design

We wanted the IDE to be real-time and reflect user-made changes to other users in the same room. We felt that this was important so that the peer-programming experience was more dynamic. By allowing users to see the other user’s changes live, we could convey the “real-time” feeling of an actual live programming interview, which reflects PeerPreps overall goal.

By default, CodeMirror’s IDE will only reflect changes to the client using it. Thus, to ensure that other users would also be able to see changes to the IDE in real-time, we decided to use WebSockets.

WebSockets enable bidirectional communication between a client and a server over a single, long-lasting connection. They are designed to enhance interactivity and provide real-time functionality in web applications.

4.3.3 Database Setup

We created a database to store data relating to the room ID and relevant user IDs. The database schema is described in section 3.4 above. By using a database, we can support the creation of and joining a custom room. Specifically, when a user wants to join a custom-made room, the database is first checked for the existence of some room with the requested room ID.

4.3.4 Websocket Setup

To set up our socket system, we first implemented server side code using Express. We then defined the socket server instance and configured the CORS to allow for GET, POST, PUT and DELETE methods. This is because our sockets would be interacting with the database as explained above.

4.3.5 Stateful Component Setup

We defined three stored states; CodeRef, EditorRef and SocketRef. CodeRef stores the current code inside the IDE, EditorRef stores the current IDE object while SocketRef stores the client socket. Below, we discuss why we needed each reference.

CodeRef: Enables “live” coding by storing an updatable reference of the code inside the editor. This reference is updated when another user changes the contents of the IDE, then passed to the client’s EditorRef instance.

EditorRef: Enables dynamic functions such as language-specific syntax highlighting. When a user changes the code language of the IDE, the server is notified of the new language. The server then informs all other clients in the room of the new language. Other clients can then update their EditorRef with the new language to observe the correct syntax highlighting.

SocketRef: Defines actions that the client’s socket should perform, and also events that it should listen for. The SocketRef also allows us to define a socket ID and a username for the client’s socket, which it uses when communicating with the socket server.

4.3.6 Action Codes

Action codes are specific strings that relate to defined actions to be carried out by the socket instance. Defining such action codes allows us to easily identify and call various socket functions.

Each action is defined below:

JOIN: Action code indicating that a client socket has established a connection to the server socket. We require this code to be sent from the client side so that the server can add the client socket to the “room”.

```
socket.on(ACTIONS.JOIN, async ({ roomId }) => {
  await clientMapRepo.addEntry(socket.id, socket.username);
  const clients = await getAllConnectedClients(roomId);

  if (clients.length >= 2) {
    io.to(socket.id).emit(ACTIONS.JOIN_FAILED);
    return;
  }

  console.log(`${socket.username} connected to room ${roomId}`);

  socket.join(roomId);
  // await clientMapRepo.addEntry(socket.id, socket.username);
  clients.push({ socketId: socket.id, username: socket.username });

  console.log(clients);

  clients.forEach(({ socketId }) => {
    io.to(socketId).emit(ACTIONS.JOINED, {
      clients,
      socketId: socket.id,
      username: socket.username,
      fromSocket: socketId
    });
  });
});
```

Logic for JOIN action

Once the server receives the action code, they will add the client object to the room database, specifically the client’s socket ID and username. The server will also inform the client socket of any other clients that are currently in the room. This is essential to convey information about all clients to each other.

DISCONNECTED: Action code indicating that a client socket has left the room. This code is sent from client to server when the client leaves the room in any way, for example by clicking the “leave room” button. The server will then remove the relevant client row in the database and inform all other clients that a specific client has left the room. This is to ensure that various components that rely on this information, such as the dynamic user icons, are updated correctly.

```

socket.on("disconnecting", async () => {
  const clientMapping = await clientMapRepo.getBySocketId(socket.id);

  socket.in(socket.roomId).emit(ACTIONS.DISCONNECTED, {
    socketId: socket.id,
    username: clientMapping.username
  });

  await clientMapRepo.deleteBySocketId(socket.id);
  socket.leave();
});

```

Logic for DISCONNECTED action

CODE-CHANGE: Action code indicating that the code in the IDE has changed. This is the core of how we enable the IDE to be “shared” between users. Once either user types in the IDE, their socket will inform the server that the value (code) of the IDE has changed. The server will then inform the other client sockets in the room of this change and also pass the updated value, which is passed to the IDE component. This gives the impression of “live” updating.

```

editorRef.current.on("change", (instance, changes) => {
  const { origin } = changes;
  const code = instance.getValue();
  onCodeChange(code);

  if (origin !== "setValue") {
    socketRef.current.emit(ACTIONS.CODE_CHANGE, {
      roomId,
      code,
    });
  }
});

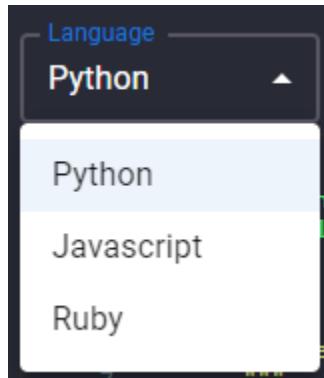
```

Logic for CODE_CHANGE action

SYNC_CODE: Action code used for syncing a user’s IDE to the shared IDE state in the room when they join. This code is mainly used in our custom room feature, where a user first creates a room, then invites other users to join by sending them the room ID. When another user joins the room, they need to know what code exists in the shared editor. Thus, when the new user joins, their socket will send a SYNC_CODE code to the socket server, which will then return the current code in the live editor.

JOIN_FAILED: Action code used to inform clients that the room is full. This code is sent by the server to users when they try to join a room which already has 2 people. This design is meant to enforce our principle of having 2 people per coding session.

CHANGE_LANGUAGE: Action code indicating that the code language in the editor has changed. This code is first sent by the client that wants to change the editor language to the server when the dropdown option for code language is changed.



Dropdown menu for code language

When the server receives the code, it will inform all other sockets in the room to change their editor language by emitting a CHANGE_LANGUAGE code. The other sockets will then update their EditorRef and CodeRef accordingly.

```
const handleLanguageChange = (e) => {
  setLanguage(e.target.value);
  socketRef.current.emit(ACTIONS.CHANGE_LANGUAGE, {
    roomId,
    language: e.target.value,
  });
};
```

Logic for sending CHANGE_LANGUAGE code

EXECUTE_CODE: Action code for executing the code present in the IDE. This code is sent from client to server after the API call has been made to Judge0 to run the code in the IDE. The action code indicates whether or not all test cases were passed for the current question. If so, then the UI of the room will be updated for both users to indicate a successful solution. Otherwise, the UI will indicate that some test cases failed.

```

useEffect(() => {
  if (socketRef.current) {
    socketRef.current.on(ACTIONS.EXECUTE_CODE, ({ result }) => {
      if (Array.isArray(result)) {
        const allPassed = result.every(item => item.status === "Passed");
        if (allPassed) {
          setAllPassModalVisible(true);
        }
      }

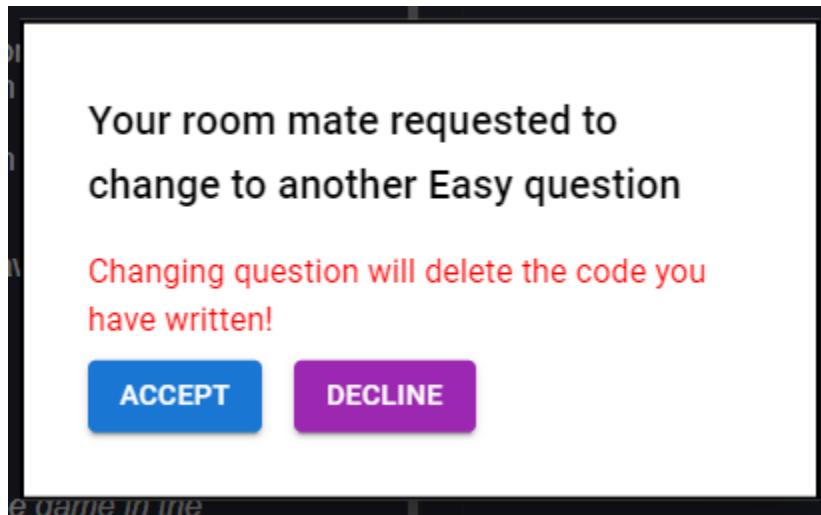
      setExecutionResults(result);
      setResultsVisible(true);
      setTestCaseBtnSelected(false);
    });
  }
}, [socketRef.current]);

```

Logic for sending EXECUTE_CODE code

REQUEST_QUESTION_CHANGE, DECLINE_QUESTION_CHANGE, CHANGE_QUESTION,

SYNC_QUESTION: These codes both facilitate the changing question functionality within a room. The current question in the room can be changed by clicking on the “Change Question” button. When a user clicks the button, a REQUEST_QUESTION_CHANGE code is sent to the socket server. The server then informs the other client socket in the room of this request, and opens a pop-up in the front-end to inform the user.

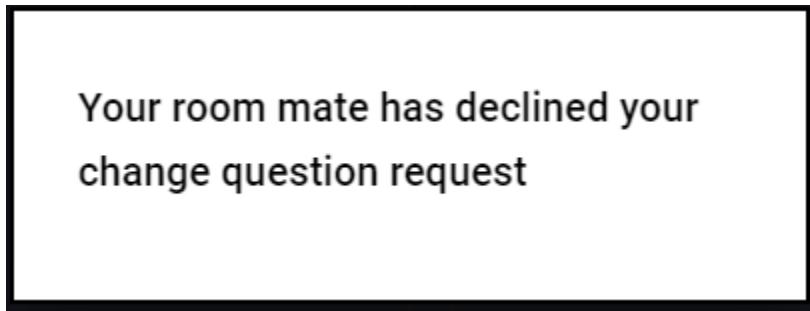


Pop-up when a change question request is made

The other user has the option to accept or decline the request. If the other user accepts, then a new question of the specified difficulty is chosen (randomly) and used as the question for the

room. A SYNC_QUESTION code is then sent to the server, which informs other client sockets to update their question to the new question. A CHANGE_QUESTION code is also sent to the server, which is in charge of handling the state of the dropdown menu; upon a request being accepted, the other user's selected dropdown menu option is updated.

On the other hand, if the other user declines, then a DECLINE_QUESTION_CHANGE code is sent to the server, which informs the requesting client socket. A modal appears on the requesting user's screen, telling them that their request to change the question was declined. No CHANGE_QUESTION code is sent, and the UI of the dropdown menu is unchanged.



Modal when a request is declined

4.4 Communication Service

The Communication Microservice is a React-based module designed for establishing and managing peer-to-peer video connections. It utilizes simple-peer for WebRTC-based communication and socket.io-client for signaling and coordination of peer connections. The service includes components for rendering video streams (PeerVideo) and managing overall video communication (Video). The service also utilizes the socket.io-client to allow real-time text communication between 2 users.

4.4.1 Video Calling Feature

Two main key components are implemented in this feature (PeerVideo and Video component)

PeerVideo is a functional component responsible for rendering a single peer's video stream.

Implementation:

- WebRTC Stream Rendering: Utilizes simple-peer to receive and display a video stream from a peer.
- React Hooks: Uses useRef to reference the video DOM element and useEffect to attach the stream to the video element when it becomes available.

Video is the main component that orchestrates the video communication, handling user media, peer connections, and UI interactions.

Key Features

- Dynamic Peer Management: Dynamically manages peer connections as users join or leave a room.
- Socket.IO Integration: Uses socket.io-client for coordinating peer connections.
- User Media Management: Acquires and manages user media streams (video and audio).
- UI Controls: Provides UI elements for controlling the video stream and displaying peer videos.

Technical Details:

- State Management: Uses useState and useRef for managing state and references.
- Socket Event Handling: Implements event listeners for socket events such as user joined, user-disconnected, and receiving returned signal.
- Peer Connection Lifecycle: Manages creation, signaling, and destruction of peer connections.
- Environmental Configuration: Configures socket connection using environment variables.

Methods

- createPeer(userToSignal, callerID, stream): Initializes a new peer connection for outgoing calls.
- addPeer(incomingSignal, callerID, stream): Handles incoming peer connections.

- `toggleVideo()`: Toggles the local video stream on and off.
- `turnOffAllMediaStreams()`: Utility method to stop all media streams in the document.

UI Considerations (Draggable library)

- Draggable UI: Incorporated react-draggable to allow users to reposition their video feeds on the screen. We chose this library because we did not want the video feeds to block the user's view of the question or the code, hindering the user experience.

Library Choice (simple-peer and webRTC)

The key library used for this feature is simple-peer, a lightweight wrapper around WebRTC's complex APIs, making it easier to use. It abstracts away many of the low-level details required to set up and manage a WebRTC connection, allowing developers to focus on higher-level application logic. Personally, I have utilized WebRTC before in several other applications and the simple-peer library really helps to simplify the process.

The key benefits that the simple-peer library offers is:

- 1) Simplification of WebRTC - being a lightweight wrapper of WebRTC, it simplifies the process of utilizing WebRTC
- 2) Cross-browser compatibility - It handles differences and inconsistencies in the WebRTC implementation across various browsers, providing a more consistent behavior.
- 3) Open-source nature - Being open-source, it benefits from community contributions, regular updates, and bug fixes.

The benefits of WebRTC are:

- 1) Real-Time Communication: WebRTC (Web Real-Time Communication) is an open-source project that provides web browsers and mobile applications with real-time communication (RTC) capabilities via simple APIs. It supports video, audio, and data communication without needing to install plugins or third-party software.
- 2) Peer-to-Peer Connectivity: One of WebRTC's key features is its ability to establish direct peer-to-peer connections. This reduces latency and improves the quality of video and audio transmission, as data doesn't need to be routed through a server.
- 3) NAT Traversal: WebRTC handles Network Address Translator (NAT) traversal, which is crucial for establishing connections between devices that might be behind different types of NATs on the internet. It uses techniques like STUN and TURN to achieve this.

Conclusion:

For the video calling features, the key considerations that we took into place were:

- 1) Allow for quick, reliable exchanging of data between two clients without the use of any server
- 2) Allow support for different browsers
- 3) Firewall considerations

Because of this, the simple-peer and webRTC library is thus suitable for the implementation of this feature.

4.4.2 Chat Feature

The Chat component is a real-time messaging feature implemented in a React application, designed for exchanging messages between clients in a chat room. It uses socket.io for WebSocket communication to enable real-time, bidirectional, event-based communication.

Front End: Text Chat Component (Chat)

- The Chat component allows users to send and receive messages in real-time.
- It is designed to be part of a larger application, where the chat room ID is dynamically set.

Key Features

- Real-Time Messaging: Users can send and receive messages instantaneously.
- Draggable and Closable Chat Window: The chat interface is draggable, allowing users to reposition it on the screen. This has the same rationale as the video feature, where we don't want this to block the question/code
- Room-Based Communication: Users join specific rooms for isolated chatting environments.

Implementation

- State Management: Uses React's useState and useRef for managing component state and references.
- WebSocket Connection: Establishes a WebSocket connection to the server using socket.io-client.
- Event Handling: Listens for key events like connect, connect_error, and receive-message.
- Message Sending: Allows users to type and send messages, which are then emitted to the server.

Back-End: WebSocket Server

- The server is set up using Express with socket.io for handling WebSocket connections.
- It manages user connections, room-based interactions, and real-time messaging.

Key Functionalities

- Room Management: Handles users joining specific chat rooms.
- Message Broadcasting: Broadcasts messages to the other user
- Connection Lifecycle: Manages user connections and disconnections, ensuring proper room clean-up.

Event Handling

- on 'join': Adds a user to a specific chat room.
- on 'message': Broadcasts a message to the other user in the room
- on 'disconnect': Removes the user from the room and updates other users in the room.

Usage

The server needs to be running and accessible for the front-end Chat component to function correctly.

Library Choice (Socket.io)

The key reasons why we chose this library:

- 1) Bidirectional Communication: Supports two-way data flow, eliminating repetitive server polling and reducing network load.
- 2) Reduced Latency: Offers faster data transmission with persistent connections, ideal for frequent small data exchanges.
- 3) Scalability: Efficiently handles numerous concurrent connections, scalable across multiple servers.

Conclusion

Socket communication offers a robust solution for real-time, efficient, and interactive web applications. Its ability to provide instant, two-way communication with reduced latency and network overhead makes it an ideal choice for a wide range of applications that require real-time data exchange. For a feature such as text communication, the real time nature is a priority (short to no delays) and Socket.io provides nice scalability for this purpose.

4.5 History Service

The history service serves the crucial purpose of logging and storing users' attempts at solving coding questions. This functionality is invaluable for both learners and educators. For users, the history service allows them to review their progress, track improvements, and learn from their mistakes. It provides a personalized learning journey by offering insights into problem-solving strategies and iterative development. Additionally, the history service contributes to a collaborative learning environment, as users can share their coding attempts with peers or instructors for feedback and discussion.

4.5.1 Front End Design

The screenshot shows the PeerPrep application interface. On the left, there is a sidebar with a user profile icon (circle with 'u'), the username 'username', and a role indicator 'Role: USER'. Below this are two buttons: 'CHANGE PASSWORD' (blue) and 'DELETE ACCOUNT' (red). The main content area has a title 'Answered Questions ⓘ'. A table lists one question: 'Minimum Cuts to Divide a Circle' solved on 'Nov 15 2023, 459 hrs' with a complexity of 'Easy'. Below this is a 'Solved Questions' section with a progress bar. The progress bar shows 0 solved out of 7 Easy questions, 0 solved out of 23 Medium questions, and 0 solved out of 18 Hard questions. The entire interface has a dark theme with light-colored text and icons.

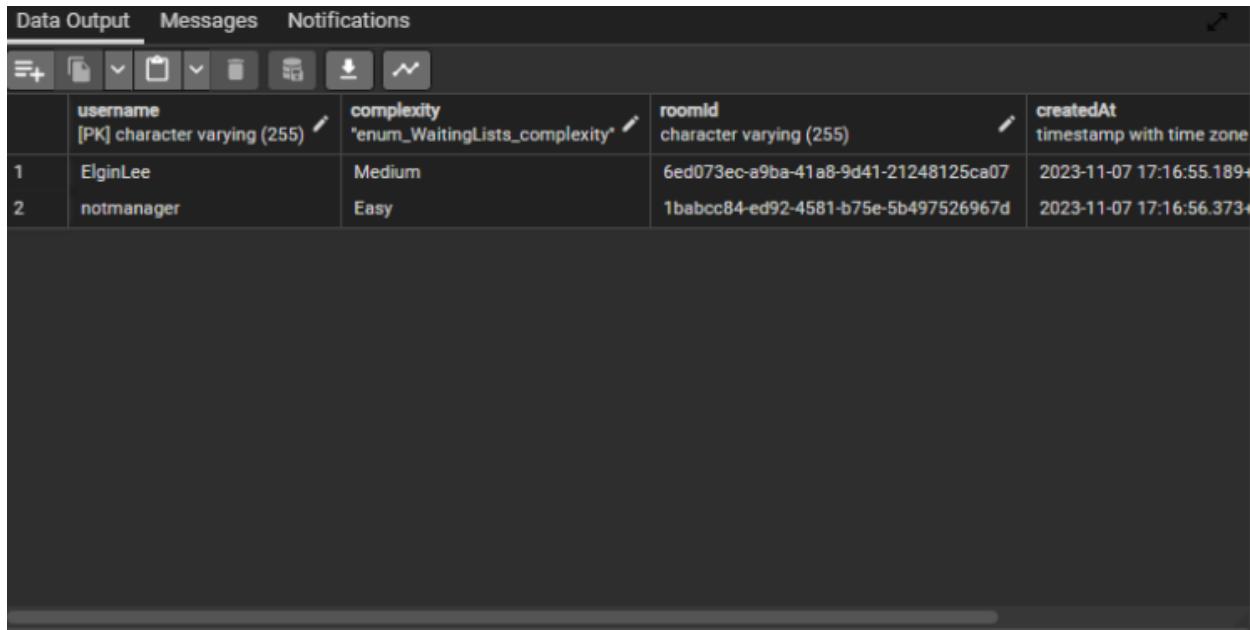
4.5.2 Key Features

- History Table: A user's attempts at answering a question is stored in the database. The history table translates the data into something more readable for the users.
- Progress Bar: The progress bar tracks the number of easy, medium and hard questions solved by the user.

4.6 Matching Service

The design of our matching service makes use of a database, with the schema [here](#).

Image showing database with sample values



A screenshot of a database management tool interface. The top navigation bar includes tabs for "Data Output", "Messages", and "Notifications". Below the navigation bar is a toolbar with various icons for database operations like insert, update, delete, and search. The main area displays a table with four columns and two rows of data. The columns are labeled: "username [PK] character varying (255)", "complexity 'enum_WaitingLists_complexity'", "roomid character varying (255)", and "createdAt timestamp with time zone". The first row contains the value "ElginLee" for "username", "Medium" for "complexity", "6ed073ec-a9ba-41a8-9d41-21248125ca07" for "roomid", and "2023-11-07 17:16:55.189+00:00" for "createdAt". The second row contains the value "notmanager" for "username", "Easy" for "complexity", "1babcc84-ed92-4581-b75e-5b497526967d" for "roomid", and "2023-11-07 17:16:56.373+00:00" for "createdAt".

	username [PK] character varying (255)	complexity 'enum_WaitingLists_complexity'	roomid character varying (255)	createdAt timestamp with time zone
1	ElginLee	Medium	6ed073ec-a9ba-41a8-9d41-21248125ca07	2023-11-07 17:16:55.189+00:00
2	notmanager	Easy	1babcc84-ed92-4581-b75e-5b497526967d	2023-11-07 17:16:56.373+00:00

It is implemented in such a way that it is based on a FIFO system according to the complexity. As such, the “complexity” forms our 3 different queues (“Easy”, “Medium”, “Hard”).

Alternative implementation with in memory array

We could have made use of 3 in memory arrays to be our queue. However, it makes our matching service less scalable because it makes the application stateful. Therefore, we chose to use a database instead.

Alternative implementation with a message queue

We could have done this matching functionality with a message queue. This works by having our matching-service publish a message to the queue when a user joins the queue. It will also have a consumer that does the matching for users. However, this has some limitations that made us choose to use a database instead. Firstly, using a database allows for more flexibility in the future when we want to implement more complex matchmaking that involves more than just complexity. Secondly, we do not see much difference between using a database as a queue, compared to using the publisher and consumer will be within the same service (in other words we’re just utilizing it like a database). Hence, we decided to go with a database design for matching-service for future flexibility.

4.7 Question Service

This service's sole purpose is to handle the questions of PeerPrep. The sole design considerations of the database can be found [here](#). The REST API endpoints could be found [here](#). The reason why we chose to use HTTP is due to the widespread support for browsers, which is the main platform our application will run on. Regarding the question service's security, it can be found [here](#).

4.8 User Service

This service's sole purpose is to handle the authentication aspect of PeerPrep. It is an express-based node backend server which provides authentication-related endpoints for the frontend to interact with. The sole design considerations of the authentication considerations can be found [here](#). The REST API endpoints could be found [here](#). Regarding the question service's security, it can be found [here](#).

5. Frontend

5.1 React

We chose React as our frontend framework as it enables us to utilize a component based architecture where we can reuse several UI components, making the code more modular and maintainable. Furthermore, it supports Hot Module Replacement (HMR) which allows results of code changes to be seen in real time without the need to refresh a page during development. Lastly, since React has a rich ecosystem, with many libraries and tools such as state management libraries, routing libraries, and UI component libraries, it can help us improve our development flow.

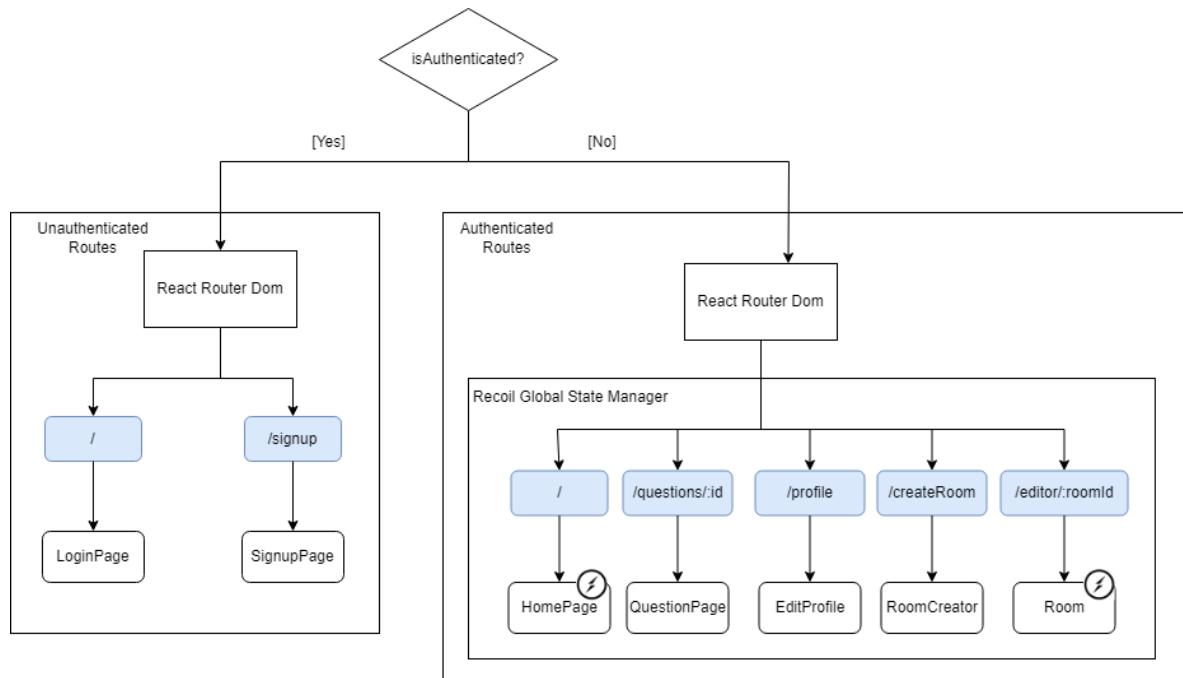
5.2 Architecture Diagram

This diagram gives a high level overview of how our frontend is structured.

We first check whether a user is authenticated via the existence of a JWT token stored in its local storage. If there is such a token, we will mark this user as an authenticated user, and he will interact with the application within the Authenticated Routes. Otherwise, the user will be interacting with the application at the unauthenticated route.

Within the unauthenticated routes, the user can access the LoginPage and SignUp page via the routes defined.

Within the authenticated routes, the user can access the HomePage, QuestionPage, EditProfile, RoomCreator, and Room



Our group also utilized Recoil as our Global State Manager. We needed this for our countdown timer on joining a queue, which provides the user with the flexibility of navigating to other pages within the authenticated routes without leaving a queue. This adds to the usability of our application since they can do other things like viewing questions while waiting to be matched.

5.2.1 Legend

Symbol	Meaning
Diamond	A decision boundary that splits the routes into 2.
Blue rounded box	Signifies the path to access that particular React component.
Uncoloured rounded box	Signifies that it is a React Component.
Squared Box	No special meaning, but just to group the other entities for clarity.
	Websocket. Any service with this symbol denotes that websocket is utilized.

5.2.2 Design Decision-Making Process

Our design decision to check whether a user is authenticated via only checking for the existence of JWT token in the user's local storage lies in the reduction of network traffic between the frontend and the user service. Furthermore, all services authenticate the JWT Token attached to the header of a request, including any attempts to make a websocket connection. Even if a malicious user attempts to enter the authenticated routes with a fake JWT token, he will not be able to establish anything meaningful.

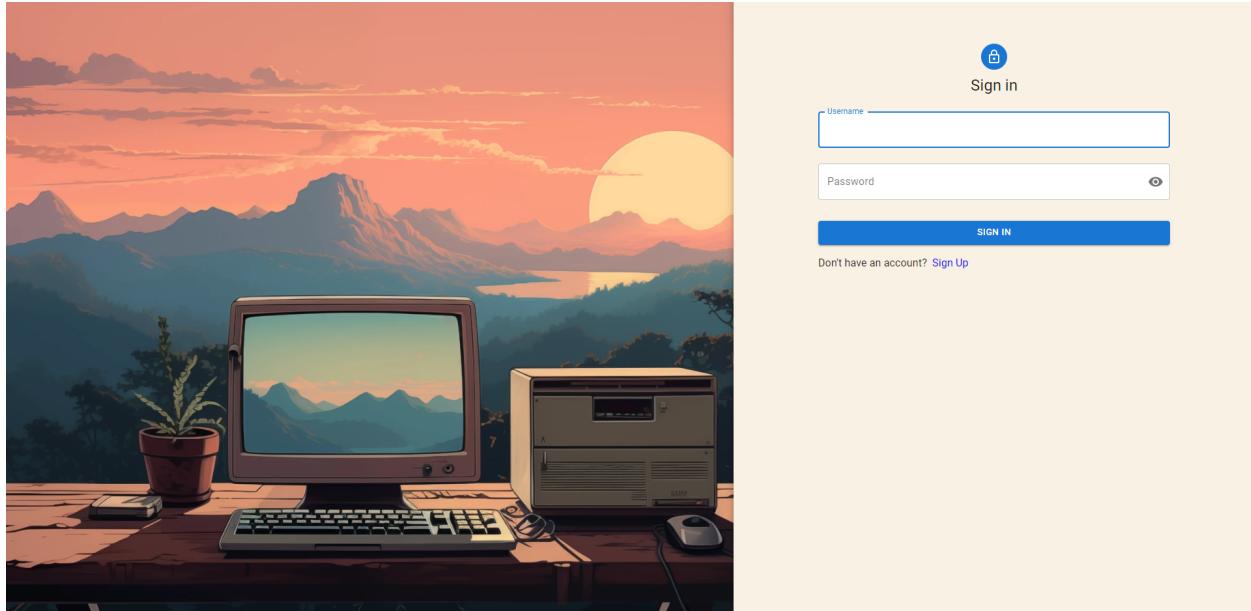
5.3 Key Libraries Utilized

Package	Rationale
socket.io-client	Allows for bidirectional communication between the frontend and service, and is a fairly popular package
Recoil	A global state manager for us to implement a countdown timer, providing the user with the capability of navigating to other pages while remaining in queue. We did not use Redux because it involves much more boilerplate code and complexity compared to Recoil. For our simple use case, we decided that recoil is a perfect choice for us.
react-draft-wysiwyg	A user-friendly interface for users to type their content when adding a new question, in addition to supporting images.
CodeMirror	For the room's collaborative editor with syntax highlighting with multiple language support
Dompurify	Since we need to inject html code to render our question, we utilized this library to protect our application from malicious actors who might attempt a XSS attack.
Axios	A simple to use way to send CRUD API requests to the backend
React-router-dom	Used to navigate between pages, to help maintain SPA (Single Page Application)
Material UI	Material-UI library that offers React components following Material Design guidelines for building a UI with a consistent design.
marked-react	A React library for parsing and rendering Markdown text.
openai	The official OpenAI API client for JavaScript, used for integrating OpenAI functionalities.
react-draggable	Enables draggable functionality for React components.
simple-peer	A library for WebRTC peer-to-peer communication, commonly used for video/audio streaming features.

6. UI

We designed our UI to be intuitive and easy to understand. Below, we discuss the rationale behind the design of each page in our application, and how it improves user experience.

6.1 Login Page



Our login page is displayed above. Various design choices are explained below.

Placement of Login Fields: The username and password fields are centrally located on the right half of the screen. This makes them the focal point of the page, ensuring that users know exactly where to input their credentials.

Call-to-Action Button: The "SIGN IN" button is prominent and colored differently from other elements, making it clear that it is the next step after entering credentials.

System Feedback: There is an eye icon in the password field, which allows users to toggle the visibility of their password. This provides users with control over their password input and feedback on their actions.

6.2 Sign Up Page

The screenshot shows a clean, minimalist sign-up interface. At the top center, there's a blue circular icon with a white padlock symbol, followed by the text "Sign Up". Below this are three input fields: "Username", "Password", and "Confirm Password", each accompanied by a small eye icon to the right for password visibility. A large blue rectangular button labeled "REGISTER" is centered below the input fields. At the bottom right of the button, the text "Already have an account? Log In" is displayed.

We discuss the functionality and design choices of our sign up page below.

Contrasting Call-to-Action Button: The "REGISTER" button is large and in a contrasting blue color, making it easily identifiable as the next step after filling out the form.

Logical Flow: The layout follows a logical flow, from top to bottom, which is the natural reading order for most users. This arrangement aligns with users' expectations for how a form should be structured.

6.3 Home Page

The screenshot shows the PeerPrep home page. At the top, there is a navigation bar with icons for profile, search, and notifications, followed by the text "PeerPrep". To the right are "CUSTOM ROOM" and a user icon. Below the navigation bar is a dark header with the text "Match with another user" and three buttons: "Queue Easy" (green), "Queue Medium" (orange), and "Queue Hard" (red). The main content area has a dark background with a light gray header containing "All", "Easy", "Medium", and "Hard" tabs. Below this is a table with columns "Title", "Category", and "Complexity". The table lists 15 questions, each with its title, category (e.g., Array, Bit Manipulation, String, Greedy, Math, etc.), and complexity level (Easy, Medium, Hard).

Title	Category	Complexity
1. Find the K-th of an Array	Array, Bit Manipulation	Easy
2. Maximum Points After Collecting Coins From All Nodes :)	Array, Tree	Hard
3. Minimum Sum of Mountain Triplets I	Array	Easy
4. Minimum Number of Changes to Make Binary String Beautiful	String, Greedy	Medium
5. Divisible and Non-divisible Sums Difference	Math	Easy
6. Minimum Equal Sum of Two Arrays After Replacing Zeros	Array, Greedy	Medium
7. Minimum Number of Groups to Create a Valid Assignment	Array, Hash Table, Greedy	Medium
8. Find Indices With Index and Value Difference I	Array	Easy
9. Last Visited Integers	Array, String, Simulation	Easy
10. Subarrays Distinct Element Sum of Squares I	Array, Hash Table	Easy
11. Maximum Value of an Ordered Triplet I	Array	Easy
12. Apply Operations on Array to Maximize Sum of Squares	Array, Hash Table, Greedy, Bit Manipulation	Hard
13. Maximum Value of an Ordered Triplet II	Array	Medium
14. Minimum Processing Time	Array, Greedy, Sorting	Medium
15. Shortest and Lexicographically Smallest Beautiful String	String, Sliding Window	Medium

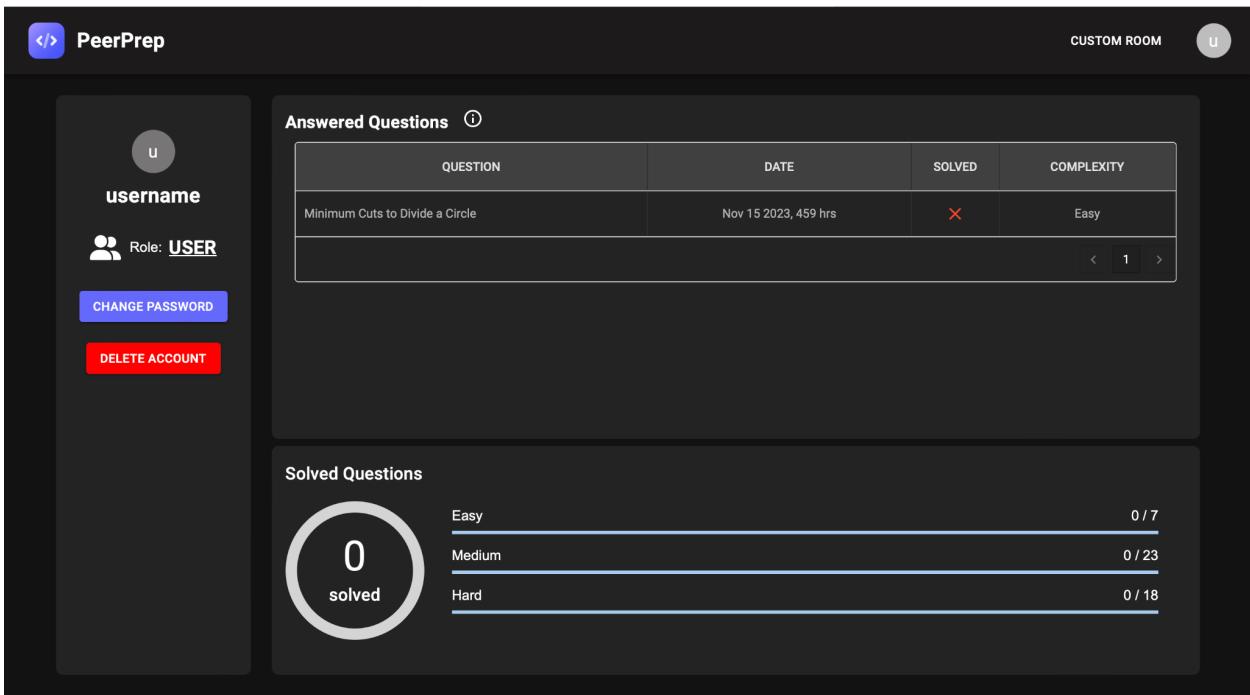
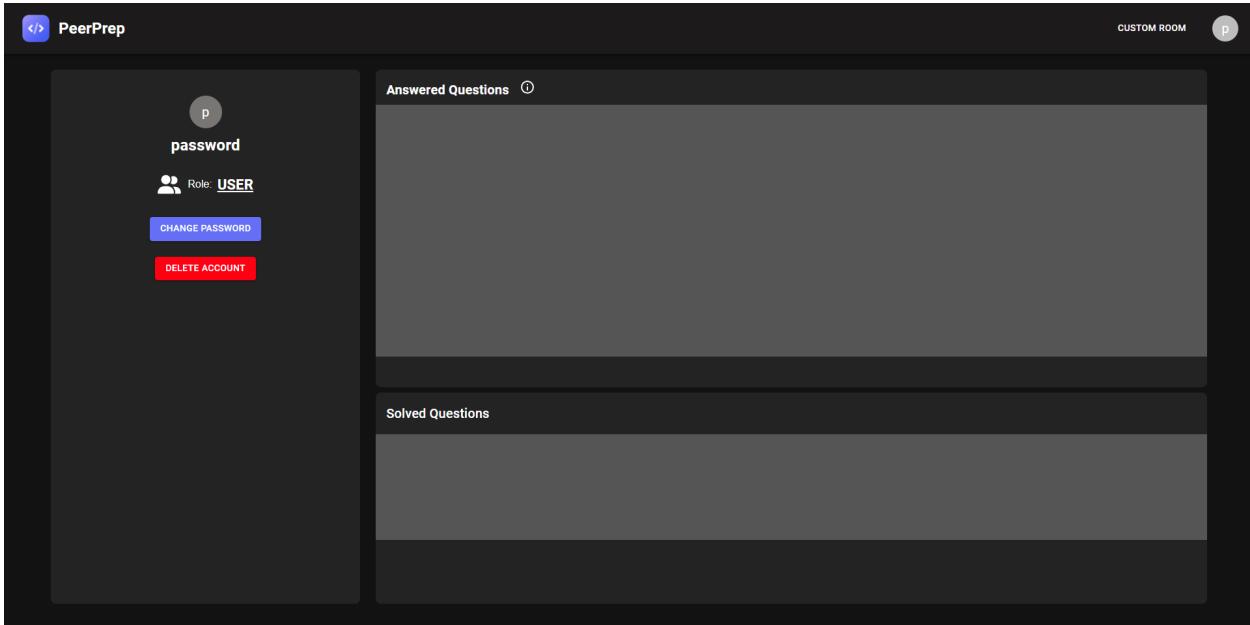
Our home page displays the question repository and the matching buttons. At the top, we have a navigation bar. We discuss the functionality and design choices of our home page below.

Navigation Bar: The navigation bar helps users quickly locate and navigate to different sections of the site. It provides a clear path for users to follow, thus improving navigation and user experience.

Segmented Controls for Difficulty Levels: Below the navigation bar are segmented controls for "Easy," "Medium," and "Hard," which allow users to filter content based on the difficulty level. This improves user experience by making it easy to find content that matches the user's skill level.

Categorization Tabs: The tabs "All," "Easy," "Medium," and "Hard" above the content list are for filtering the list. This helps users to quickly switch between different views according to their preferences or needs.

6.4 User Profile Page



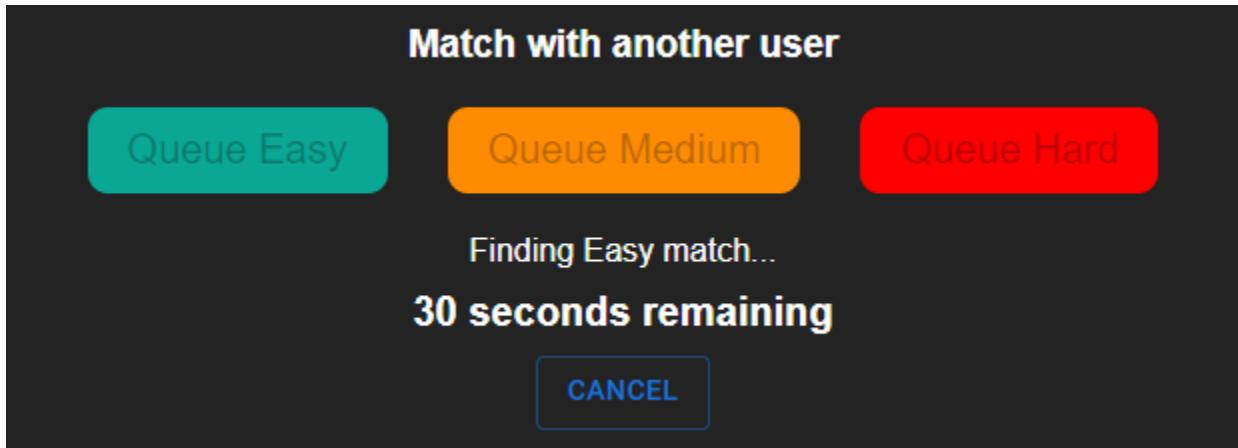
Our user profile page serves three functions: changing of user password, deletion of user account and the tracking of their history. In this section, we discuss the design decisions regarding the profile page.

Contrasting Buttons: The "Change Password" and "Deregister" buttons stand out due to their contrasting colors against the background, making it clear where to click to perform these actions.

Warning Text: For the "Delete Account" section, there is a warning in red, "Warning: Account cannot be recovered!" This is a critical UX element as it clearly communicates the consequences of the action, ensuring the user is informed before making a decision.

Two Tables To Track Their History: For Answered Questions, we track any question the user has attempted in the past regardless of whether they had successfully solved the question. Meanwhile, the Solved Questions table only tracks the number of questions that the user has solved successfully, meaning that they have passed all the test cases for the question. The idea behind having a split table as such is to ensure that the user will have a kept history of questions they have attempted as well as to see their progress on the number of solved questions they have. This ties in with our history service where so long as the button run is pressed, it will end up stored in the history of every user in the room. The tables which will be updated will depend on whether they pass or fail the test cases.

6.5 Queuing State

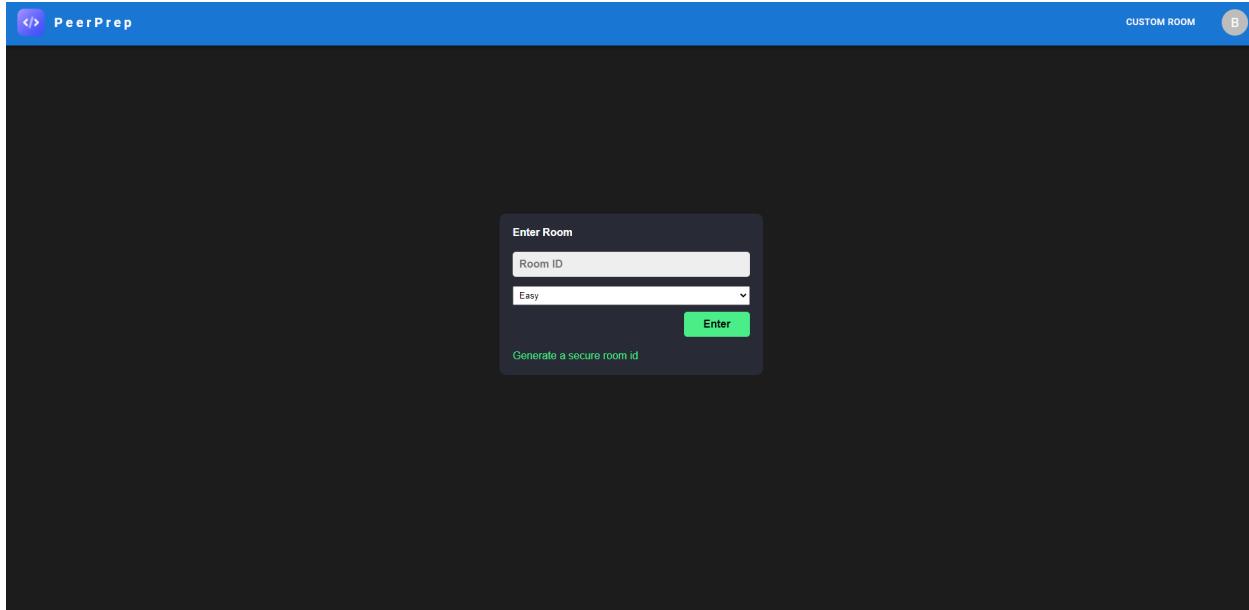


From the home page, the user is able to queue to match with another user to enter a problem-solving room. The user may select one of three buttons to enter the queue; “Queue Easy”, “Queue Medium” and “Queue Hard”, all of which will place the user in the described queue.

Queue Timer: After a user has clicked any queue button, a dynamic timer component will appear, indicating the time remaining before the user is ejected from the queue. This component lets the user know that they won’t be queueing indefinitely, and also indicates that the queueing is happening behind the scenes.

Disabled Queue Button: Once a user enters the queue, the other queue buttons are disabled. This serves a dual purpose: to ensure that a user cannot enter two queues at the same time, and to remind the user that they are currently in a queue.

6.6 Custom Room Portal



From the home page, the user is able to enter the custom room portal by clicking on the “CUSTOM ROOM” button on the navigation bar. The custom room interface allows a user to create a room with a custom room ID and question difficulty. Below are our design decisions for the UI of this page.

Centralized Form: The form is centered on the page, which is a common practice to draw the user's focus directly to the action they need to perform. It makes the form the most prominent feature on the page.

Custom Room ID Field: The user can choose a custom valid string as their room ID. This room ID can then be shared to other users, who can then join the room with the ID.

Secure Room ID Generator: The user has the option to generate a secure room ID using the button on the bottom left of the form. This uses the UUIDv4 library to generate a secure UUID to be used as the room ID. This option improves UX by allowing the user to choose between custom and generated room IDs.

6.7 Programming Room

The screenshot shows a programming room interface. At the top, there's a bar with a 'BT' button labeled 'Big Tester', and three buttons: 'Change Question', 'Copy Room ID', and 'Leave Room'. The main area has a dark background. On the left, there's a section titled 'Find the Width of Columns of a Grid' with a 'Easy' difficulty level. It contains text about a 0-indexed matrix and examples. On the right, there's a code editor with a 'Language' dropdown set to 'Python'. The code is:

```
1 class Solution(object):
2     def findColumnWidth(self, grid):
3         """
4             :type grid: List[List[int]]
5             :rtype: List[int]
6         """
7
```

Below the code editor is a 'Console ^' button and a 'Run' button. At the bottom, there are buttons for 'Chat', 'Video', and 'AI Assistant'.

The user is able to enter the pair programming room via either the custom room portal or by queuing (and matching) with another user. The programming room has many features which can be separated into four distinct groups:

- 1) Room Manipulation
- 2) Communication Tools
- 3) Interactive Development Environment (IDE)
- 4) General Features

6.7.1 Room Manipulation

The room manipulation group consists of three features; “Change Question”, “Copy Room ID” and “Leave Room”, all of which can be located in the bar at the top of the page. We discuss the various design decisions made in the implementation of these features.

Functional Grouping: We group room manipulation features together to ensure that UX is smooth, especially when a user wants to perform common actions such as copying room ID. Furthermore, the “Leave Room” feature is colored brightly, indicating the importance of the button.

Quick Access to Room ID: We wanted to make the room ID easily available to users for two reasons; to enable easy addition of new users to the room (via sharing the ID with other users), and to enable users to rejoin the room in the situation where they have left the room accidentally.

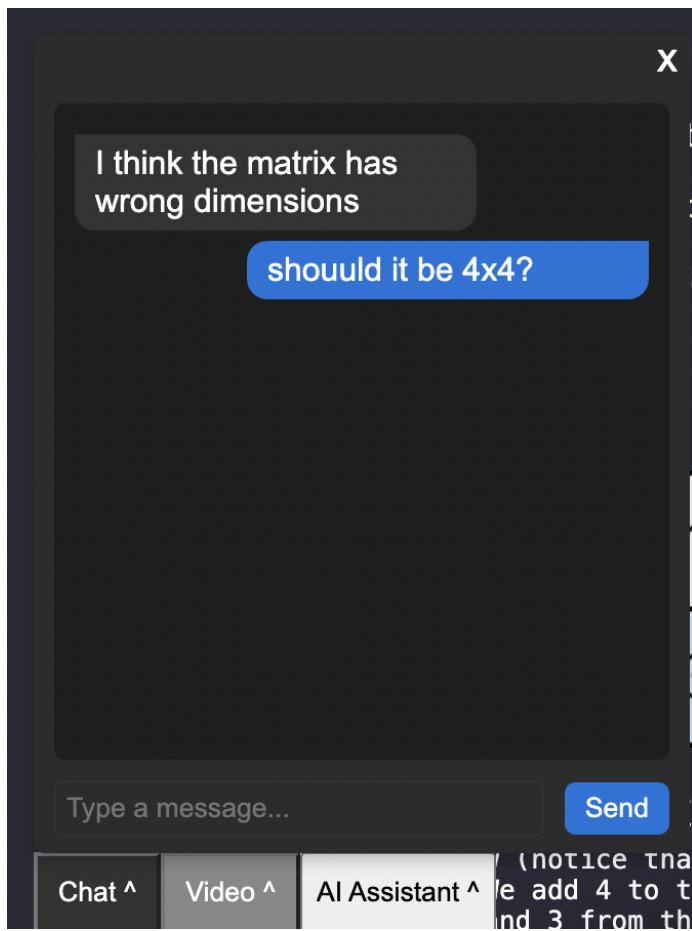
Question Changing: The question changing button sends a request to the other user (if there is another user in the room) to change the question to some new question of a selectable difficulty. This functionality is useful for when a question has been completed successfully, and user(s) want to choose a new question.

6.7.2 Communication Tools

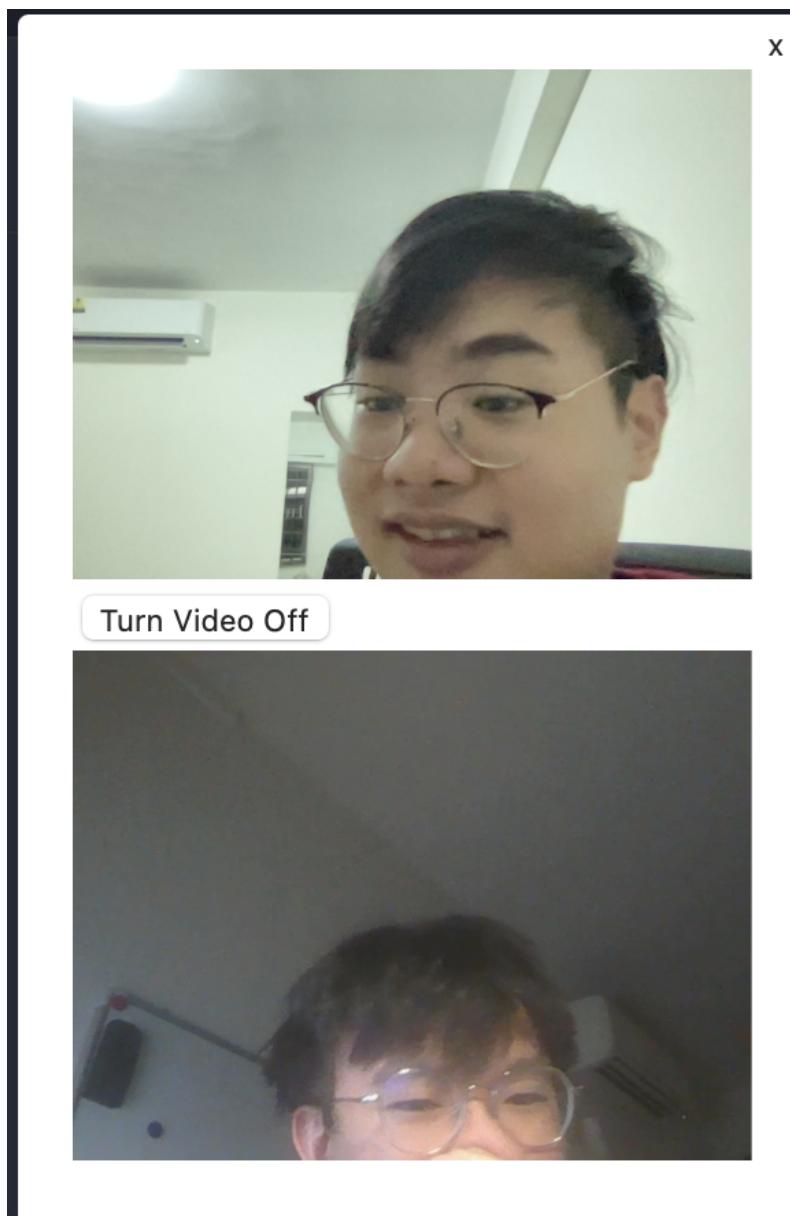
At the bottom-left of the page are three pop-up components: chat, video and AI assistant. We discuss why these are useful UI elements below.

Quick Communication Access: The communication components can be easily accessed by clicking on either (or both) component(s). This design allows users to easily communicate with each other. Furthermore, users are able to navigate between text chat and video interface quickly with this design.

Chat Box:



Video calling feature:



AI Assistant: The AI assistant allows the user to communicate with OpenAI's ChatGPT API. The user can send queries via this chat system, similar to ChatGPT's actual functionality. Having the assistant grouped together with the other communication tools achieves the goal of grouping components with similar functionality together.

can you give me the python code to return the sum of elements in a list?

System:

Sure, here's a simple Python code snippet that sums up all the elements in a list using the built-in `sum()` function:

```
def sum_of_elements(my_list):
    return sum(my_list)

# test the function
print(sum_of_elements([1, 2, 3, 4, 5])) # Output
```

In the code above, `my_list` is an input list of numbers. The `sum()` function returns the sum of all elements in the list. The `sum_of_elements` function thus returns the sum of all elements in the input list.

Type your message...

Send

6.7.3 Interactive Development Environment (IDE)

The right side of the page houses the IDE. Both users can interact with the IDE and see the other user's updates in real time. We chose to house the IDE on the right to emulate LeetCode, another popular programming website. We also gave the IDE ample space by giving it the entire right side of the page to ensure that there was enough space in the IDE for questions that need more code.

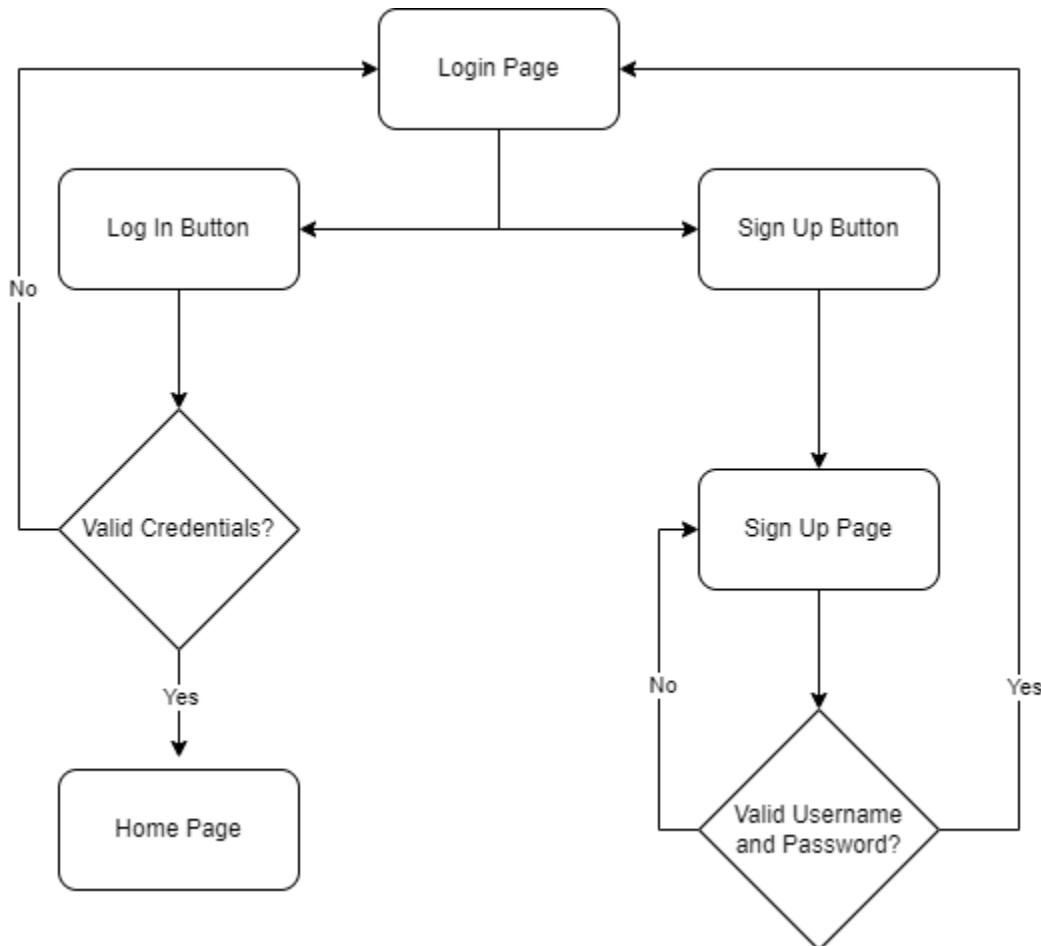
6.7.4 General Features

Our main other UI component of the room is the user icons on the top-left of the page. The icons indicate which users are in the room at the moment. This UI helps to show the number of users in the room at the moment. It also alerts users if another user leaves or joins the room.

7. UX

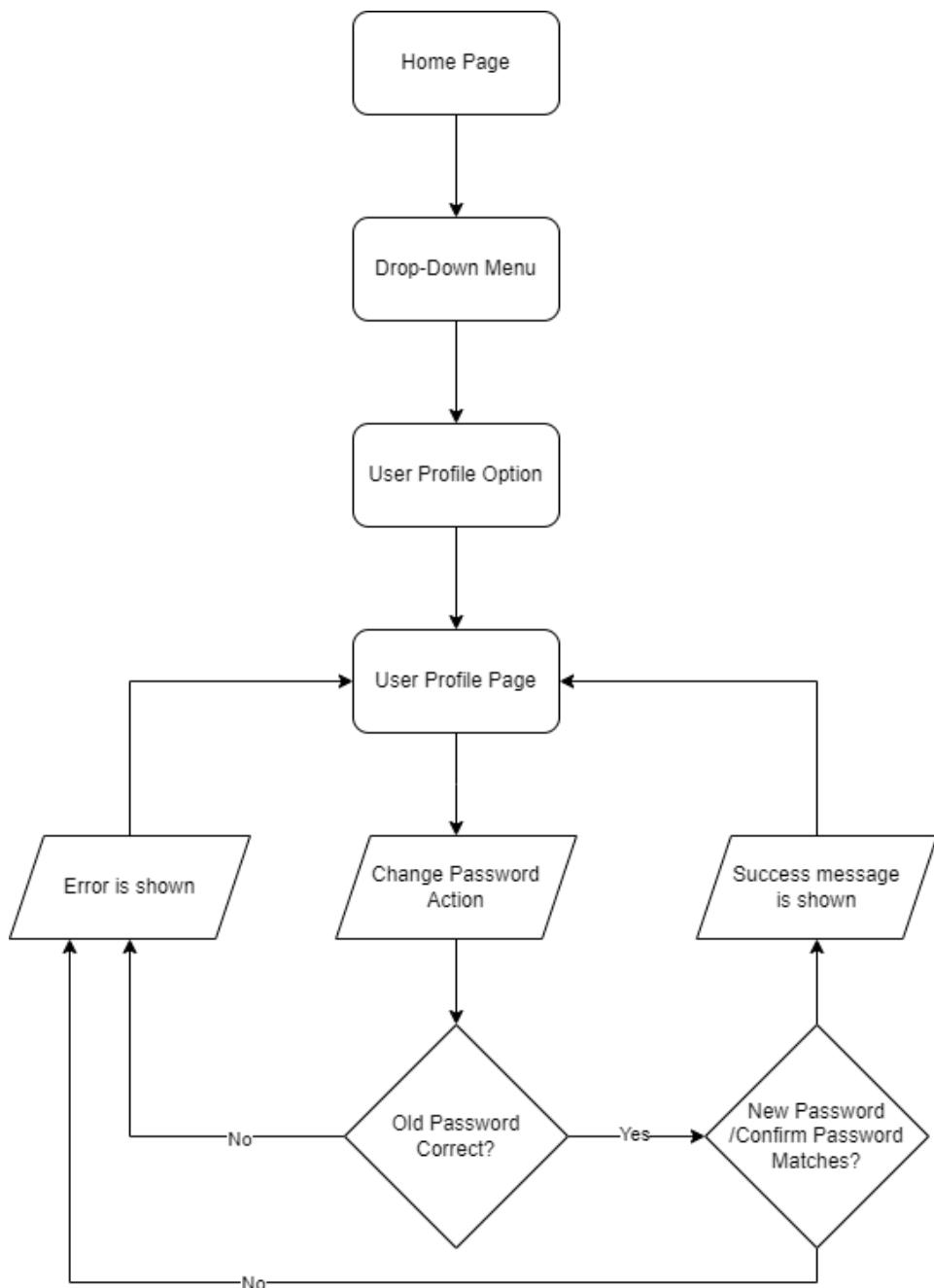
7.1 Login/Sign Up

Our login page allows a user to navigate to either the sign-up page or the home page, depending on the components they interact with. The diagram below describes our user flow.

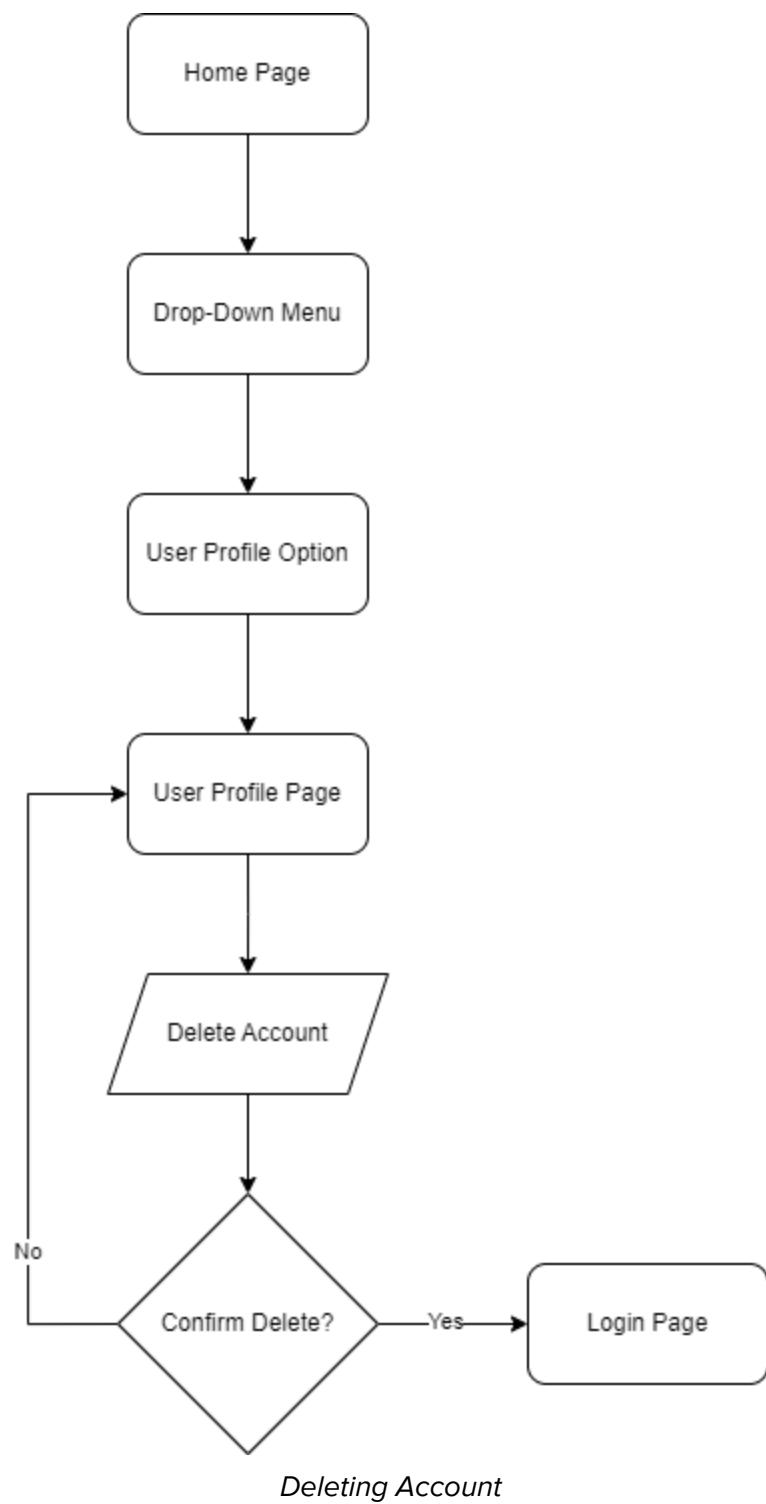


7.2 User Profile Options

The user can perform two options with regards to their profile; changing their password or deactivating their account. Profile options can be accessed via a drop-down menu in the navigation bar. The diagrams below describe how a user can perform either of profile-related actions.



Password Changing



7.3 Matching

The user performs matching through one of the three options: “Queue Easy”, “Queue Medium” and “Queue Hard”. The diagram below displays how the matching is handled.

When a user attempts to join an “Easy” queue, it will first check whether somebody is already queuing for the “Easy” complexity. If there is someone queuing, then he will match with this person. If not, he will wait for 30 seconds for someone to match with him while staying in the queue. Within these 30 seconds, if somebody else joins the “Easy” queue, he will be matched, else, he will be removed from the queue.

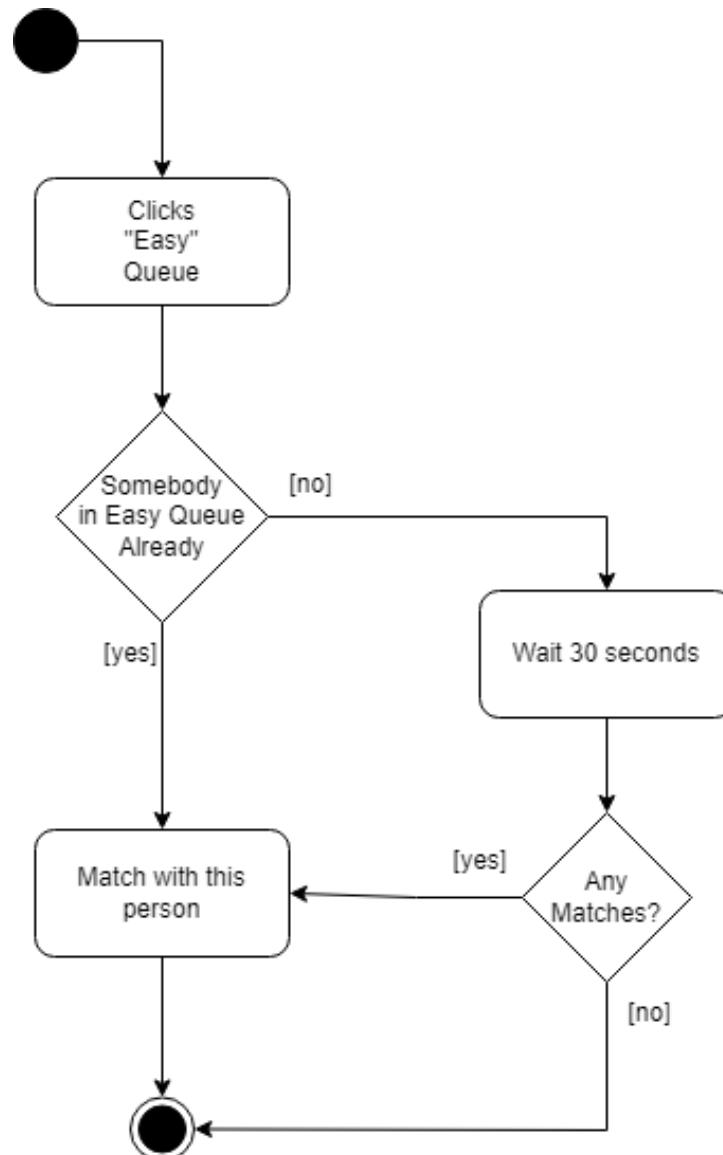


Diagram that shows a sample of “Easy” queue handling

8. Authentication & Security

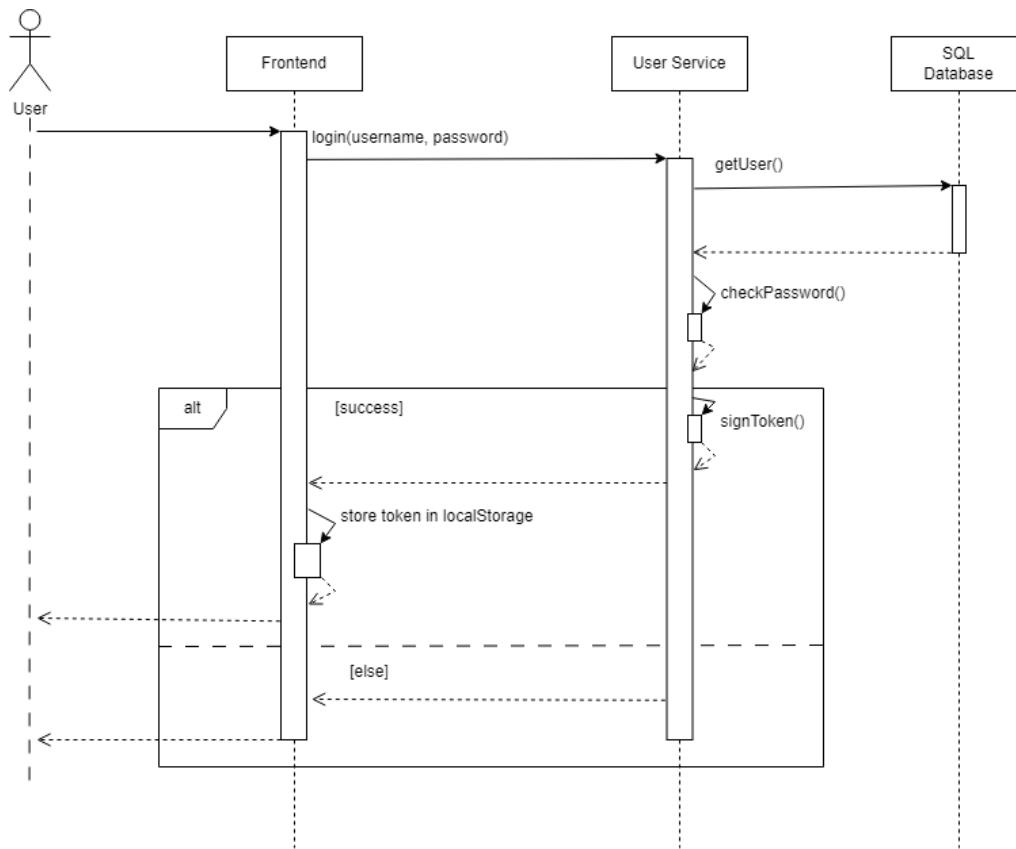
8.1 Process of authentication

Our authentication procedure begins when a user enters their username and password into a login form on the frontend of our application. The frontend then transmits this data by sending a request to our user service. The user service, upon receiving the request, proceeds to access our SQL database to retrieve the hashed password associated with the user's provided username.

Once the hashed password is obtained, the user service proceeds to hash the password sent from the frontend (bcrypt). Subsequently, it compares the newly hashed password with the retrieved hashed password from the database. If there is a match, indicating that the entered password is correct, the user service generates a JsonWebToken (JWT) to signify successful authentication. This JWT is then sent back to the frontend, where it is stored in the browser's local storage.

At this point, the user is authenticated, and they gain access to [authenticated routes](#) within the application.

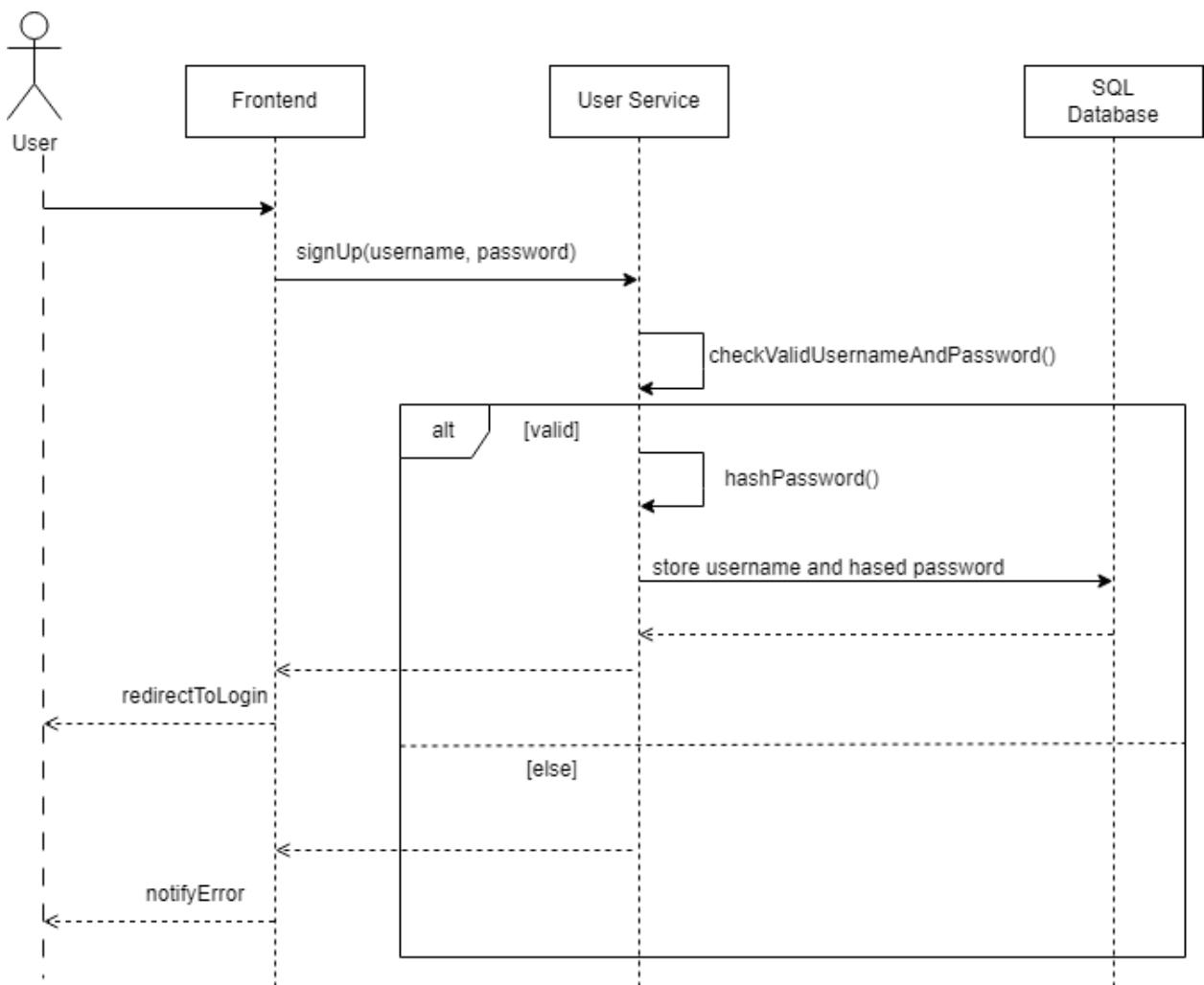
Suppose the login fails, then the frontend will notify the user about his incorrect username or password, without receiving / storing any JWT token.



Sequence diagram showing login flow

8.2 Process of Registration

A user fills in a registration form in the frontend, providing his / her desired username and password. The frontend then sends a request to the user service, which checks whether it is a valid pair of username and password. It makes sure that there is no duplicate username, and password is of minimum length 8. If both are valid, it hashes the password with bcrypt, and saves the username and hashed password to the database. The user is then redirected to the login page where he can [login](#) with his newly registered username and password. If not, the user will be notified of the registration error.



Sequence diagram showing registration flow

8.3 Bcrypt

According to our research, bcrypt is specifically designed for securely hashing passwords, by incorporating techniques such as salting and multiple rounds of hashing. Hence, it is extremely computationally expensive and time consuming for attackers to crack hashed passwords using techniques like brute force or dictionary attacks.

8.4 JsonWebToken (JWT)

JWTs provide a level of data integrity and confidentiality, and the digital signature ensures that the token hasn't been tampered with. This way, it is extremely difficult for users to tamper with their JWT token in an attempt to gain privileged roles such as the manager role, or pretend to be a signed in user when he / she is not.

8.5 Securing REST Endpoints

Our JWT Token is signed with a secret key, with variables “username” and “isManager” attached to it. The JWT Token makes it difficult for users to forge their identities by either changing their username or manager status.

```
const token = jwt.sign(
  {
    username: dbUser.username,
    isManager: dbUser.isManager,
  },
  process.env.JWT_SECRET_KEY,
);
```

8.5.1 User Service

Every request sent to certain routes goes through a middleware that authenticates the JWT token sent by the user from the frontend. If the request does not have a header that attaches the token, the request will be rejected and the controller will not be invoked. This authenticateJwt middleware also checks the signature of the JWT token to make sure it has not been tampered with.

```
10 userRouter.get("/username", [authenticateJwt], userController.getUsername);
11
12 userRouter.post("/register", validateUser(), userController.createUser);
13
14 userRouter.post("/login", userController.loginUser);
15
16 userRouter.put(
17   "/update/password",
18   [authenticateJwt],
19   validateNewPassword(),
20   userController.updatePassword
21 );
22
23 userRouter.delete("/deregister", [authenticateJwt], userController.deleteUser);
```

8.5.2 Question Service

For question service, in addition to checking for a valid JWT token that denotes that a user is signed in, we have additional middleware that checks their manager status. It is impossible for them to forge this status because it is [attached with the JWT token](#). If they are not a manager, their request will be refused and the controller will not be executed.

```
7 // Fetch a question by id
8 questionRouter.get('/questions/:id', [authenticateJwt], questionController.getQuestionById);
9
10 // Fetch all questions
11 questionRouter.get('/', [authenticateJwt], questionController.getAllQuestions);
12
13 // Add a question
14 questionRouter.post('/', [authenticateJwt], [authenticateManager], validateQuestion(), questionController.addQuestion);
15
16 // Update a question
17 questionRouter.put('/', [authenticateJwt], [authenticateManager], validateUpdateQuestion(), questionController.updateQuestion);
18
19 // delete questions
20 questionRouter.delete('/', [authenticateJwt], [authenticateManager], questionController.deleteQuestion);
21
22 // Get random question by complexity
23 questionRouter.get('/random/:complexity', [authenticateJwt], questionController.getRandomQuestion);
```

8.5.3 Collaboration Service

For the collaboration service, the routes are also protected as shown in this diagram below.

```
6 // Adds a new room
7 collaborationRouter.post('/', [authenticateJwt], collaborationController.addRoom);
8
9 // Gets a room by id
0 collaborationRouter.get('/:roomId', [authenticateJwt], collaborationController.getRoomById)
1
2 // Update room question
3 collaborationRouter.put('/room-question', [authenticateJwt], collaborationController.updateRoomQuestion);
```

8.5.4 Code Execution Service

Only users who are authenticated can execute and test their code.

```
6 // Fetch a question by id
7 codeExecRouter.post('/execute-all', [authenticateJwt], codeExecController.runAllTestCases);
8
```

8.6 Securing Websockets

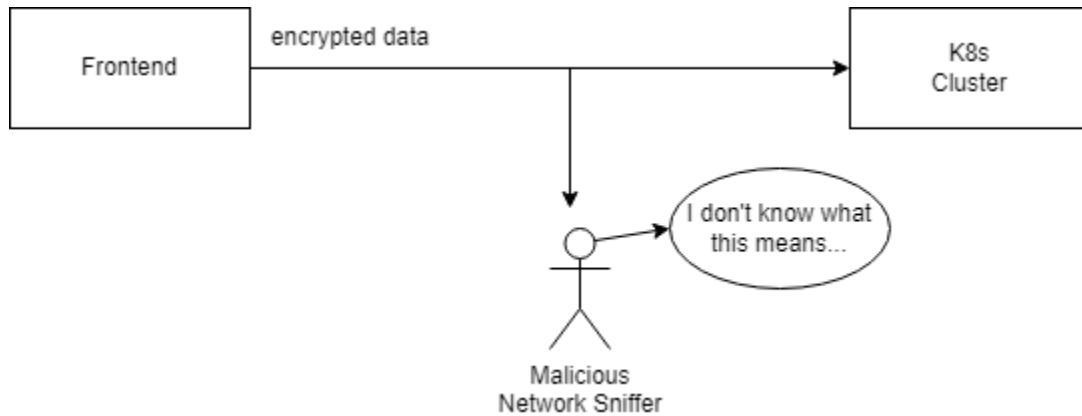
Our services that make use of websockets ensure that they only allow connections from authenticated users. This process requires the user to attach their JWT token to the header when establishing a connection.

The following image shows our usage of middlewares to authenticate a JWT token from the handshake process between the user and server. If the JWT token is invalid, the handshake will fail and a connection will fail to establish.

```
50  io.use((socket, next) => {
51    const token = socket.handshake.query.token;
52
53    jwt.verify(token, process.env.JWT_SECRET_KEY, (err, decoded) => {
54      if (err) {
55        return next(new Error("Authentication Error"));
56      }
57
58      socket.username = decoded.username;
59      next();
60    });
61  });
62});
```

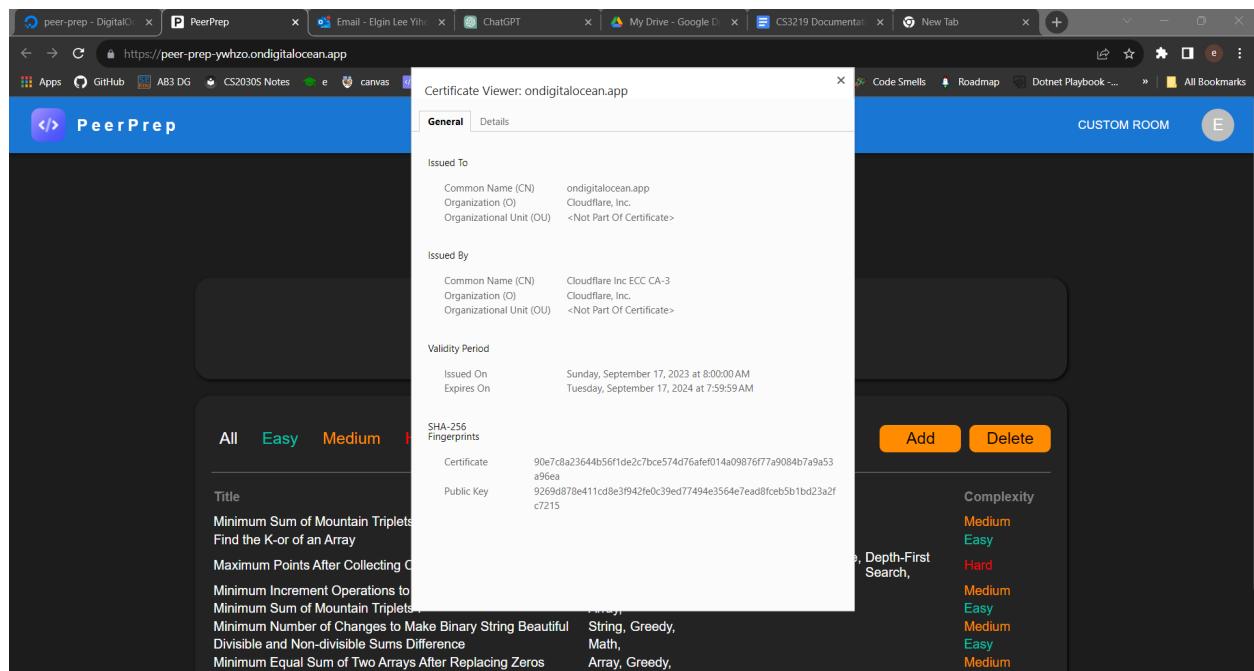
8.7 Security of traffic through HTTPS

The traffic between our frontend and backend (Kubernetes cluster) is supported by HTTPS. This means that data transmitted between the frontend and the server is encrypted. This encryption is achieved with protocols like TLS, which ensures that the data exchanged can only be deciphered by the intended recipient.



8.7.1 Frontend HTTPS

As the frontend is deployed on DigitalOcean Applications, it provides HTTPS by default. This TLS certificate is issued by Let's Encrypt, a widely recognized Certificate Authority (CA).



8.7.2 Backend HTTPS

By default, the Kubernetes load balancer is HTTP protocol instead of HTTPS.

We need to manually request for a TLS certificate from Let's Encrypt. Unfortunately, Let's Encrypt does not issue TLS certificates to IP addresses, but only to domain names. Hence, our group purchased a domain name from NameCheap, and configured the domain to point to our load balancer's IP.

Domain Name Configuration

Type	Host	Value	TTL	
A Record	@	188.166.196.206	Automatic	
+ ADD NEW RECORD				

DigitalOcean Load Balancer

LOAD BALANCERS (1)				
a1a74b2f4806a497b8118b9c8f0c0...	188.166.196.206	1/1	—	

Using kubectl and defined cert-manager-issuer.yaml, we configure the certificate issuer to be from Let's Encrypt.

```
1  apiVersion: cert-manager.io/v1
2  kind: ClusterIssuer
3  metadata:
4    name: letsencrypt-prod
5    namespace: peerprep
6  spec:
7    acme:
8      email: elginlee_2000@hotmail.com
9      server: https://acme-v02.api.letsencrypt.org/directory
10     privateKeySecretRef:
11       name: letsencrypt-prod-private-key
12     solvers:
13       # Use the HTTP-01 challenge provider
14       - http01:
15         ingress:
16           class: nginx
```

Then, within the ingress.yaml file, we request for a certificate.

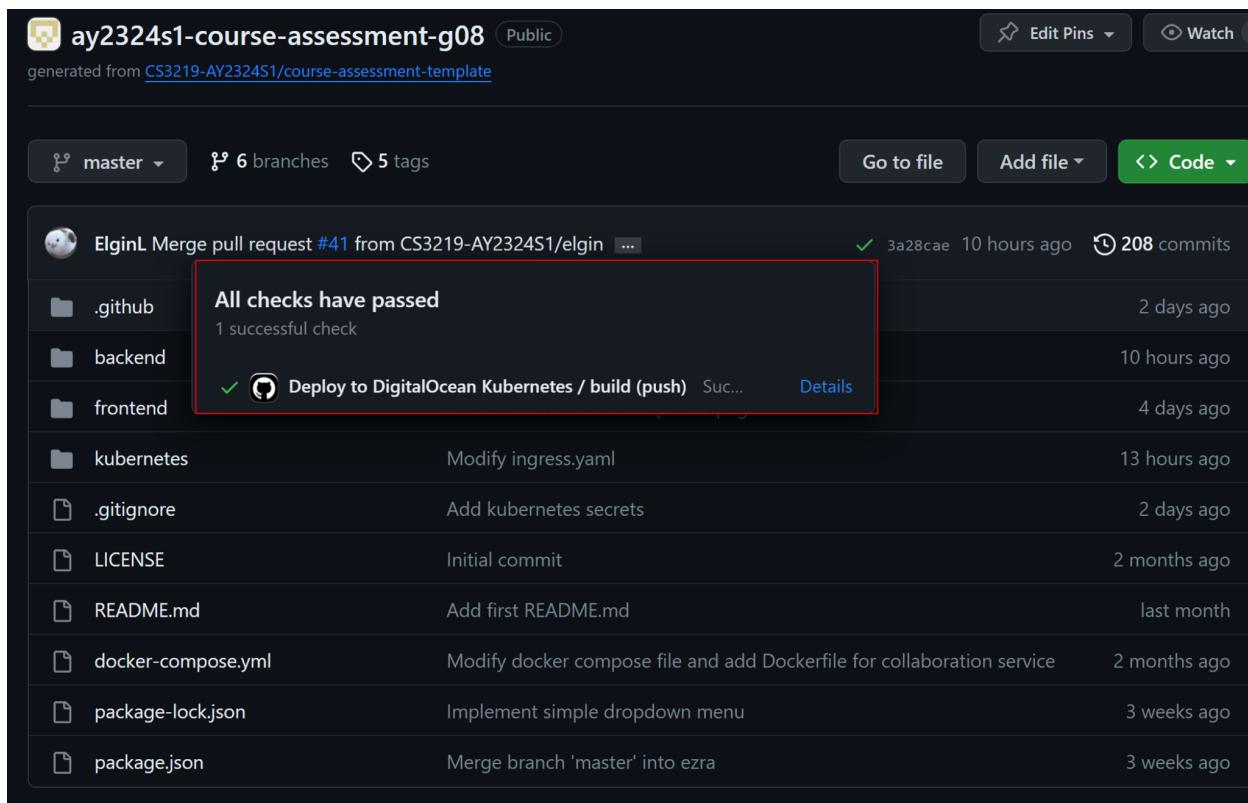
```
kubernetes > ➜ ingress.yaml
 1  apiVersion: networking.k8s.io/v1
 2  kind: Ingress
 3  metadata:
 4    name: peerprep-ingress
 5    namespace: peerprep
 6    annotations:
 7      nginx.ingress.kubernetes.io/proxy-read-timeout: "3600"
 8      nginx.ingress.kubernetes.io/proxy-send-timeout: "3600"
 9      nginx.ingress.kubernetes.io/websocket-services: "collaboration-service,matching-service,communication-service"
10      cert-manager.io/cluster-issuer: letsencrypt-prod
11
12  spec:
13    ingressClassName: "nginx"
14    tls:
15      - hosts:
16        - peerprepbackend.com
17        secretName: peerprep-tls
18    rules:
19      - host: "peerprepbackend.com"
20        http:
21          paths:
22            - path: /question-service
23              pathType: Prefix
```

This process provides our backend with HTTPS protocol, with our domain name specified.

9. Continuous Deployment

9.1 Kubernetes Cluster Automated Deployment

To improve the productivity of our team, we have set up a continuous deployment workflow with the help of Github Actions. We have a series of steps defined on push to our master branch on our Github repository, that helps to deploy and update our Kubernetes Cluster with the updated code.



The screenshot shows a GitHub repository page for 'ay2324s1-course-assessment-g08'. The repository has 6 branches and 5 tags. A pull request from 'ElginL' has been merged, indicated by a green checkmark and the commit hash '3a28cae' with a timestamp of '10 hours ago'. The commit message is 'Merge pull request #41 from CS3219-AY2324S1/elgin ...'. A red box highlights the 'Deploy to DigitalOcean Kubernetes / build (push)' status, which is also marked with a green checkmark and the same timestamp. The commit details show 208 commits in total. Below the commit list, there is a list of files: .github, backend, frontend, kubernetes, .gitignore, LICENSE, README.md, docker-compose.yml, package-lock.json, and package.json. The 'kubernetes' file is listed with the action 'Modify ingress.yaml'. The 'package.json' file is listed with the action 'Merge branch 'master' into ezra'.

File	Action	Time
.github		2 days ago
backend		10 hours ago
frontend		4 days ago
kubernetes	Modify ingress.yaml	13 hours ago
.gitignore	Add kubernetes secrets	2 days ago
LICENSE	Initial commit	2 months ago
README.md	Add first README.md	last month
docker-compose.yml	Modify docker compose file and add Dockerfile for collaboration service	2 months ago
package-lock.json	Implement simple dropdown menu	3 weeks ago
package.json	Merge branch 'master' into ezra	3 weeks ago

Image showing updates made to the kubernetes cluster through Github Actions

✓ Change name for workflow files Deploy collaboration service to DigitalOcean Kubernetes #1: Commit a309727 pushed by ElginL	play	yesterday ⌚ 1m 20s	...
✓ Change name for workflow files Deploy user service to DigitalOcean Kubernetes #1: Commit a309727 pushed by ElginL	play	yesterday ⌚ 1m 39s	...
✓ Change name for workflow files Deploy code exec service to DigitalOcean Kubernetes #1: Commit a309727 pushed by ElginL	play	yesterday ⌚ 1m 26s	...
✓ Change name for workflow files Deploy communication service to DigitalOcean Kubernetes #1: Commit a309727 pushed by ElginL	play	yesterday ⌚ 1m 26s	...
✓ Change name for workflow files Deploy matching service to DigitalOcean Kubernetes #1: Commit a309727 pushed by ElginL	play	yesterday ⌚ 1m 25s	...
✓ Change name for workflow files Deploy question service to DigitalOcean Kubernetes #1: Commit a309727 pushed by ElginL	play	yesterday ⌚ 1m 31s	...

Workflow Description

build	succeeded 47 minutes ago in 1m 6s	Search logs	...
> ✓ Set up job	1s		
> ✓ Checkout master	1s		
> ✓ Install doctl	2s		
> ✓ Update Docker settings	1s		
> ✓ Build code execution service container image	19s		
> ✓ Log in to DigitalOcean Container Registry with short-lived credentials	1s		
> ✓ Push code-exec-service image to DigitalOcean Container Registry	31s		
> ✓ Update code execution deployment file	0s		
> ✓ Save DigitalOcean kubeconfig with short-lived credentials	0s		
> ✓ Deploy code-exec-deployment DigitalOcean Kubernetes	4s		
> ✓ Configure kubernetes ingress	2s		
> ✓ Post Checkout master	0s		
> ✓ Complete job	0s		

Sample workflow for code execution service

The workflow begins by building a new container image for the service that we have. It then pushes the image built to [Digital Ocean's container registry](#). It then updates the kubernetes deployment yaml file defined in our Github repository with the new image name, and runs the kubectl apply command to configure our cluster. Lastly, it runs the kubectl apply command to update our ingress.yaml file to update the cluster's ingress.

Without this automated deployment workflow, it will be extremely time consuming and error prone when deploying our backend because we have a series of docker and kubectl commands to run. This workflow helps to strengthen our application's correctness and saves our time as well which could be better spent on fixing bugs / implementing new features.

9.1.1 Snapshot of digital ocean container registry

The screenshot shows the DigitalOcean Container Registry interface for the repository 'peerprep'. At the top, there is a navigation bar with tabs for 'Repositories' (which is selected), 'History', and 'Settings'. On the right, there is a blue 'Actions' button. Below the navigation bar, there are two main sections: 'STORAGE USAGE' and 'GARBAGE COLLECTION'. The 'STORAGE USAGE' section shows a progress bar at 1% completion, indicating 984.27 MB / 100 GB used. The 'GARBAGE COLLECTION' section shows 455 MB of unused registry data and a link to 'Empty garbage'. The main area displays a list of services and their corresponding Docker images:

Service	Image Count	Recent Image Tag	Last Updated	Actions
code-exec-service	11 Images	3a28cae	10 hours ago	...
collaboration-service	12 Images	3a28cae	10 hours ago	...
communication-service	12 Images	3a28cae	10 hours ago	...
matching-service	12 Images	3a28cae	10 hours ago	...
question-service	12 Images	3a28cae	10 hours ago	...
user-service	13 Images	3a28cae	10 hours ago	...

9.1.2 Efficiency of continuous deployment

We create dedicated workflow files for each of our microservices, with each file designed to monitor specific changes relevant to that particular microservice. This approach ensures that when changes are made and pushed to the master branch, only the affected microservices are rebuilt and deployed. This targeted workflow minimizes unnecessary updates and deployment for other microservices, making our process highly efficient and resource-friendly.

9.2 Frontend Automated Deployment

The process of frontend automated deployment is done primarily with the help of Digital Ocean's Apps. We have installed a 3rd party application on our Github repository. It observes for code changes in our Github repository and builds the application whenever there is a change to the frontend folder, which is configured by us.

Apps

[Create App](#)

Name	Plan	Last Deployed	Live App URL
 peer-prep	Pro	about 11 hours ago	peer-prep-ywhzo.ondigitalocean.app ↗

The build script is simple and simplifies the deployment process.

```
1 alerts:
2 - rule: DEPLOYMENT_FAILED
3 - rule: DOMAIN_FAILED
4 features:
5 - buildpack-stack=ubuntu-22
6 ingress:
7   rules:
8     - component:
9       |   name: s1-course-assessment-g08
10      |   match:
11      |   |   path:
12      |   |   prefix: /
13     name: peer-prep
14   region: sgp
15 services:
16 - environment_slug: node-js
17 envs:
18 - key: REACT_APP_QUESTION_SERVICE_URL
19   scope: RUN_AND_BUILD_TIME
20   value: https://peerprefbackend.com
21 - key: REACT_APP_USER_SERVICE_URL
22   scope: RUN_AND_BUILD_TIME
23   value: https://peerprefbackend.com
24 - key: REACT_APP_MATCHING_SERVICE_URL
25   scope: RUN_AND_BUILD_TIME
26   value: https://peerprefbackend.com
27 - key: REACT_APP_COLLABORATION_SERVICE_URL
28   scope: RUN_AND_BUILD_TIME
29   value: https://peerprefbackend.com
30 - key: REACT_APP_COMMUNICATION_SERVICE_URL
31   scope: RUN_AND_BUILD_TIME
32   value: https://peerprefbackend.com
33 - key: REACT_APP_CODE_EXEC_SERVICE_URL
34   scope: RUN_AND_BUILD_TIME
35   value: https://peerprefbackend.com
36 github:
37   branch: master
38   deploy_on_push: true
39   repo: CS3219-AY2324S1/ay2324s1-course-assessment-g08
40 http_port: 3000
41 instance_count: 2
42 instance_size_slug: professional-xs
43 name: s1-course-assessment-g08
44 source_dir: /frontend
```

Defining environment variables for our React application to use in production

Specifies where the code we want to deploy is located

10. Code Organization

Our code is structured such that it splits the frontend and backend code logic. This helps to improve cohesion as we have a folder that deals only with the frontend code, and another folder that deals with the backend.

ElginL Merge pull request #41 from CS3219-AY2324S1/elgin	... 3a28cae 11 hours ago	208 commits
.github	Remove unused logs and configure workflow.yml	2 days ago
backend	Fix cors origin	11 hours ago
frontend	Add username render in profile page	4 days ago
kubernetes	Modify ingress.yaml	14 hours ago
.gitignore	Add kubernetes secrets	2 days ago
LICENSE	Initial commit	2 months ago
README.md	Add first README.md	last month
docker-compose.yml	Modify docker compose file and add Dockerfile for collaboration service	2 months ago
package-lock.json	Implement simple dropdown menu	3 weeks ago
package.json	Merge branch 'master' into ezra	3 weeks ago

Within the backend folder, we further split our code into different modules, each of which represents a separate service. Again, it improves cohesion and each subgroup need not intersect with each other in terms of code changes to their own microservice.

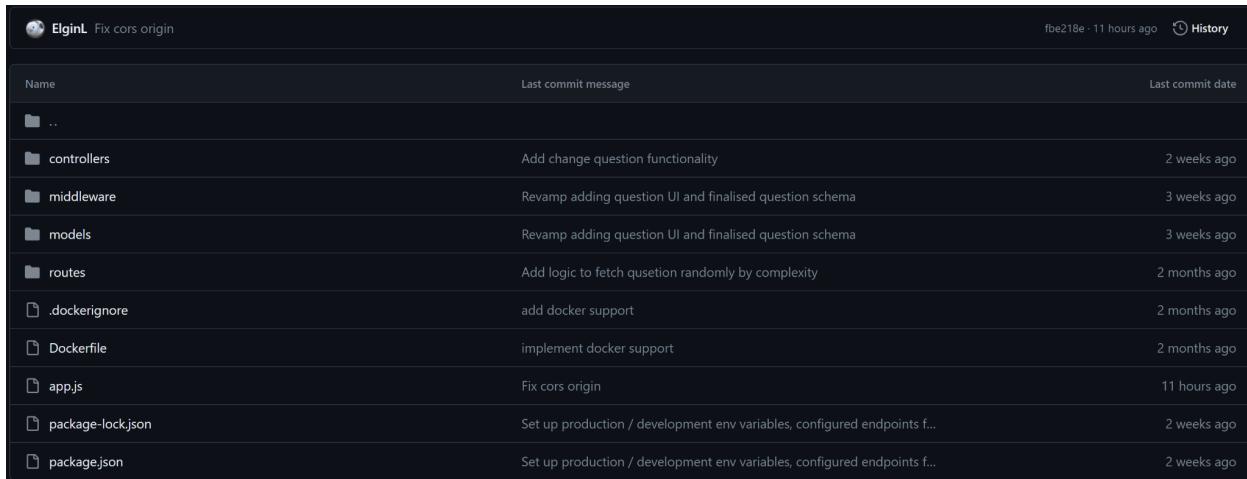
ElginL Fix cors origin	fbe218e - 11 hours ago	History
Name	Last commit message	Last commit date
..		
code-exec-service	Fix cors origin	11 hours ago
collaboration-service	Modify ingress.yaml	14 hours ago
communication-service	Fix cors origin	11 hours ago
matching-service	Modify ingress.yaml	14 hours ago
question-service	Fix cors origin	11 hours ago
user-service	Fix cors origin	11 hours ago
.gitignore	Extract code execution to a service	last week

An alternative code structure that could have been possible is to create separate Github repositories for each microservice, but this wasn't done because the course requires us to use a central repository. Furthermore, the difference is not as big since we are working in small teams and need not restrict another team's rights to a certain service.

10.1 MVC architectural pattern

Our team made use of MVC (Model-View-Controller) to organize our services that set up REST endpoints. It enforces separation of concern to make it simpler to maintain the application, as changes in one component rarely affects the other. It improves reusability as well.

In the image below, there exists controllers, models, and routes directory.



Name	Last commit message	Last commit date
...		
controllers	Add change question functionality	2 weeks ago
middleware	Revamp adding question UI and finalised question schema	3 weeks ago
models	Revamp adding question UI and finalised question schema	3 weeks ago
routes	Add logic to fetch question randomly by complexity	2 months ago
.dockerignore	add docker support	2 months ago
Dockerfile	implement docker support	2 months ago
app.js	Fix cors origin	11 hours ago
package-lock.json	Set up production / development env variables, configured endpoints f...	2 weeks ago
package.json	Set up production / development env variables, configured endpoints f...	2 weeks ago

10.1.1 Model

The model defines the schema for the data that will be interacted with. For the case of question service, it will be the [questions schema](#), test case schema [Test Case Schema](#), and [code template schema](#).

10.1.2 View

This is the interface that the users will interact with by sending a request from the frontend, in order to achieve the result that they want.

```
// Fetch a question by id
questionRouter.get('/questions/:id', authenticateJwt, questionController.getQuestionById);

// Fetch all questions
questionRouter.get('/', authenticateJwt, questionController.getAllQuestions);

// Add a question
questionRouter.post('/', authenticateJwt, authenticateManager, validateQuestion(), questionController.addQuestion);

// delete questions
questionRouter.delete('/', authenticateJwt, authenticateManager, questionController.deleteQuestion);

// Get random question by complexity
questionRouter.get('/random/:complexity', authenticateJwt, questionController.getRandomQuestion);
```

10.1.3 Controller

The controller contains the business logic that contains all the logic that is executed after the view routes the request to the controller.

```
const getQuestionById = (req, res, next) => {
    const id = req.params.id;

    Question.findById(id)
        .populate('testCases')
        .populate('codeTemplate')
        .exec()
        .then(question => {
            if (!question) {
                return res.status(404).json({ error: 'Cannot find question' });
            }

            res.status(200).json(question);
        }).catch(err => next(err));
};

const getAllQuestions = (req, res, next) => {
    Question.find()
        .then(questions => res.status(200).json(questions))
        .catch(err => next(err));
};
```

11. ORM framework

11.1 Sequelize

Our team made use of Sequelize as our Object-Relational mapping (ORM) framework to map the SQL database table to object classes. This allows us to work with database records as if they were regular objects. The biggest perks of all is the improved security which is part of our application's NFRs. Since we do not need to write SQL code, there is no way for an SQL injection attack to occur.

11.2 Mongoose

Our team made use of mongoose as our ORM framework for working with MongoDB. It provides a lot of benefits such as validation (rules for data stored in MongoDB), schema definition, and allowing relations, which is [important for our database schema design.](#)

12. Kubernetes Horizontal Pod Autoscaler

In this section, we will be load testing our application and making sure that our horizontal pod autoscaler is working.

To demonstrate this, we will be using **Grafana K6** to perform the load test.

Image showing the initial state of the pods before the load test script is run.

NAME	READY	STATUS	RESTARTS	AGE
pod/code-exec-service-55db988499-xxtwc	1/1	Running	2 (4h49m ago)	41h
pod/collaboration-service-7f8f86fd99-htfrv	1/1	Running	0	2d11h
pod/communication-service-764477995f-6tkhl	1/1	Running	0	2d11h
pod/matching-service-8479fdb46b-kn55c	1/1	Running	0	36h
pod/nginx-ingress-ingress-nginx-controller-6559469f-fz6h4	1/1	Running	0	7d1h
pod/question-service-645cd66844-2q4wx	1/1	Running	0	2d11h
pod/user-service-75bc7f68cf-bxtcw	1/1	Running	0	2d11h

Image showing the initial metrics of CPU usage for each pod

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
horizontalpodautoscaler.autoscaling/code-exec-hpa	Deployment/code-exec-service	1%/80%	1	3	1	7d5h
horizontalpodautoscaler.autoscaling/question-hpa	Deployment/question-service	1%/80%	1	3	1	20d
horizontalpodautoscaler.autoscaling/user-hpa	Deployment/user-service	1%/80%	1	3	1	20d

The load testing script is relatively simple, we will be spinning up 10 virtual users, and over a period of 60 seconds, 800 GET requests per second will be sent to the question service.

```
1 import http from 'k6/http';
2
3 export const options = {
4     vus: 10,           // Number of Virtual Users
5     duration: '60s', // Test duration (e.g., 60 seconds)
6     rps: 800,         // Requests per second
7 };
8
9 export default function () {
10     const url = 'https://peerprepbackend.com/question-service';
11
12     http.get(url);
13 }
14
```


13. API Endpoints

This section highlights all the REST endpoints that can be called on our backend, along with its functionalities and possible responses.

13.1 Question Service

13.1.1 Fetching a question by ID

Endpoint: <http://localhost:3001/question-service/questions/:id>

HTTP Method: GET

Description: Sending a GET request to this endpoint will return a question with the id specified.

Authentication required: Yes

Manager Role required: No

Responses			
Status Code	Status Desc.	Output Description	Sample Response
200	OK	A question with the id specified returned with all fields populated	<pre>{ "_id": "654c9f3d9d77f86a3bf349ce", "title": "Minimum Sum of Mountain Triplets II", "categories": ["Array", "Tree"], "complexity": "Medium", "description": "<p>You are given a 0-indexed array <code>nums</code> of integers.</p>", "codeTemplate": { "_id": "654c9f3d9d77f86a3bf349c6", "templates": { "Python": "class Solution(object):\n def minimumSum(self, nums):\n \n", "Javascript": "var minimumSum = function(nums) {\n \n \n};", "Ruby": "def minimum_sum(nums)\n \n end\n }\n }, "testCases": [{ "_id": "654c9f3d9d77f86a3bf349c8", "input": { "nums": [8,6,1,5,3] }, "output": 9 }] }</pre>

404	NOT FOUND	Error message saying that the question with that id specified cannot be found	{ "error": "Cannot find question" }
401	UNAUTHORIZED	Missing header token, not authenticated to retrieve question	{ "msg": "Authentication failed: missing token" }
401	UNAUTHORIZED	Invalid token attached to header	{ "msg": "Authentication failed: invalid token" }

13.1.2 Fetch all questions

Endpoint: <http://localhost:3001/question-service>

HTTP Method: GET

Description: Sending a GET request to this endpoint will return all questions from the database

Authentication required: Yes

Manager Role required: No

Responses			
Status Code	Status Desc.	Output Description	Sample Response
200	OK	An array of questions, but some fields (codeTemplate, testCases) are not populated with data, rationale here	{ " <u>_id</u> ": "654c9f3d9d77f86a3bf349ce", "title": "Minimum Sum of Mountain Triplets II", "categories": ["Array", "Tree"], "complexity": "Medium", "description": "<p>You are given a 0-indexed array <code>nums</code> of integers.</p>", "codeTemplate": "654c9f3d9d77f86a3bf349c6", "testCases": ["654c9f3d9d77f86a3bf349c8", "654c9f3d9d77f86a3bf349c9", "654c9f3d9d77f86a3bf349ca"] }
401	UNAUTHORIZED	Missing header token, not authenticated to retrieve question	{ "msg": "Authentication failed: missing token" }
401	UNAUTHORIZED	Invalid token attached to header	{ "msg": "Authentication failed: invalid token" }

13.1.3 Add a question

Endpoint: <http://localhost:3001/question-service>

HTTP Method: POST

Description: Sending a POST request to this endpoint will attempt to add a new question to the database

Authentication required: Yes

Manager Role required: Yes

Responses			
Status Code	Status Desc.	Input	Sample Response
201	Created	{ "title": "abcdef", "categories": ["array", "string"], "complexity": "Easy", "description": "String desc", "codeTemplates": [{ "language": "Python", "template": "hello" }], "inputs": [{ "n": 5, "m": 6 }, { "n":5, "m": 1 }], "outputs": [15, -15] }	"Question Saved Successfully"
400	BAD REQUEST	{ "title": "", "categories": ["array", "string"], "complexity": "Easy", "description": "String descr", "codeTemplates": [{ "language": "Python", "template": "hello" }], "inputs": [{ "n": 5, "m": 6 }, { "n":5, "m": 1 }], "outputs": [15, -15] }	{ "msg": "Title is required" }
400	BAD REQUEST	{ "title": "abcd", "categories": ["array", "string"], "complexity": "Easy", "description": "" }	{ "msg": "Description is required" }

		<pre> "codeTemplates": [{ "language": "Python", "template": "hello" }], "inputs": [{ "n": 5, "m": 6 }, { "n":5, "m": 1 }], "outputs": [15, -15] } </pre>	
400	BAD REQUEST	<pre> { "title": "abcde", "categories": ["Array"], "complexity": "Easy", "description": "desdc", "codeTemplates": [{ "language": "Python", "template": "hello" }], "inputs": [{ "n": 5, "m": 6 }], "outputs": [15, -15] } </pre>	<pre> { "msg": "Inputs and Outputs must have the same length" } </pre>
400	BAD REQUEST	Adding another question which has the same title as an existing question in the database	<pre> { "msg": "Question with this title already exists" } </pre>
400	BAD REQUEST	Adding another question which has the same description as an existing question in the database	<pre> { "msg": "Question with this description already exists" } </pre>
401	UNAUTHORIZED	Request sent without a token attached to the HTTP header	<pre> { "msg": "Authentication failed: missing token" } </pre>
401	UNAUTHORIZED	Request sent with token attached to HTTP header, but don't have a manager status	<pre> { "msg": "Authentication failed: not a manager" } </pre>
401	UNAUTHORIZED	Invalid token attached to header	<pre> { "msg": "Authentication failed: invalid token" } </pre>

13.1.4 Update a question

Endpoint: <http://localhost:3001/question-service>

HTTP Method: PUT

Description: Sending a PUT request to this endpoint will attempt to add a new question to the database

Authentication required: Yes

Manager Role required: Yes

Responses			
Status Code	Status Desc.	Input	Sample Response
200	OK	{ "questionId": "654c9f3d9d77f86a3bf349ce", "title": "updated title", "categories": ["Array", "String"], "complexity": "Medium", "description": "updated description", "codeTemplates": [{ "language": "Python", "template": "hello" }], "inputs": [{ "n": 5, "m": 6 }], "outputs": [15] }	"Question updated successfully"
400	BAD REQUEST	A scenario where questionId does not exist in the database { "questionId": "654c9f3d9d77f86a3bf349ce", "title": "abcde", "categories": ["Array"], "complexity": "Easy", "description": "desdc", "codeTemplates": [{ "language": "Python", "template": "hello" }], "inputs": [{ "n": 5, "m": 6 }], "outputs": [15] }	{ "msg": "Unable to find question with the id specified" }

13.1.5 Delete a question

Endpoint: <http://localhost:3001/question-service>

HTTP Method: DELETE

Description: Sending a DELETE request to this endpoint will attempt to delete a question from the database with the specified id

Authentication required: Yes

Manager Role required: Yes

Responses			
Status Code	Status Desc.	Input	Sample Response
204	No Content	{ "ids": ["654c9f3f9d77f86a3bf349f2", "654c9f409d77f86a3bf349fe"] }	"Question updated successfully"
401	UNAUTHORIZED	Request sent without a token attached to the HTTP header	{ "msg": "Authentication failed: missing token" }
401	UNAUTHORIZED	Request sent with token attached to HTTP header, but don't have a manager status	{ "msg": "Authentication failed: not a manager" }
401	UNAUTHORIZED	Invalid token attached to header	{ "msg": "Authentication failed: invalid token" }

13.1.6 Get random question

Endpoint: <http://localhost:3001/question-service/random/:complexity>

HTTP Method: GET

Description: Sending a GET request to this endpoint will get a single random question according to the complexity specified

Authentication required: Yes

Manager Role required: No

Responses			
Status Code	Status Desc.	Output Description	Sample Response
200	OK	A random question according to the complexity specified	{ "_id": "654ca2469d77f86a3bf34d81", "title": "Length of the Longest Valid Substring", "categories": ["Array", "Hash Table", "String", "Sliding Window"], "complexity": "Hard", "description": "<p>You are given a string <code>word</code> and an array of strings <code>forbidden</code>.</p>\n\n <p>A string is called valid... (truncated) "codeTemplate": "654ca2469d77f86a3bf34d7b", "testCases": ["654ca2469d77f86a3bf34d7d", "654ca2469d77f86a3bf34d7e"] }</p>
404	NOT FOUND	Unable to get a random question, because there's no question with the complexity specified	{ "msg": "No questions found" }

401	UNAUTHORIZED	Request sent without a token attached to the HTTP header	{ "msg": "Authentication failed: missing token" }
401	UNAUTHORIZED	Invalid token attached to header	{ "msg": "Authentication failed: invalid token" }

13.2 User Service

13.2.1 Register

Endpoint: <http://localhost:3002/user-service/register>

HTTP Method: POST

Description: Sending a POST request to this endpoint with a username and password will create a new account, stored in the database

Authentication required: No

Manager Role required: No

Responses			
Status Code	Status Desc.	Input	Sample Response
201	CREATED	{ "username": "HelloWorld", "password": "ElginLee" }	{ "msg": "User created successfully" }
400	BAD REQUEST	A scenario where username already exist in the database { "username": "HelloWorld", "password": "ElginLee" }	{ "msg": "Username is already in use" }
400	BAD REQUEST	{ "username": "HelloWorld", "password": "Hi" }	{ "msg": "Username is already in use, Password must be between 8 and 20 characters" }
400	BAD REQUEST	{ "username": "abc", "password": "HelloWorld" }	{ "msg": "Username must be between 5 and 12 characters" }

13.2.2 Login

Endpoint: <http://localhost:3002/user-service/login>

HTTP Method: POST

Description: Sending a POST request to this endpoint with a username and password will attempt a login, to receive a JWT Token

Authentication required: No

Manager Role required: No

Responses			
Status Code	Status Desc.	Input	Sample Response
200	OK	Logging in with valid username and password <pre>{ "username": "HelloWorld", "password": "ElginLee" }</pre>	<pre>{ "token": "*****", "username": "HelloWorld", "isManager": false }</pre>
401	UNAUTHORIZED	Logging in with invalid username / password <pre>{ "username": "HelloWorldddd", "password": "ElginLee" }</pre>	<pre>{ "msg": "Username or password is invalid" }</pre>

13.2.3 Get username

Endpoint: <http://localhost:3002/user-service/username>

HTTP Method: GET

Description: Sending a GET request to this endpoint with a JWT token will return the username of the user

Authentication required: Yes

Manager Role required: No

Responses			
Status Code	Status Desc.	Input	Sample Response
200	OK	Request sent with a valid JWT Token	{ "username": "ElginLee" }
401	UNAUTHORIZED	Request sent without a token attached to the HTTP header	{ "msg": "Authentication failed: missing token" }
401	UNAUTHORIZED	Invalid token attached to header	{ "msg": "Authentication failed: invalid token" }

13.2.4 Update password

Endpoint: <http://localhost:3002/user-service/update/password>

HTTP Method: PUT

Description: Sending a PUT request to this endpoint will attempt to change the password of the user

Authentication required: Yes

Manager Role required: No

Responses			
Status Code	Status Desc.	Input	Sample Response
200	OK	Correct old password and new password is valid <pre>{ "oldPassword": "ElginLee", "newPassword": "password1" }</pre>	<pre>{ "msg": "Update password successful" }</pre>
401	UNAUTHORIZED	Incorrect old password <pre>{ "oldPassword": "abcdefg", "newPassword": "password1" }</pre>	<pre>{ "msg": "Your old password is incorrect" }</pre>
400	BAD REQUEST	New password is invalid <pre>{ "oldPassword": "password1", "newPassword": "ab" }</pre>	<pre>{ "msg": "Password must be between 8 and 20 characters" }</pre>
401	UNAUTHORIZED	Request sent without a token attached to the HTTP header	<pre>{ "msg": "Authentication failed: missing token" }</pre>

401	UNAUTHORIZED	Invalid token attached to header	<pre>{ "msg": "Authentication failed: invalid token" }</pre>
-----	--------------	----------------------------------	--

13.2.5 Deregister

Endpoint: <http://localhost:3002/user-service/deregister>

HTTP Method: DELETE

Description: Sending a DELETE request to this endpoint will attempt to delete the user from the database

Authentication required: Yes

Manager Role required: No

Responses			
Status Code	Status Desc.	Input	Sample Response
200	OK	Valid JWT token attached to header of request	<pre>{ "msg": "User successfully deleted" }</pre>
404	NOT FOUND	Username attached to JWT token not found in database	<pre>{ "msg": "Delete failed, user not found" }</pre>
401	UNAUTHORIZED	Request sent without a token attached to the HTTP header	<pre>{ "msg": "Authentication failed: missing token" }</pre>
401	UNAUTHORIZED	Invalid token attached to header	<pre>{ "msg": "Authentication failed: invalid token" }</pre>

13.3 Collaboration Service

13.3.1 Create Room

Endpoint: <http://localhost:3004/collaboration-service/>

HTTP Method: POST

Description: Sending a POST request to this endpoint will attempt to create a room

Authentication required: Yes

Manager Role required: No

Responses			
Status Code	Status Desc.	Input	Sample Response
201	CREATED	{ "roomId": "d69d8c43-64ee-4689-9b77-101adf48f120", "questionId": "654ca2359d77f86a3bf34b83" }	{ "msg": "Successfully created" }
401	UNAUTHORIZED	Request sent without a token attached to the HTTP header	{ "msg": "Authentication failed: missing token" }
401	UNAUTHORIZED	Invalid token attached to header	{ "msg": "Authentication failed: invalid token" }

13.3.2 Get Room

Endpoint: <http://localhost:3004/collaboration-service/:roomId>

HTTP Method: GET

Description: Sending a GET request with the specified roomId to this endpoint will attempt to get the room's details

Authentication required: Yes

Manager Role required: No

Responses			
Status Code	Status Desc.	Input	Sample Response
200	OK	Valid JWT token attached to header and sending a GET request to the endpoint with the specified roomId	{ "roomId": "d69d8c43-64ee-4689-9b77-101adf4 8f120", "questionId": "654ca2359d77f86a3bf34b83" }
404	NOT FOUND	Invalid roomId specified that does not exist in the database	{ "error": "Cannot find room" }
401	UNAUTHORIZED	Request sent without a token attached to the HTTP header	{ "msg": "Authentication failed: missing token" }
401	UNAUTHORIZED	Invalid token attached to header	{ "msg": "Authentication failed: invalid token" }

13.3.3 Update room question

Endpoint: <http://localhost:3004/collaboration-service/room-question>

HTTP Method: PUT

Description: Sending a PUT request with the specified roomId and new questionId will update the room to have the new questionId

Authentication required: Yes

Manager Role required: No

Responses			
Status Code	Status Desc.	Input	Sample Response
200	OK	{ "roomId": "d69d8c43-64ee-4689-9b77-101adf48f120", "questionId": "654ca2359d77f86a3bf34b83" }	{ "msg": "Successfully updated question for room" }
400	BAD REQUEST	Invalid roomId specified that does not exist in the database	{ "error": "Room id is invalid" }
401	UNAUTHORIZED	Request sent without a token attached to the HTTP header	{ "msg": "Authentication failed: missing token" }
401	UNAUTHORIZED	Invalid token attached to header	{ "msg": "Authentication failed: invalid token" }

13.4 Code Execution Service

13.4.1 Execute all test cases

Endpoint: <http://localhost:3007/code-exec-service/execute-all>

HTTP Method: POST

Description: Sending a POST request attempts to run all test cases specified

Authentication required: Yes

Manager Role required: No

Responses			
Status Code	Status Desc.	Input	Sample Response
200	OK	{ "code": "def hello(x): ... (truncated)", "language": "Python", "allInputs": [{ "n": 5, "m": 6 }, { "n": 5, "m": 1 }], "allExpectedOutputs": [15, -15] }	{ "status": "All test cases successfully executed", "results": [{ "status": "Passed", "message": "" }, { "status": "Failed", "message": "" }] }
400	BAD REQUEST	Code sent with syntax error according to the language	{ "status": "Runtime Error", "message": "<Reason of failure here>", }
400	BAD REQUEST	User removed the method signature and there is no longer a way to execute the code properly	{ "status": "Runtime Error", "message": "Missing method signature, or you have not made changes to the code", }
400	BAD REQUEST	Code sent that takes too long to run	{ "status": "Time Limit Exceeded", "message": "Code took too long to run", }

401	UNAUTHORIZED	Request sent without a token attached to the HTTP header	{ "msg": "Authentication failed: missing token" }
401	UNAUTHORIZED	Invalid token attached to header	{ "msg": "Authentication failed: invalid token" }
500	Internal Server Error	Happens when Judge0 API fails to be called	{ "status": "Internal Server Error", "message": "<Reason of failure>" }

13.5 AI Service

Endpoint: <http://localhost:3006/ai-service/chatgpt>

HTTP Method: POST

Description: Sending a POST request fetches an appropriate response from ChatGPT

Authentication required: No

Manager Role required: No

Responses			
Status Code	Status Desc.	Input	Sample Response
200	OK	{ "Messages": [[{"role": "user", "content": "How to print hello world?"}]] }	{role: 'system', content: 'In Python, you can print "Hello World" with this I...ill appear as output in your console or terminal.'}
500	Internal Server Error	Happens when OpenAI fails to return a response to the query	{ "status": "Internal Server Error", "message": "<Reason of failure>" }

13.6 History Service

13.6.1 Add answered question

Endpoint: <http://localhost:3008/history-service/add-answered-question>

HTTP Method: POST

Description: Sending a POST request adds an answered question to the history database

Authentication required: Yes

Manager role required: No

Responses			
Status Code	Status Desc.	Input	Sample Response
200	OK	{ questionId, questionTitle, complexity, username, username2, answeredAt, isSolved, roomId }	{ success: Question added to `username's` history }
404	error	Happens when the database is not accessible	{ "error": error }

13.6.2 Get all answered question by username

Endpoint: <http://localhost:3008/history-service/history/:username>

HTTP Method: GET

Description: Sending a GET request returns all answered questions of a username

Authentication required: Yes

Manager role required: No

Responses			
Status Code	Status Desc.	Input	Sample Response
200	OK	{}	{ Question details }

14. Crucial API Latency Test

Getting all questions (a total of 400 questions) takes only 222ms, which meets the [NFR requirement](#) of our question service.

The screenshot shows a Postman request for `http://localhost:3001/question-service/`. The request method is GET, and the URL is `http://localhost:3001/question-service/`. The 'Params' tab is selected, showing a single query parameter 'Key' with value 'Value'. The 'Headers' tab shows 8 headers. The 'Tests' tab contains the following code:

```
if (response.status === 200) {
    console.log('All questions retrieved successfully');
}
```

The 'Body' tab displays the JSON response, which includes an ID, title, categories, complexity, and a detailed description about finding the K-th element in an array. The 'Tests' tab also includes a check for the total number of questions.

On the right, the 'Test Results' panel shows the API call details. It indicates a status of 200 OK, a time of 222 ms, and a size of 859.22 KB. A timeline chart shows the execution steps: Prepare (7.88 ms), Socket Initialization (2.56 ms), DNS Lookup (0.66 ms), TCP Handshake (0.78 ms), Transfer Start (202.95 ms), Download (14.81 ms), and Process (0.58 ms). The total duration is 230.18 ms.

Adding a new question takes 222ms as well, which meets the [NFR requirement](#) of our question service

The screenshot shows the Postman interface for a POST request to `http://localhost:3001/question-service`. The request body is a JSON object representing a question:

```
1  {
2    "title": "abcde",
3    "categories": ["Array"],
4    "complexity": "Easy",
5    "description": "desdc",
6    "codeTemplates": [ { "language": "Python", "template": "hello" }],
7    "inputs": [ { "n": 5, "m": 6 } ],
8    "outputs": [15]
9 }
```

The response status is 201 Created, with a time of 222 ms and a size of 274 B. The timeline details the request process:

EVENT	TIME
Prepare	6.9 ms
Socket Initialization	1.24 ms
DNS Lookup	0.33 ms
TCP Handshake	0.48 ms
Transfer Start	217.98 ms
Download	1.65 ms
Process	0.07 ms
Total	228.62 ms

Updating a question takes 202ms, which meets the [NFR requirement](#) of our question service

The screenshot shows a POST request to `http://localhost:3001/question-service`. The request body is a JSON object representing a question:

```
1  {
2    ...
3    "questionId": "6552493e7d30751ac39b1b8d",
4    ...
5    "title": "abcde",
6    ...
7    "categories": ["Array"],
8    ...
9    "complexity": "Easy",
10   ...
11   "description": "desdc",
12   ...
13   "codeTemplates": [ { "language": "Python", "template": "hello" } ],
14   ...
15   "inputs": [ { "n": 5, "m": 6 } ],
16   ...
17   "outputs": [15]
```

The response status is 201 Created, time: 202 ms, size: 276 B. The timeline shows the following events and their times:

EVENT	TIME
Prepare	6.19 ms
Socket Initialization	2.27 ms
DNS Lookup	0.69 ms
TCP Handshake	0.62 ms
Transfer Start	194.54 ms
Download	3.47 ms
Process	0.27 ms
Total	208.02 ms

Deleting a question takes 7ms, which meets our [NFR requirement](#) for our question service

The screenshot shows a Postman request configuration for a DELETE operation:

- URL:** `http://localhost:3001/question-service`
- Method:** `DELETE`
- Body:** `JSON` (selected)
- Body Content:**

```
1 {  
2   ... "ids": [  
3     ... "6552403e7d30751ac39b1b8d"  
4   ]  
5 }
```

Below the request, the response details are shown:

- Status:** 204 No Content
- Time:** 7 ms
- Size:** 190 B
- Total Time:** 10.24 ms

The timeline chart displays the following events and their durations:

Event	Time (ms)
Prepare	4.16 ms
Socket Initialization	0.99 ms
DNS Lookup	0.31 ms
TCP Handshake	0.52 ms
Transfer Start	2.91 ms
Download	1.32 ms
Process	0.07 ms
Total	10.24 ms

15. Software Development Process

15.1 Sprint System and Work Allocation

We used Asana, a collaboration platform, to keep track of our progress. Asana allows us to create different task items and allocate them to various sprints. This allowed us to achieve a high level of organization and allocate work smoothly based on our remaining tasks.

The screenshot shows the Asana project management interface for a project titled "CS3230-T08". The board view is organized into several columns representing different stages or sprints:

- Board Sprint 2:** Contains tasks such as "[A3] As a owner, I want registered users who are not a maintainer to not be able to add / update questions, so that I can protect against malicious users" and "[A5] As a user, I want to have a shared code editor with the user I am matched with, along with the randomized question, so that we can collaborate".
- Backlog:** Contains the task "[A1] Web scrape leetcode questions".
- Done:** Contains tasks such as "[A1] As a user, I want to have a page where I can see basic information of all questions, so that I can select the question I am interested in to view", "[A2] As a user, I want to be able to update my password and my username, so that it is more customizable", "[A2] As a user, I want to be able to log out of my account", and "[A2] As an administrator, I want user profiles to be managed by a SQL database, so that I can create personalized experience for users".
- Board Sprint 1:** Contains the task "+ Add task".
- Prioritized:** Contains the task "+ Add task".
- Developing:** Contains the task "+ Add task".

At the top of the interface, there are various navigation tabs like Overview, List, Board, Timeline, Calendar, Workflow, Dashboard, Messages, Files, and a Share button. There are also filter and sort options available.

Asana's Interface

Each of our sprints lasted two weeks, with a bi-weekly review on Saturdays to discuss the finished sprint and also to plan the upcoming sprint. Through Asana, we allocated tasks to team members and also used priority labels to further organize our workflow. Sprint meetings were structured; we first discussed our progress in the finished sprint, as well as any problems we ran into. We would then plan the overall targets for the upcoming sprint, then distribute the work amongst the team.

Tasks have their own priorities, assignees and due dates

In this project, we all wanted to move out of our comfort zone to pick up new skills, so rather than allocate tasks to members based on a member's strengths, we allowed members to pick and choose tasks that they wanted to do. By doing this, we aimed to become more well-rounded developers overall.

We held daily standup meetings on telegram every morning, updating one another on what we have done the previous day, what we are planning to accomplish for the day, and whether we have any difficulties faced. This helps us understand the progress of each other, and problems to keep in view.

15.2 Pair Programming

Our group participates heavily in pair programming, where one person will be watching another person code. This helps in improving our code quality and learning opportunities as we'll be catching out each other on bad practices, and is helpful for rubber duck debugging as well.

15.3 Source control and collaboration

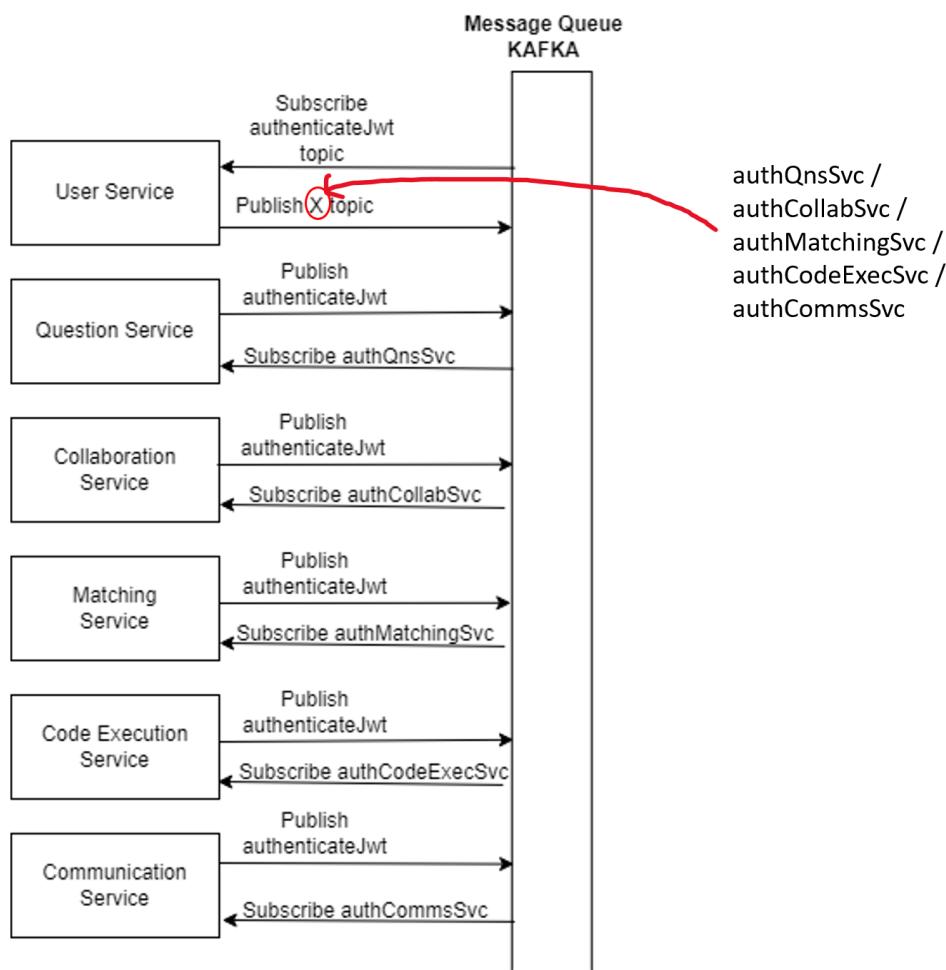
We also made use of GitHub's pull request and review system extensively. By developing on individual branches, we avoided the issue of contaminating the main codebase, and also ensured that changes made by members were of sufficiently high quality before combining them with our main codebase. This was especially useful for code-heavy features, allowing for prior discussion and explanation before merging.

16. Improvements and Enhancements

16.1 Lack of cohesion for services

Our services are not as cohesive as they could have been. For instance, our [authentication handling](#) for REST endpoints is haphazardly done. The user service should have been the only service that is responsible for verifying JWT tokens. However, our design made it such that all services will verify it themselves. Hence, every service needs to know the secret password that is used to sign the token, which does not make much sense.

Instead, we should have made use of a message queue for service-service communication.

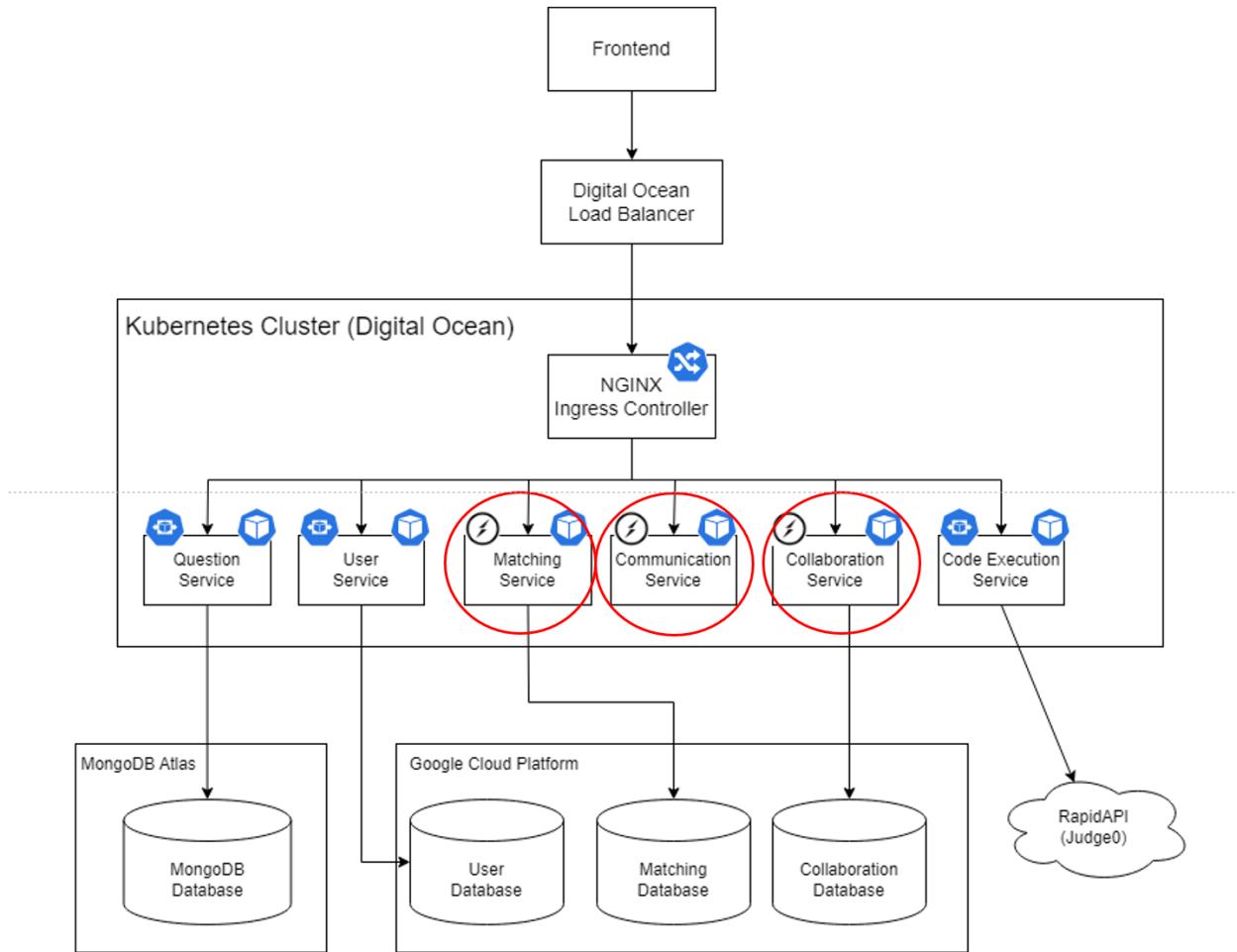


In this design, any changes in authentication methods need not be replicated across all services, making our microservices more modular.

16.2 Scalability of Stateful Services

Although our services reside in a Kubernetes cluster, we did not manage to fully harness the benefits of scalability, particularly in our stateful services that make use of websockets.

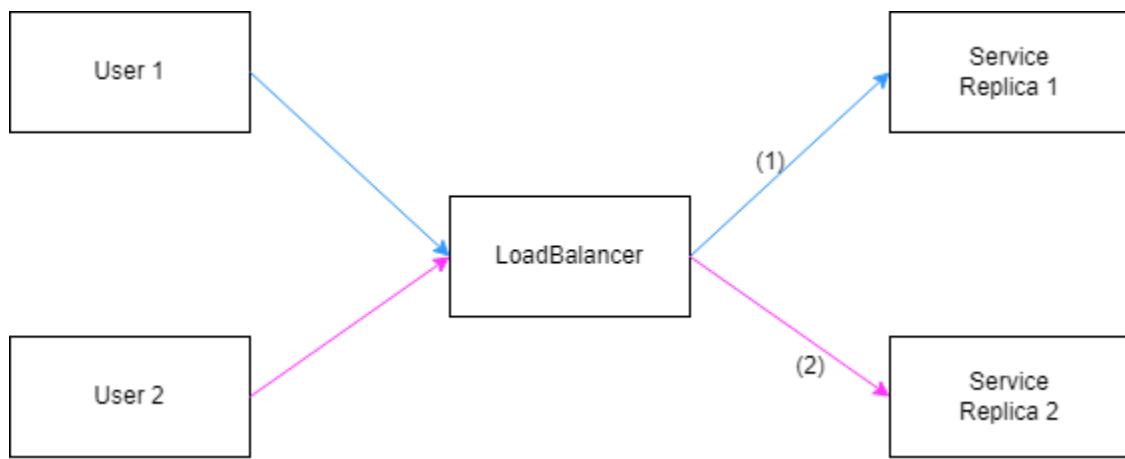
The services that are circled in red in the diagram below denotes the stateful services that are not scalable.



16.2.1 Problem

The reason for this is because of the use of in memory data structures, and websockets which are stateful. When we declare an io instance using the library “socket.io”, these io instances have their own state.

Imagine this scenario depicted by the diagram below. User 1 sends a request that routes to service replica 1, and sets up a connection with the websocket instance there. User 2 sends a request that routes to service replica 2, and sets up a web socket connection. Now, suppose both users join a socket room which has the same room id, but they will not be able to send messages to one another because they are on different io instances. This poses the problem of scalability.



16.2.2 Possible Fix

We need to employ some form of central state management, using tools like Redis. This will change our stateful service to stateless, since all state management is done through the central database. In the context of the example above, when user 1 connects to service replica 1, it should probably publish a message to notify service replica 2 that this user has connected. This way, both users can communicate with one another.

16.3 JWT Token Handling

Currently, our application's JWT token handling is not good. It does not have an expiry set to it. Although there are benefits such as convenience for users since they do not need to reauthenticate frequently, the disadvantages far outweighs the advantages. It poses some form of security risk because if a token is compromised or stolen, an attacker could have persistent access to the system for an extended period, and lead to lesser control over user sessions and access. With long-lived tokens, it's harder to enforce access policies, and revoking access for a user or deactivating their session is challenging.

What we could have done instead is to make use of refresh tokens. Refresh tokens is a long-lived token that is issued to a client application during the initial authentication process, and used to obtain a new access token when the current access token expires. This reduces the lifetime of access tokens, and when an access token is compromised, its limited validity window minimizes the potential for misuse. Furthermore, it allows clients to obtain a new access token without requiring users to re-enter their credentials or perform the entire authentication process repeatedly.

16.4 Logic handling for Add Question, Update Question

Our input validation logic for adding questions and updating questions is in both the backend and frontend code. However, this is not a good design because it is hazardous and difficult to maintain. Instead, we should have decided on whether we want the validation logic to be either in frontend or backend. Preferably, it should have been in the backend so that we can prevent rogue values added to the database in case some users did not add a question through our interface but send a POST request through postman instead.

16.5 Reasoning behind using database for the Matchmaking Service

Before our implementation of our matchmaking service, we were advised to use a message queue in order to match users against one another. However, after deep consideration of our vision for this application, we realized that using a database for our matchmaking service would open more opportunities to improve our application in the future such as gamifying this peerprep application in the near future which would include elo matchmaking to further improve our users experience.

17. Reflections and learning points

This is a very interesting project that provides us with many opportunities to learn and employ new tools. It requires us to implement features that none of us have done before such as matching service, collaboration service, AI service and code execution. It kind of forces us to question a lot of design decisions, what we could have done better or how it could have been done differently in every step of the project. As we went along implementing the different features we had in mind, we realized that we had not put enough thought into how each feature might overflow into the others. This resulted in having to spend a lot of time looking through the code of each implementation in order to see where we are able to fit it in. Although we did not manage to implement the software perfectly since it has some [flaws](#) that we did not have enough time to fix, we're still glad that we know the limitations of our application and how we could have improved it compared to being oblivious.

From this project, we manage to learn more about microservices and how to utilize the DDD technique to spot which services are needed. We also gained a better understanding of several technologies such as Websockets, Docker, and Kubernetes. It also gives us an experience of working together in teams, coming up with requirements together and brainstorming about the design of the application.

Most importantly, we learned the importance of understanding requirements, which is a key part of the software development process. At the core of understanding requirements is grasping what the end users need and expect from the software. Properly identifying user needs ensures that the developed software solves the right problems and adds real value. We realized that in order to identify our exact requirements, we have to look at the application from the viewpoint of a user rather than a developer. When identifying our requirements, we should also think of any potential opportunities to build upon our current features to provide our users with a better experience. On the side of working as a team for this project, clear and well-understood requirements improve communication among team members and between developers and stakeholders. This clarity reduces misunderstandings and ensures that everyone is working towards a common goal.

