CS3219 Software Engineering Principles and Patterns

AY23/24 Semester 1

Group 9

Project Report

Student Name	Matriculation Number
Ang Jia Le Marcus	A0235025R
Chan Choon Yong	A0222743L
Lum Wee Kiat	A0234939R
Wilson Ng Jing An	A0244474B
Wu Yuxin	A0205199Y

Table of Contents

1. Introduction

1.1 Background

The initiation of the PeerPrep project stems from the ever-increasing demand for effective technical interview preparation. In today's competitive job market, technical interviews have become a critical component in the evaluation of prospective candidates. Many aspiring individuals, especially students and job-seekers, face the challenge of adequately preparing for these interviews. The PeerPrep platform has been conceived as a solution to address this pressing need.

1.2 Purpose and Objectives

The primary purpose of the PeerPrep project is to create a comprehensive technical interview preparation platform that assists users in honing their skills and increasing their confidence in tackling whiteboard-style interview questions. This platform is designed to bridge the gap between theoretical knowledge and practical application by offering users a collaborative environment to practice interview questions. The project aims to achieve the following key objectives:

S/N	Objective	Description
O1	Create a Web Application	Develop a web-based platform that facilitates technical interview practice and peer matching.
O2	User Account Management	Allow users to create accounts, login and manage their profiles.
O3	Question Difficulty Selection	Enable users to choose from various question difficulty levels (easy, medium, or hard) based on their preferences and skill level.
O4	Topic Filtering	Provide the option for users to select specific topics they wish for viewing.
O5	Peer Matching System	Implement a real-time peer matching system that pairs users with others of similar difficulty level choice.
O6	Collaborative Workspace	Offer a collaborative space for paired users to work on interview questions together in real-time.

O7	Graceful Termination	Allow users to terminate collaborative sessions gracefully, ensuring a positive user experience.
O9	History Review	Allow users to review records of past collaboration which include questions done and code history.
O9	Easy and Quick Simulation	The platform should provide an easy and efficient way for users to simulate technical interviews, emphasizing the user-friendliness of the platform. From login to collaborative practice, the platform must be quick, easy, and intuitive for users, making PeerPrep an ideal platform for quick, efficient, and effective technical interview practices.

Table 1: Project Objectives Table

1.3 Scenario

To illustrate the practical context of the PeerPrep platform, consider the following scenario:

Imagine a student eager to prepare for technical interviews. This student visits the PeerPrep website, creates an account, and logs in. After login, the student selects the difficulty level of the interview questions (easy, medium, or hard) they wish to practice. The system then initiates a search for another online student with the same difficulty level choice.

If a match is successfully found within a predetermined time frame, both students are provided with the interview question and access to a collaborative workspace where they can jointly develop their solutions. Importantly, the system allows users to conclude their collaborative sessions gracefully when they are satisfied or have completed their practice.

It is crucial to note that the PeerPrep platform's features extend beyond this scenario, catering to a diverse range of users and interview preparation needs.

2. Requirements Specification

2.1 Quality Attributes Prioritization Matrix

To establish clear and prioritized application requirements, it is essential to identify and rank key quality attributes based on project purpose and objectives. Utilizing a prioritization matrix for the web-based PeerPrep application, several quality attributes are pertinent, including availability, integrity, interoperability, performance, scalability, security, and usability, as detailed in L3-Requirements.pdf.

Attribute	Score	Availability	Integrity	Interoperability	Performance	Scalability	Security	Usability
Availability	2		^	^	^	<	<	^
Integrity	3			<	^	<	^	^
Interoperability	1				^	^	^	^
Performance	5					^	<	<
Scalability	4						<	^
Security	2							^
Usability	5							

Figure 1: QA Prioritization Matrix

Through the application of the above matrix, it becomes evident that usability, performance, and scalability emerge as the top three paramount quality attributes. These attributes will serve as foundational pillars for subsequent project planning and pivotal decisions, shaping the prioritization of functional requirements, technology stack selection, and architectural choices to align with the core objectives and also future enhancements of the project.

2.2 User Roles and Permissions

In the PeerPrep platform, users are classified into three primary categories: unregistered users, registered users, and administrative users. Each category corresponds to distinct user privileges and access rights. Below, there is a table illustrating the different privileges and access rights associated with each user type:

User Type	Privileges and Access Rights
Unregistered User	• Unable to access any service in PeerPrep
	• Redirected to account creation page
Registered User (Normal User)	• Account login and profile viewing
	• View questions and apply tag-based filters for enhanced search and organization
	• Selection of question difficulty
	• Matching with other users / peers for collaborative practice
	• Access to collaborative workspaces
	• Review of collaboration history
Administrative User	• Account login and profile viewing
	• Able to view, edit and delete all users
	• Able to create, edit or delete a question
	• Able to view all existing questions in question pool and apply tag-based filters for enhanced search and organization
	• Matching and Collab Feature of Normal Registered User

Table 2: User Privilege Table

2.3 Functional Requirements

In this section, functional requirements (FRs) are categorized and listed based on the service type. Also, the priority and effort for each FRs will be indicated as well. Priority is represented by H (High) in **Red**, M (Medium) in **Orange** or L (Low) in **Yellow**.

2.3.1 User Service

S/N	Description	Priority
F1	**User Service**	
F1.1	**User Account Creation**	
F1.1.1	The application should allow registration of new user accounts.	H
F1.1.2	The application should check for duplicate username / emails during registration.	H
F1.1.3	The application should do basic syntax checking to ensure user data integrity and validity during registration.	H
F1.2	**User Login**	
F1.2.1	The application should check if the username exists.	H
F1.2.2	The application should check if the password matches the username.	H
F1.3	**User Authentication**	
F1.3.1	The application should be able to identify user types and assign privileges and access rights accordingly (defined in section 2.2).	H
F1.3.1a	The application should deny access to questions for unauthenticated / unauthorized users.	H
F1.3.1b	The application should allow registered users to view the questions from the question repository.	H
F1.4	**CRUD Users**	
F1.4.1	The application should store user data upon user account creation.	H
F1.4.2a	The application should allow registered users to view their own user profile.	M
F1.4.2b	The application should allow the admin to view all existing registered users.	M

F1.4.3a	The application should allow registered users to edit their own user profile.	H
F1.4.3b	The application should allow the admin to edit profile details of any registered users.	M
F1.4.4	The application should allow the admin to delete existing registered users.	M

Table 3: User Service FR Table

2.3.2 Matching Service

S/N	Description	Priority
F2	**Matching Service**	
F2.1	**Match Request**	
F2.1.1	The application should allow users to select preferred question difficulty prior matching.	H
F2.1.2	The application should match users in pairs based on their preferred question difficulty.	H
F2.1.3	The application should inform the user when a match request failed due to timeout (30 secs).	H
F2.1.4	The application should be able to match users in pairs based on custom room ID as alternative.	H
F2.1.5	The application should bring both users to the collaboration page when matched.	H
F2.2	**Match Cancellation**	
F2.2.1	The application should allow users to cancel match requests prematurely before timeout.	M
F2.3	**Match Confirmation**	
F2.3.1	The application should inform both users about successful matching if the match requests succeed in pairing.	L
F2.3.2	The application should prompt a choice for the users to accept or decline the match session.	L

F2.3.2a	The application should bring both users to the collaboration page if both users accept the matching session.	L
F2.3.2b	The application should automatically initiate a prioritized match request if the current user accepts the match session while the other paired user declines the match session or timeout by not selecting any options.	L
F2.3.3	The application should not do anything if user decline match session or timeout by not selecting any options	L

Table 4: Matching Service FR Table

2.3.3 Question Service

S/N	Description	Priority
F3	**Question Service**	
F3.1	**CRUD Questions**	
F3.1.1	The application should allow the admins to create new questions.	H
F3.1.2	The application should do basic syntax and data constraints checks to ensure question data integrity and validity during question creation. A question MUST include and adhere to the following: ● Unique Question Id ● Non-Empty Title ● Non-Empty Description ● Defined Question Difficulty	H
F3.1.3	The application should allow both admins and registered users to view questions from the question pool.	H
F3.1.4	The application should allow admins to edit existing questions and conduct basic syntax and data constraints checks subsequently. If checks fail , the edit process is halted.	H
F3.1.5	The application should allow admins to delete existing questions from the question pool.	H
F3.2	**Filter Question**	
F3.2.1	The application should allow users to filter questions based on difficulty when viewing.	M

F3.2.2	The application should allow users to filter questions based on topic when viewing.	M

Table 5: Question Service FR Table

2.3.4 Collaboration Service

S/N	Description	Priority
F4	**Collaboration Service**	
F4.1	**Collaborative Space**	
F4.1.1	The application should display a question of selected difficulty along with a code editor.	H
F4.1.2	The application should enable concurrent code editing such that any changes made by any user on the code editor is reflected for both users.	H
F4.1.3	The application should allow users to distinguish themselves based on the unique cursor colour in the code editor.	L
F4.2	**Enhanced Code Editor**	
F4.2.1	The application should allow the users to select their preferred programming language for syntax highlighting. Once changed, it will be reflected on code editors of both users. The code editor should be able to support the following 4 programming languages: ● Java ● JavaScript ● Python ● C++	H
F4.2.2	The application should enable code auto-formatting when users click the 'Format' button. The formatted code should be reflected in the code editors for both users. [JavaScript Supported Only]	H
F4.3	**Video Communication Service**	
F4.3.1	The application should allow users to video call each other.	H
F4.3.2	The application should allow users to toggle their camera during the	H

	call.	
F4.4	**Custom Collaboration Room Service**	
F4.4.1	The application should allow users to join a custom room instead of through a match difficulty request.	H

Table 6: Collaboration Service FR Table

2.3.5 History Service

S/N	Description	Priority
F5	**History Service**	
F5.1	**Question History**	
F5.1.1	The application should keep track of the number of questions per difficulty user did in collaboration. This information should be available for use in the user interface as per UI requirements.	L
F5.2	**Collaboration History**	
F5.2.1	The application should store collaboration history information which includes the following when collaboration session ends: ● Paired User Ids ● Question done during Collaboration ● Written code in the code editor This information should be available for use in the user interface as per UI requirements.	H

Table 7: History Service FR Table

2.3.6 User Interface

S/N	Description	Priority
F6	**User Interface (UI)**	

F6.1	**Dashboard (Landing Page)**	
F6.1.1	The application should dynamically render the appropriate dashboard based on the user's role, distinguishing between Registered Users and Administrators.	H
F6.1.2	For registered users, the application should show the following in the dashboard: ● Matching Component (Select Difficulty, Match button and Cancel Button, etc) ● View Question Component (Questions displayed based on filtering) ● Custom Room Button (For custom collaboration room) ● History Component (Question Accomplishment Statistics by Difficulty, Collaboration History - Question Done, Code, etc)	H
F6.1.3	For admins, the application should show the following in the dashboard: ● Registered User List ● Question CRUD component UI	H
F6.2	**User Login Page**	
F6.2.1	The application should have a page for user login.	H
F6.3	**User Account Creation Page**	
F6.3.1	The application should have a page for account creation.	H
F6.4	**Collaboration Page**	
F6.4.1	The application should have a page for collaboration.	H

Table 8: User Interface FR Table

2.3.7 Application Deployment

S/N	Description	Priority
F7	**Application Deployment**	
F7.1	**Containerization**	
F7.1.1	Each microservices of the application should be containerized. This includes all microservices:	H

	• User Authentication Service • Collaboration Service • Communication Service • History Service • User Service • Matching Service • Question Service • User Service	
F7.2	**API Gateway**	
F7.2.1	The application should have an API Gateway that coordinates requests between the containerised microservices	H
F7.3	**Automate Testing Process**	
F7.3.1	The application should have a basic suite of test cases to verify the successful execution of the containers.	L
F7.3.2	The application should have deployed on a Cloud-based platform using CI/CD deployment pipeline for testing and deployment.	L
F7.4	**Deployment via Cloud Services (Kubernetes)**	
F7.4.1	The application should run on cloud instances hosted by a Cloud-based platform.	M
F7.4.2	The application should be choreographed by Docker & Kubernetes on the cloud.	M
F7.5	**Scaling on Cloud Instances**	
F7.5.1	The application should support horizontal and/or vertical scaling.	M
F7.6	**Service Discovery (Kubernetes)**	
F7.6.1	The application should support service discovery between Kubernetes pods using DNS features.	M
F7.7	**Cloud Monitoring**	
F7.7.1	The cloud instances running the application should have a basic suite of logging and notification.	L
F7.7.2	The cloud instances running the application should have a basic suite of network security and application security.	L

Table 9: Application Deployment FR Table

2.4 Must Have (Mx) and Nice-To-Have (Nx)

The table below captures the coverage of Mx and Nx based on FRs defined in section 2.3.

Mx / Nx	Feature	Criteria Description	FRs
M1	User Service	Responsible for user profile management.	• F1
M2	Matching Service	Responsible for matching users based on difficulty level of questions. This service can potentially be developed by offering multiple matching criteria.	• F2
M3	Question service	Responsible for maintaining a question repository tagged by difficulty level.	• F3
M4	Collaboration service	Provides the mechanism for real-time collaboration (concurrent code editing) between the authenticated and matched users in the collaborative space.	• F4.1
M5	Basic UI	Basic UI for user interaction to access the developed app.	• F6
M6	Deployment (Local)	Deploying the application on your local machine (e.g., laptop) using native technology stack. OR Deploy the app on a (local) staging environment (e.g., Docker-based, Docker + Kubernetes).	• F7.1
N1	Communication Service	Implement a mechanism to facilitate communication among the participants in the collaborative space (other than the shared workspace) like text-based chat service and/or video with voice calling service.	• F4.3
N2	History Service	Maintain a record of the questions attempted by the user with timestamp, the attempt itself and/or suggested solutions.	• F5
N5	Enhance Collaboration Service	Provide an improved code editor with code formatting (JS only). Syntax highlighting for multiple languages.	• F4.2
N9	Deployment (Production)	Deployment of the app on the production system (AWS/GCP cloud platform)	• F7.3, F7.4

N10	Scalability	The deployed application should demonstrate easy scalability of some form. An example would be using a Kubernetes horizontal pod auto-scaler to scale up the number of application pods when there is a high load.	• F7.5, F7.6
N11	API Gateway	The application should have an API gateway of some kind that redirects requests to the relevant microservices. An example would be using an ingress controller such as NGINX ingress controller if using Kubernetes	• F7.2
N12	Service Discovery	The application should demonstrate service discovery or implement a service registry of some kind. For example, Kubernetes has built-in DNS features and service networking to pods	• F7.6

Table 10: Mx/Nx - to - FR Mapping Table

2.5 Non-Functional Requirements (NFRs)

The following is a table of NFRs for PeerPrep.

Non-Functional Requirements (NFRs)		
S/N	Requirement	Description
NF1	**System Requirement**	
NF1.1	Browser Compatibility	PeerPrep should be compatible with all Long-Term Support (LTS) versions of leading web browsers, including but not limited to: • Google Chrome • Mozilla Firefox • Apple Safari • Microsoft Edge
NF2	**Constraints**	
NF2.1	PC Usage Focus	PeerPrep is primarily designed for use on PCs rather than other devices such as mobile phones or tablets. This focus on PC usage is ideal for coding and interview preparation.
NF3	**Usability**	

NF3.1	Intuitive UI	PeerPrep's user interface must be intuitive and user-friendly. Untrained users should be able to learn how to use all features within 5 minutes.
NF3.2	Straightforward Navigation	Navigation should be straightforward, allowing users to easily access and utilize all features
NF3.3	Efficient Interactions	All interactions, from account creation to collaborative sessions should be clear, efficient, and error-free.
NF3.4	Login Status Retention	Retain Login Status for 2 days to enhance ease-of-use.
NF4	**Performance**	
NF4.1	Page Loading Time	Loading time for pages, questions, and collaborative sessions should be completed in 1 second.
NF4.2	Response Time	Response time should not exceed 1 second for each type of interaction.
NF4.3	Code Synchronization	Code typed in the code editor of the collaboration space by one user should reflect on the other user's editor within 2 seconds.
NF5	**Scalability**	
NF5.1	Concurrent Users	Application should be able to sustain 10000 concurrent users.
NF5.2	Concurrent Collaborations	Application should be able to sustain 5000 concurrent collaborations.
NF5.3	Component Adaptability	Flexibility to add or remove components without disruptions.
NF6	**Security**	
NF6.1	User Registration for Access	Users must be registered with the system to be able to login and access PeerPrep services such as viewing of questions.
NF6.2	Credential Encryption and Key Rotation	Encryptions for credential and update the encryption key once every 6 months.
NF6.3	Cloud Security	Cloud instances running PeerPrep should be secured with proper network configuration and access control.

NF7	Integrity	
NF7.1	Data Integrity Protection	The system shall prevent unauthorized or invalid addition, deletion or modification of data through check constraints in UI and databases.
NF7.2	Data Consistency	Data should be consistent across all databases.

Table 11: NFR Table

3. Contributions to the Project

3.1 Individual/Sub-group Contributions

Student Name	Contribution	Special Notes
Ang Jia Le Marcus	Technical: ● Matching Service Non-Technical: ● Report - 1. Introduction - 2. Requirements Specification - 4. Architecture and Design ● 4.1 Tech Stack ● 4.2 User Flow ● 4.3 Application Flow ● 4.6.4 Matching Service (Inclusive of Architecture Diagram and SDs) ● 4.6.5 Collaborative Space and History Services (Inclusive of Architecture Diagram and SDs) - 5. Future Enhancements and Features	Despite starting with limited knowledge and exposure to various technologies, this team member made earnest efforts to overcome the learning curve. Although time constraints allowed only the acquisition of sufficient knowledge for implementing a basic version of the matching service, the member took on additional responsibilities on the non-technical side. The member played a key role in completing multiple sections of the report, encompassing the creation of various tables and diagrams. Furthermore, recognizing the importance of collaboration, the member sought input from other team members to verify the technical components they implemented, emphasizing a collective effort to save time and ensure accuracy.
Chan Choon Yong	Technical: ● User Service - Integration with PostgreSQL + REST API (M1) ● Question Service (M3) ● Serverless Function (Question Scraper) ○ AWS Lambda ● Deployment (M6) ○ Docker Containerisation ● Cloud Deployment (N9) ○ AWS Elastic Container Registry ○ AWS Elastic Beanstalk	On the technical front, the team member made substantial contributions to the development of the User Service, integrating it with PostgreSQL and implementing a REST API (M1). Their adaptability was evident in the role played in the development of the Question Service (M3) and

	○ AWS Elastic Kubernetes Service (N12) Non-Technical: ● Report (Deployment-Related)	the creation of a Serverless Function for the Question Scraper using AWS Lambda. The team member also handled the deployment process (M6), demonstrating proficiency in Docker containerization and cloud deployment (N9). In the realm of cloud deployment, the team member showcased expertise in utilizing AWS services, such as AWS Elastic Container Registry, AWS Elastic Beanstalk, and AWS Elastic Kubernetes Service (N12). This versatile understanding of cloud platforms and container orchestration significantly contributed to the project's technical aspects. On the non-technical side, the team member played a crucial role in crafting the report's deployment-related sections, showcasing their ability to communicate complex technical concepts clearly and comprehensively. This dual proficiency in both technical and non-technical domains underscores the team member's versatility and unwavering commitment to the project's overall success.
Lum Wee Kiat	Technical: ● User and Authentication Service ● API Gateway (N11)	Despite initial limited familiarity with diverse technologies, this team

	● Deployment ○ Kubernetes Deployment ○ Kubernetes Service (N12) ○ Horizontal Pod Auto-scaler (N10) Non-Technical: ● Report (User, Authentication, API Gateway-Related, Local Deployment)	member made remarkable strides in overcoming the learning curve. Notably, they took on additional responsibilities on both technical and non-technical fronts, showcasing versatility and dedication. On the technical side, the team member significantly contributed to the development of the User and Authentication Service, seamlessly integrating it with PostgreSQL and implementing a REST API. Their expertise extended to API Gateway (N11) and the intricate aspects of deployment, including Kubernetes Deployment, Kubernetes Service (N12), and Horizontal Pod Auto-scaler (N10). This demonstrated a robust understanding of cloud-native technologies and container orchestration. Simultaneously, the team member played a crucial role in the non-technical realm, particularly excelling in report sections related to User, Authentication, API Gateway, and Local Deployment. This showcased effective communication of complex technical concepts and underscored their strong overall contribution to the team's success.

Wilson Ng Jing An	Technical: ● Frontend (Full) ● Frontend - Backend Integration ● Question Service ● User Service (Partial) ● Code formatter (N5) Non-Technical: ● Report (Frontend & Question-Related)	This team member, despite starting with limited knowledge, made substantial contributions to both technical and non-technical aspects of the project. They played a key role in developing the entire Frontend, ensuring seamless Frontend-Backend Integration, and contributing significantly to the Question Service. Their partial involvement in the User Service and role in the Code Formatter (N5) showcased versatility. On the non-technical side, they made impactful contributions to multiple sections of the report, emphasizing effective communication of technical concepts. Their collaborative approach in seeking input from team members underscored a commitment to accuracy and a collective effort to save time. Overall, their journey reflects adaptability, dedication, and a holistic commitment to the project's success.
Wu Yuxin	Technical: ● Matching Service ○ Integration with RabbitMQ and SQlite DB ● Collaboration Service with Video (N5) ○ Code Editor implemented with monaco editor ○ Events synchronized with both users through socket.io ● Communication Service (N1) ○ Video with simplePeer ○ Signaling done through socket.it ● History Service (N2)	This team member, starting with limited technical knowledge, made significant strides to implement key components, including the Matching Service with RabbitMQ and SQLite DB integration, a versatile Code Editor with Monaco Editor, and the Communication Service (N1) with video capabilities using simplePeer and socket.it for signaling.

	○ REST API with mongoDB Non-Technical: ● Report (Above Technical-Related)	Their involvement extended to the History Service (N2) with a MongoDB-based REST API. Beyond technical accomplishments, the member actively contributed to the report, creating tables and diagrams, and prioritized collaboration by seeking input from team members, showcasing a comprehensive and impactful role in the project.

Table 12: Contribution Table

Note:

- Work distribution is greatly influenced by individual area of expertise(s):
  - Yuxin: ReactNative, Firebase, Python
  - Wee Kiat: Express, React, MongoDB, Firebase
  - Marcus: PostgreSQL
  - Wilson: React, PostgreSQL, MongoDB
  - Choon Yong: Deployment, Testing, Containerization, ChatGPT API

3.2 Grading for Sub-Groups

The table below is the distribution of the Nx mentioned in section 2.4 between the different sub-groups.

S/N	Sub Group Members	Nx Successfully Implemented
1	● Ang Jia Le Marcus ● Wilson Ng Jing An ● Wu Yuxin	N1, N2, N5
2	● Lum Wee Kiat ● Chan Choon Yong	N9, N10, N11, N12

Table 13: Nx Distribution Table

4. Architecture and Design

4.1 Tech Stack

Below are the tech stack used for the different microservices or components used in PeerPrep.

Services / Components	Tech Stack
User Interface (Frontend)	● NextJS ● Chakra UI
User Service	● PostgreSQL
Authentication Service	● JWT ● BCrypt
Question Service	● MongoDB

Matching service	• Socket.IO • RabbitMQ • Sequelize
Collaboration Service	• Socket.IO
Video Service (Collab)	• Peer • Socket.IO
History service	• MongoDB
Serverless	• Puppeteer
Serverless Function	• Serverless Framework • AWS Lambda
API Gateway	• NGINX
Code deployment	• Docker • Kubernetes • Github Actions • Amazon Elastic Container Registry • Amazon Elastic Kubernetes Service (using Amazon EC2 managed nodes)

Table 14: Tech Stack Table

4.2 Overall Architecture

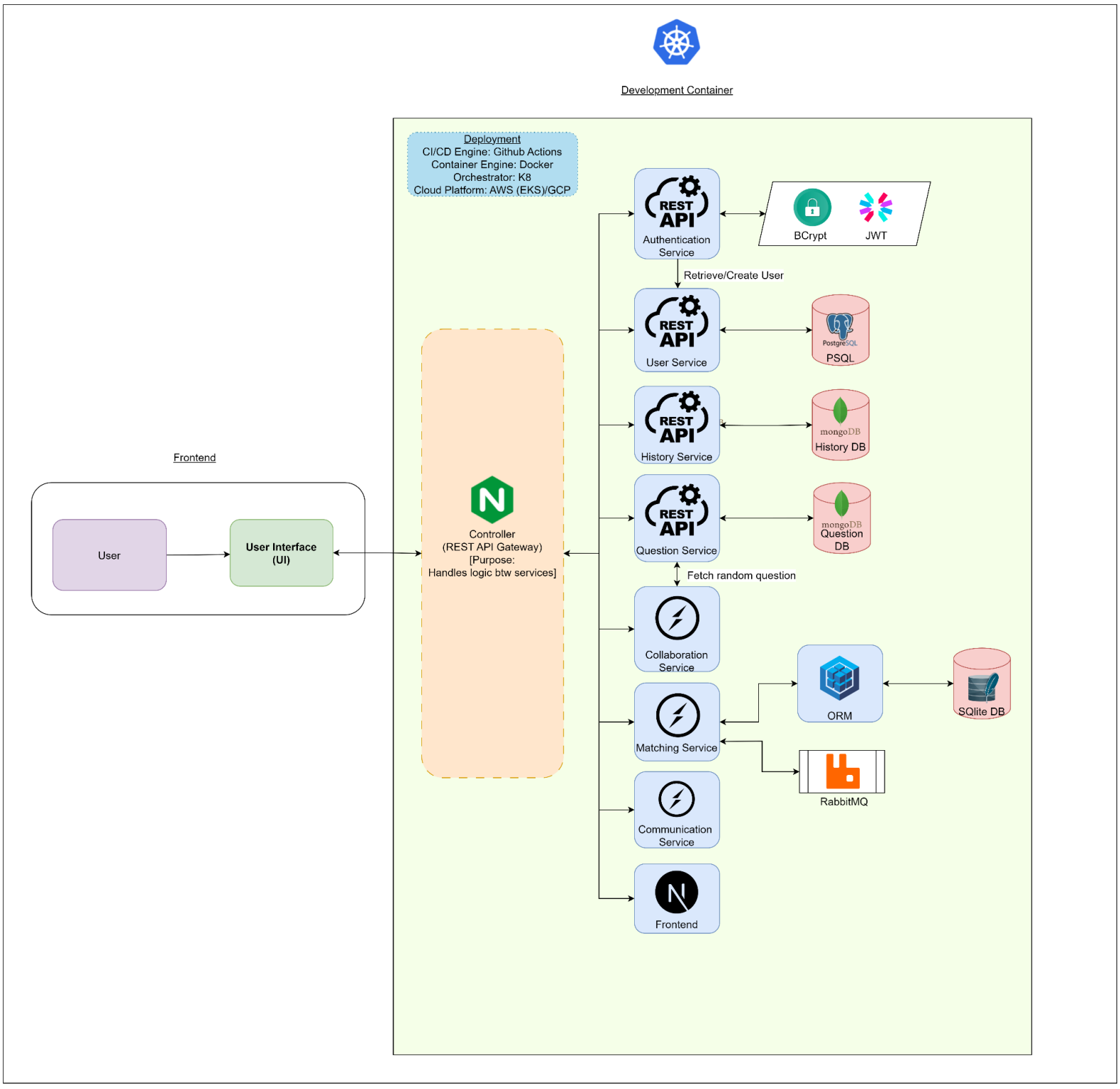

Figure 2: Overall Architecture Diagram

The team is employing microservices architecture with Kubernetes as an orchestrator. Sitting between our services and users is an NGINX API Gateway which redirects requests to the relevant microservices. Authentication Service is a central point that handles authentication and authorization in the PeerPrep application. Matching, Communication and Collaboration work together to deliver the real-time collaboration functionality. User and History services keep track of user activities. By dividing our application into microservices, it gives us the capability to scale specifically the services that are under high load or are expected to be.

In the forthcoming sections 4.3 and 4.4, user flow and application flow will be delineated to provide a comprehensive understanding of event sequences from various user perspectives. A color-coded scheme has been employed for clarity:

- Red denotes actions executed by unregistered users
- Blue signifies activities associated with admin functionalities
- Green is indicative of actions performed by registered users
- Purple represents collaborative actions that are relevant to both admin and registered users

This color-coded approach aims to enhance visual clarity and facilitate the comprehension of distinct user roles and their corresponding interactions within the application.

4.3 User Flow

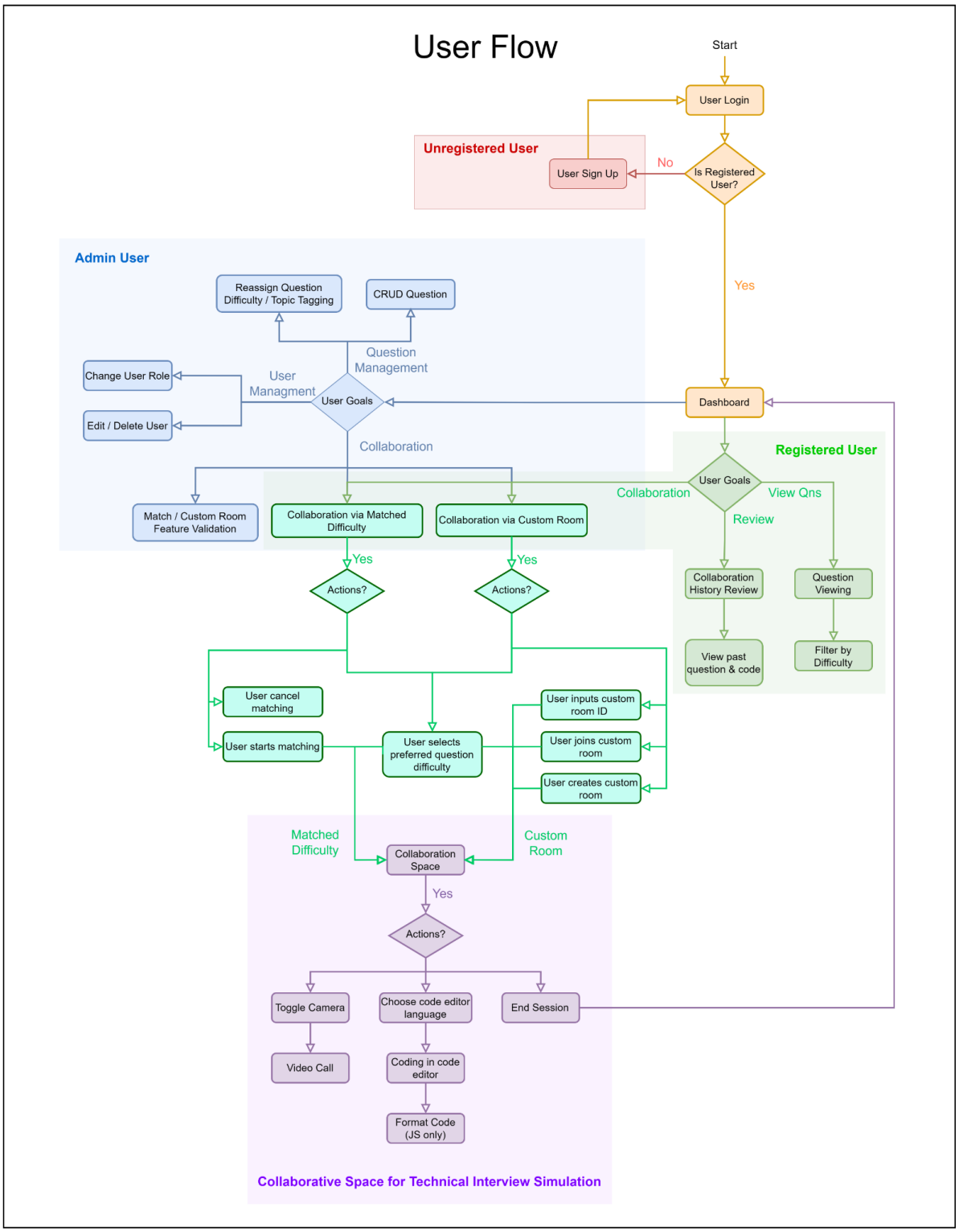

Figure 3: User Flow Diagram

4.4 Application Flow

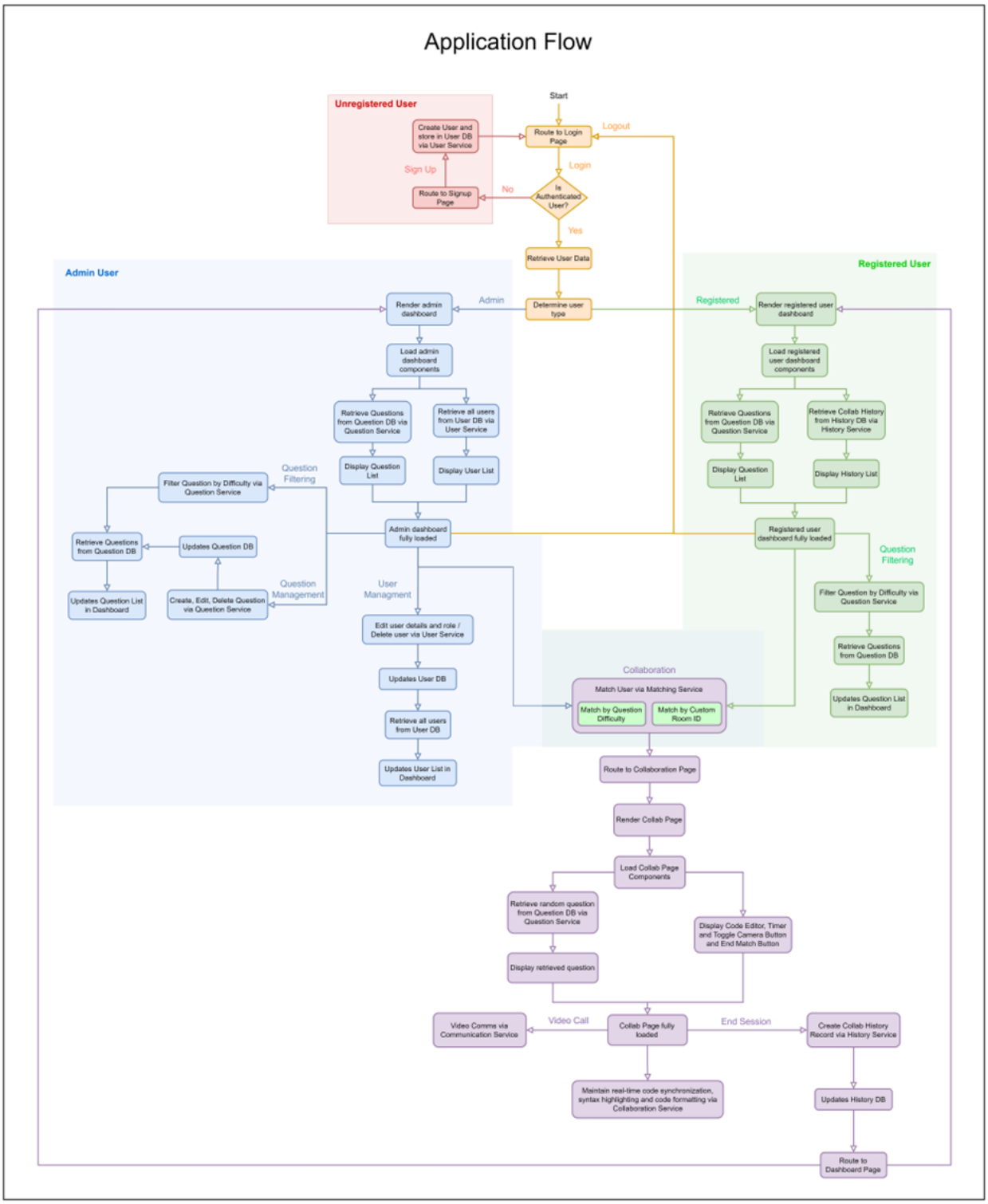

Figure 4: Overall Application Flow Diagram

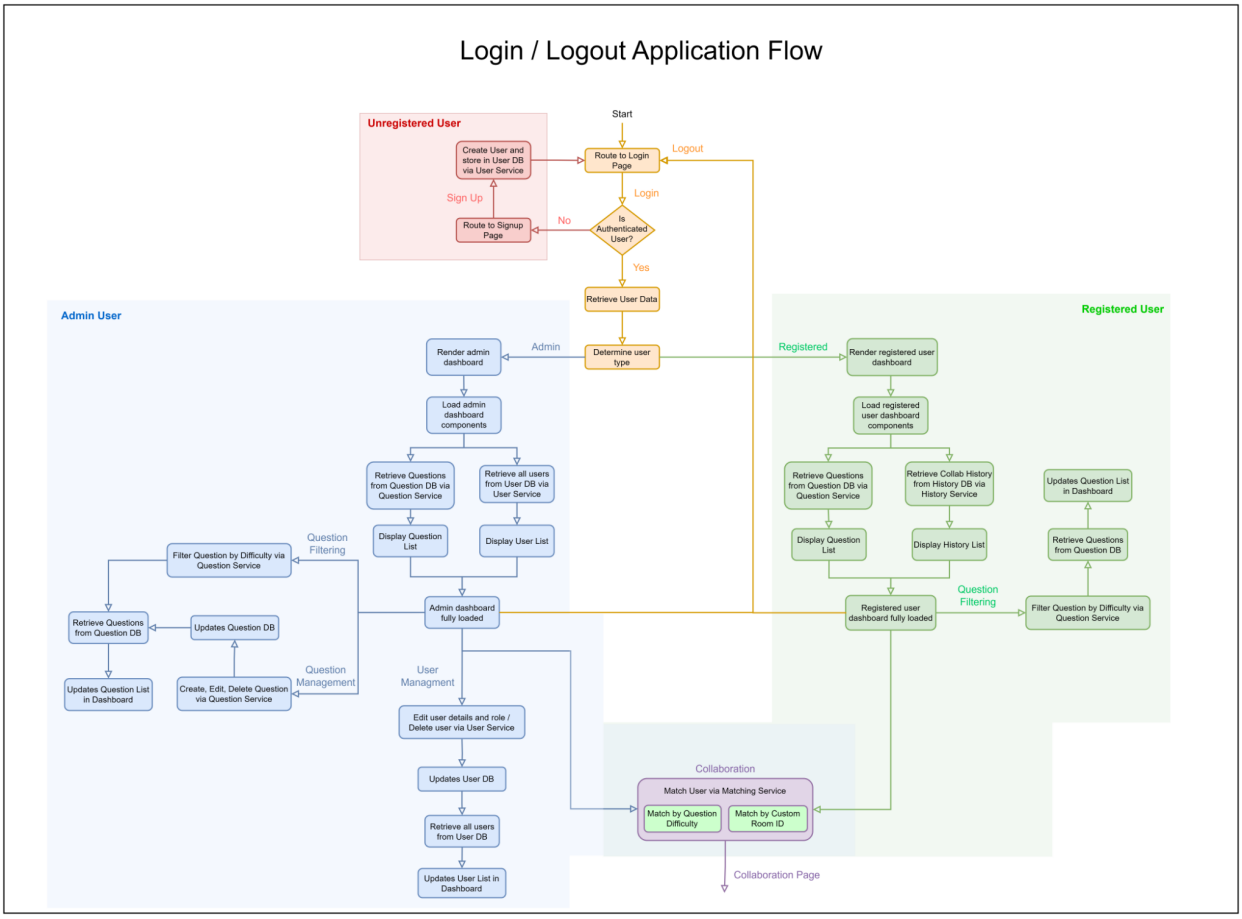

Figure 4a: Login / Logout Application Flow Diagram

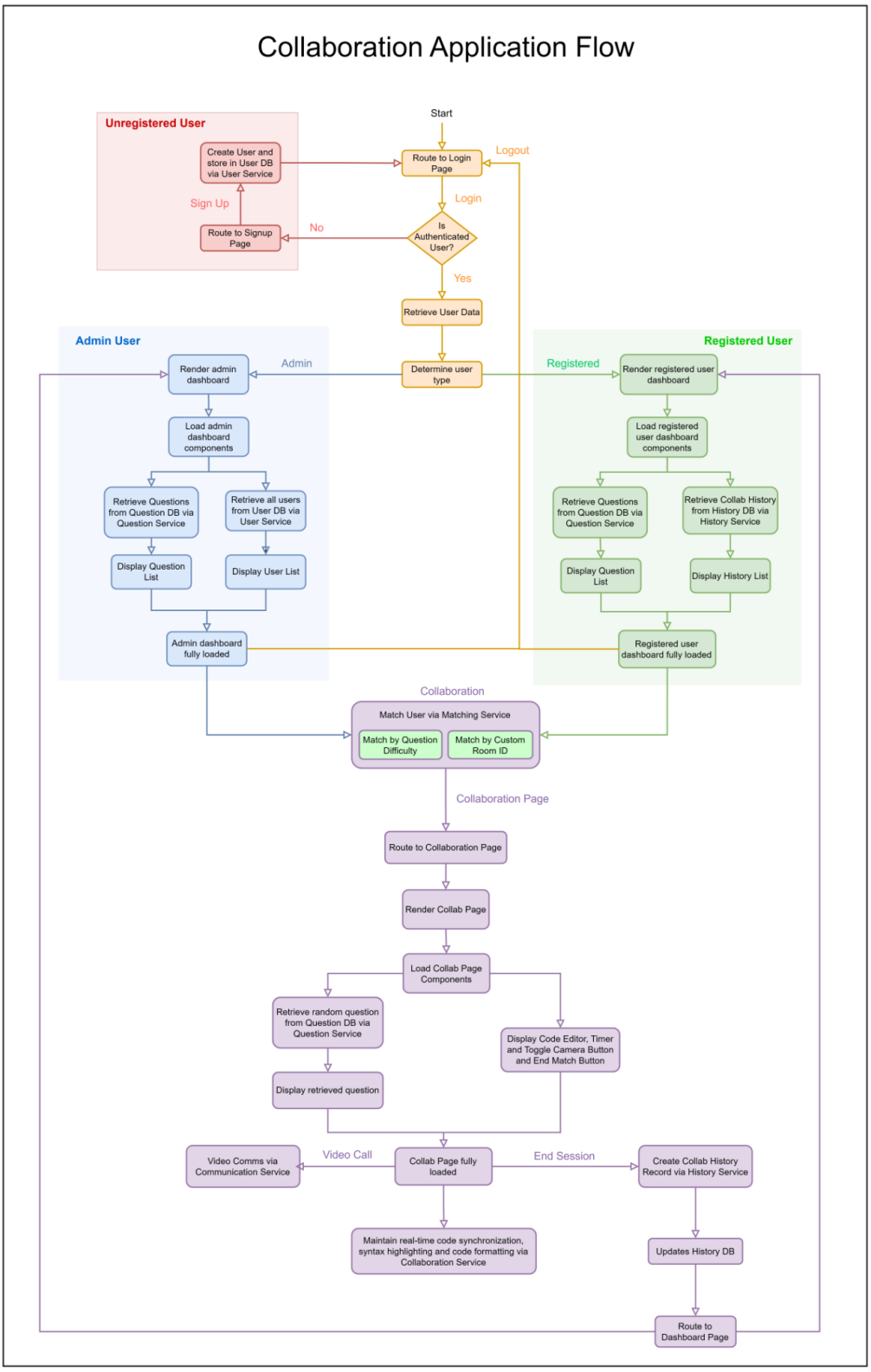

Figure 4b: Collaboration Application Flow Diagram

4.5 Database Design

The team has implemented a diverse database strategy, leveraging SQLite, PostgreSQL, and MongoDB to accommodate various models such as user, question, and history. This approach aligns with the principle of Separation of Concerns (SoC), emphasizing the isolation of different functionalities within distinct databases. This design decision contributes to maintainability, flexibility and data integrity. The intentional separation of concerns promotes clarity and ease of maintenance for each specific module, enhancing qualities such as maintainability and data integrity.

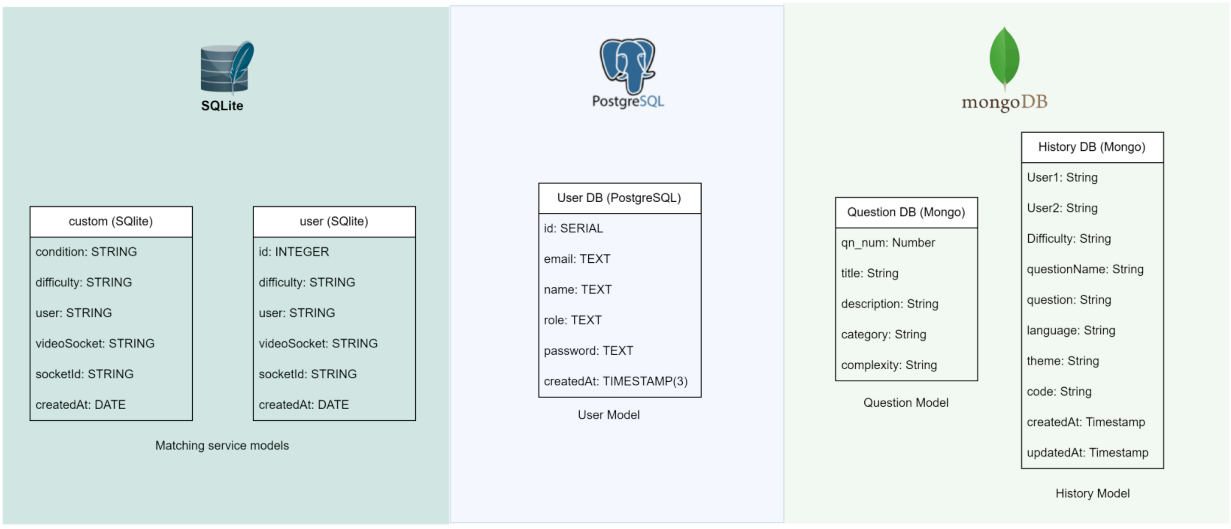

Figure 5: Database Design Diagram

4.6 Microservices

4.6.1 Frontend User Interface

The user interface is designed with the intention to prioritize the quality attributes of Usability. With that in mind, the design aims to allow the user to achieve their intended task with a minimal number of clicks.

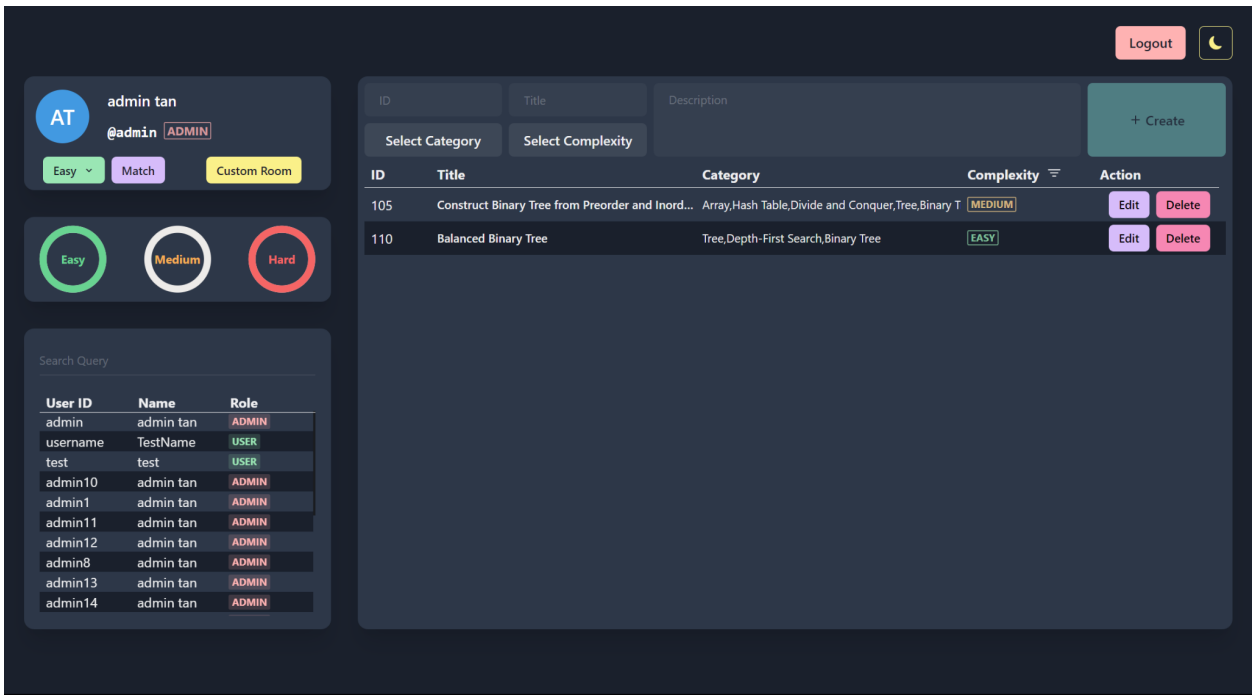

Figure 6: Dashboard UI for Admin User

However there are differences in the content presented for both admin and user accounts. As an admin, one of the roles to perform is account management. Hence, on the bottom left of the screen as seen in the above figure.

Moreover, admins should be allowed to perform higher level tasks, hence a user input field has been appended at the top of question list for admins to insert the questions.

To reduce the failure point when inserting questions, the create button will remain disabled until all the required fields are being inserted. This can be observed in Figure 7 below.

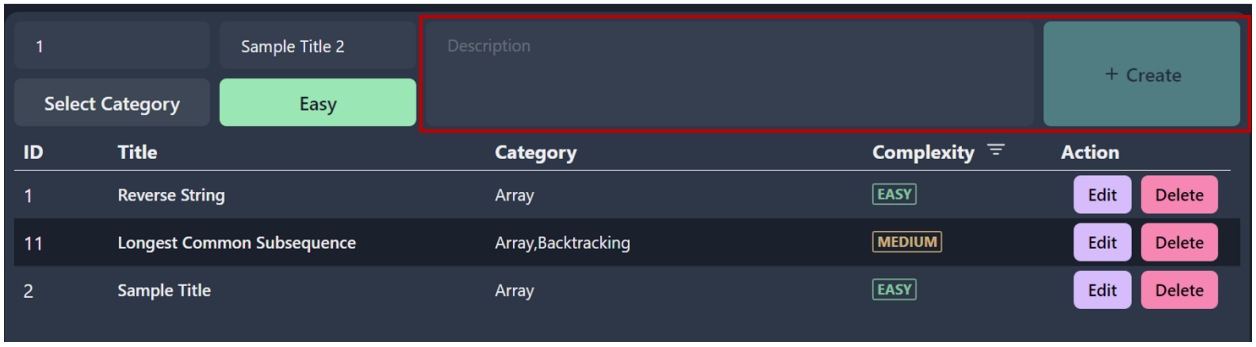

Figure 7: User Input Field (Incomplete)

Compared to admin, normal users will have their collaboration history displayed instead. Input fields are also not allowed for users to insert new questions due to access rights reasons.

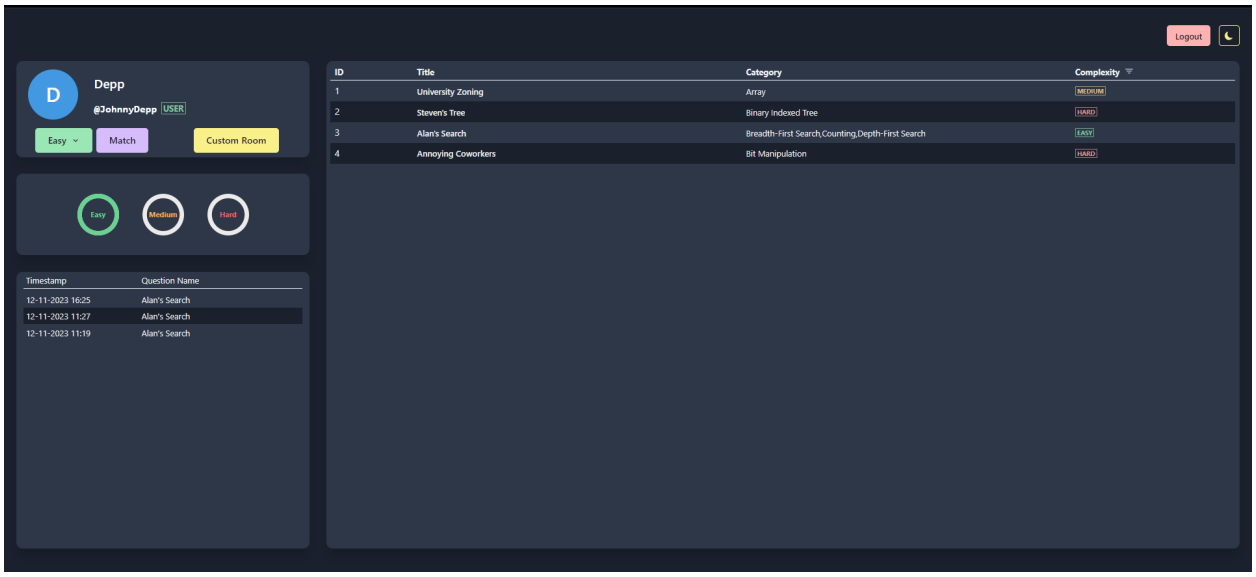

Figure 8: Dashboard UI for Registered User

The details of the questions and history are also not shown by default to prevent cluttering up the screen. Since Usability is one of the key attributes of this project, the team wants to enforce readability in the webpage.

The question descriptions and history details can be accessed by clicking on the question title and column respectively. This can be observed in Figure 9 for Questions.

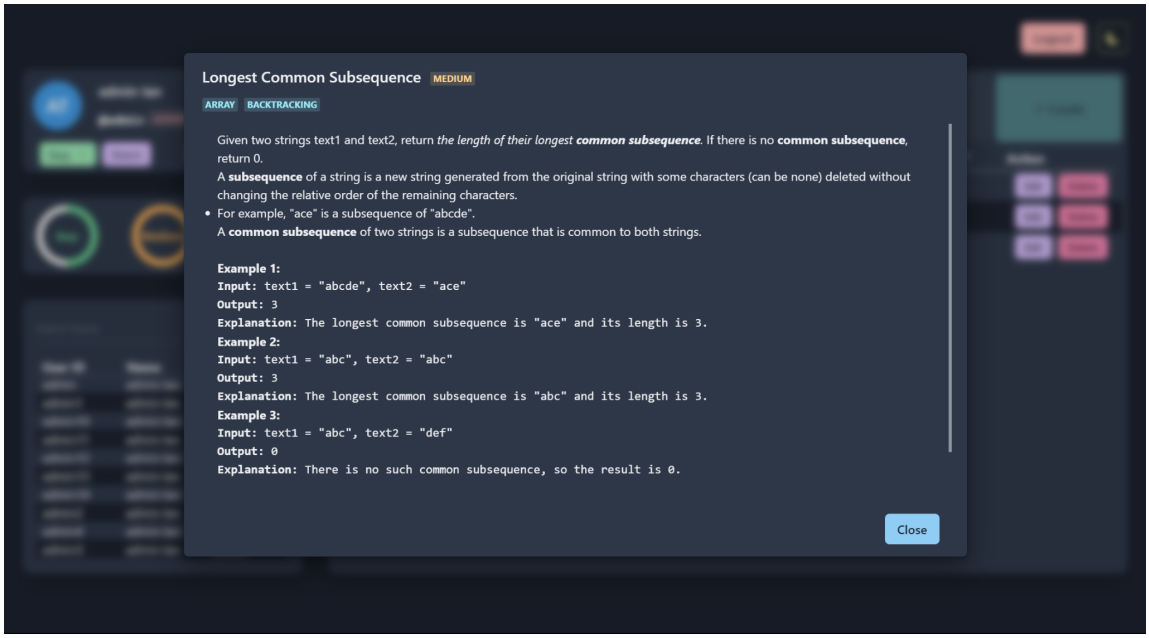

Figure 9: Question Description

The questions details are presented in a modal, formatted in HTML. In this implementation, a vertical scrollbar has also been implemented in the modal instead of the webpage. This is done to provide a better user experience of keeping the user focused. Additionally, the background has also been blurred to provide greater focus to the description. This can also be observed in Figure 10.

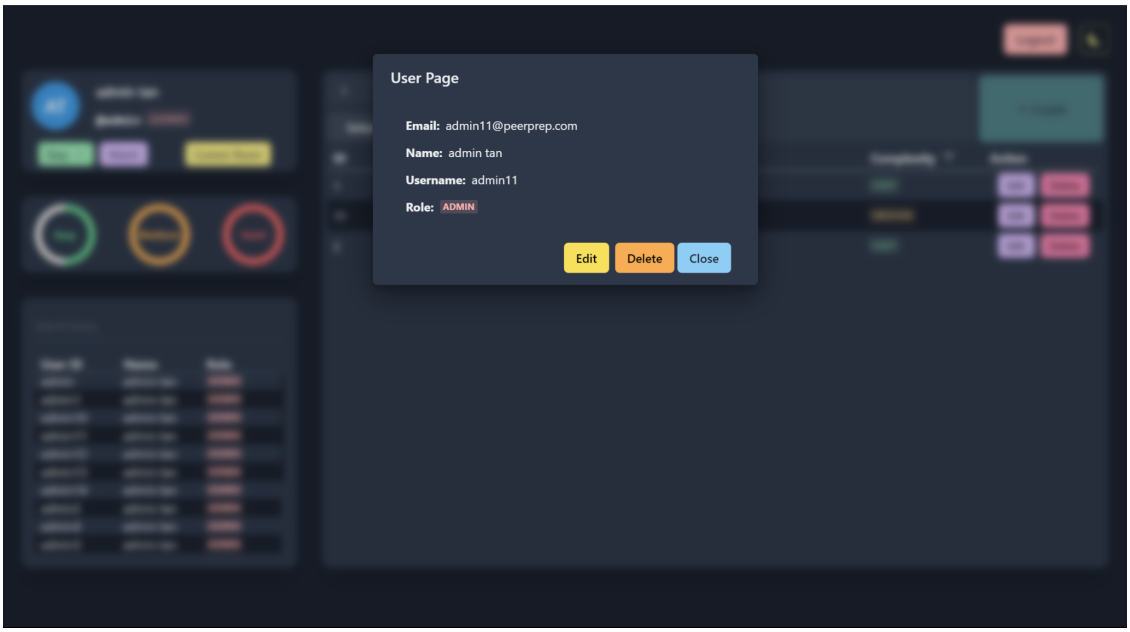

Figure 10: User Details

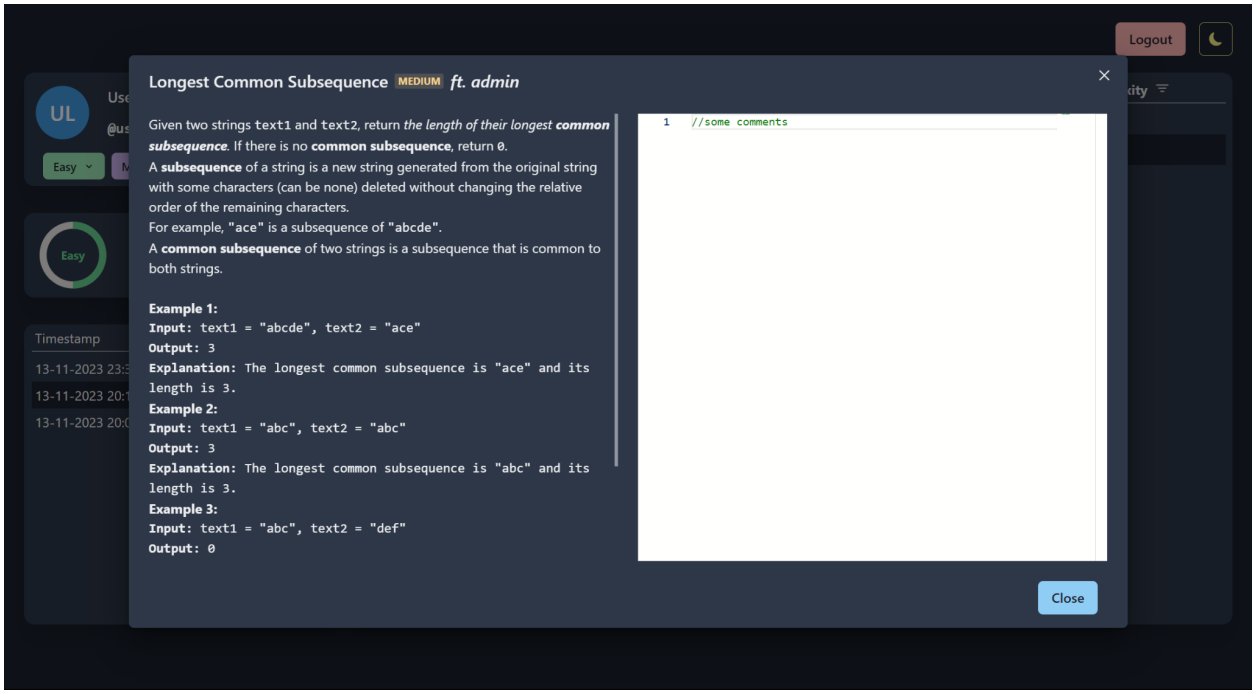

Figure 11: History Details

Similar implementation can be seen in history, where the questions attempted allows for scrolling when it exceeds the height. However, the scrolling is constrained within the modal to retain the user focus on the description, thus not diverting their attention away.

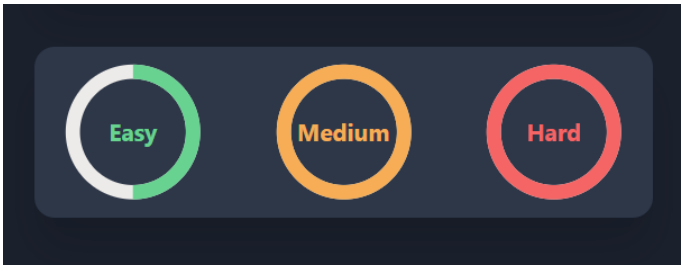

Figure 12: Questions Progress Bar

A question progress bar is implemented in the frontend to provide a visual representation of the number of questions that have been completed.

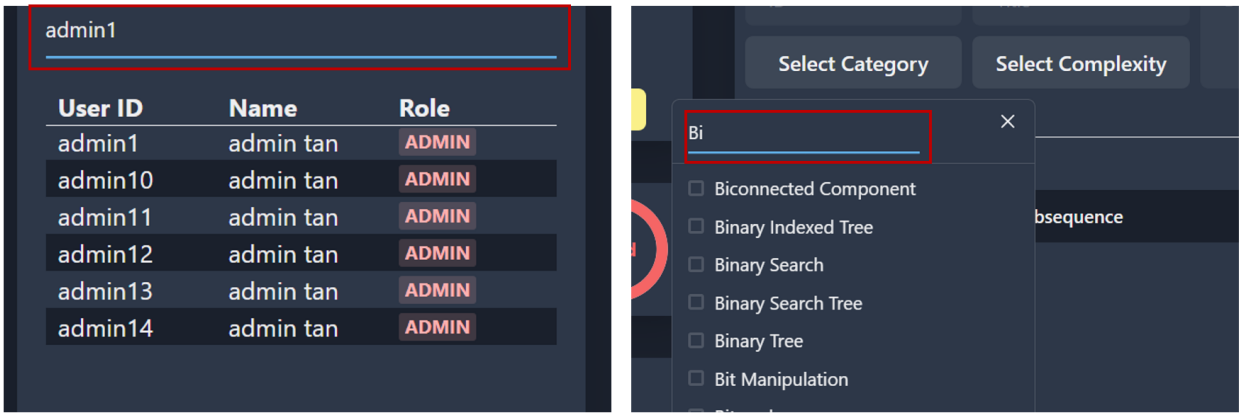

Figure 13: Search Query

The frontend has also implemented features to assist with the viewing of data when the number gets large. This feature has been implemented in Account Management and Select Category Checkbox where the number of values can get significantly large. By allowing the admin to filter out by querying, it allows a better user experience rather than iterating through large amounts of data. This implementation can be observed in Figure 13.

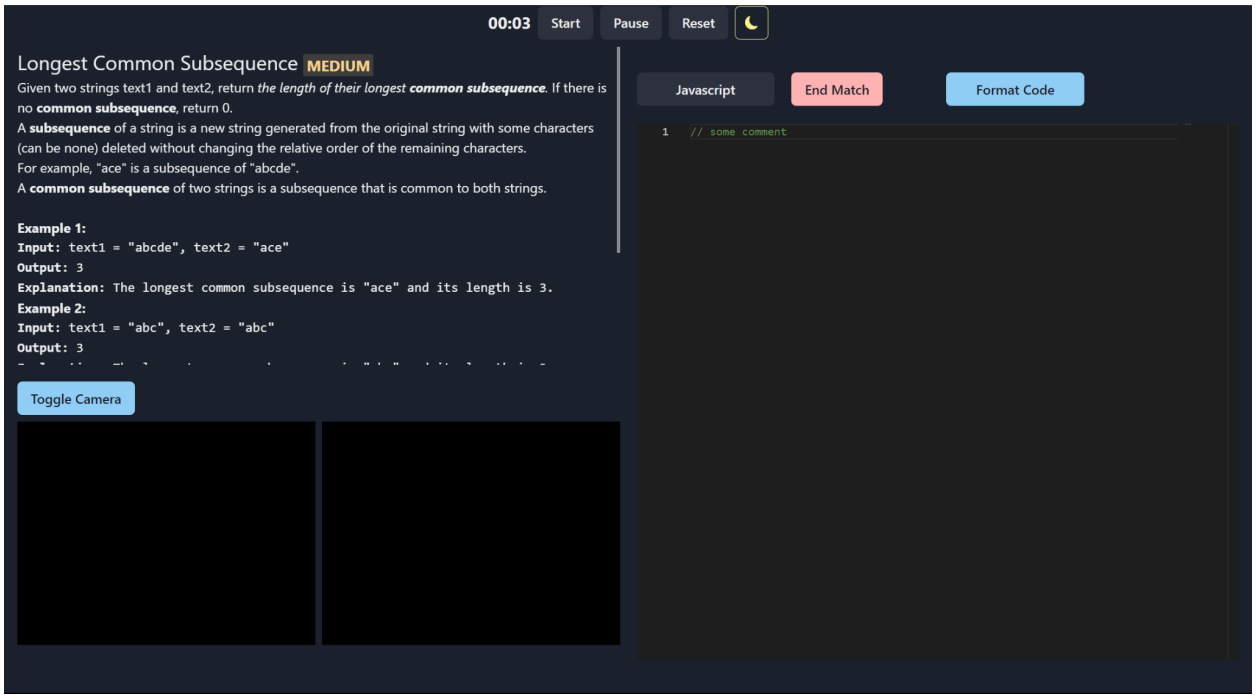

Figure 14: Collaboration Page

The collaboration page is also arranged within the perspective of the screen to allow users to see all the content without scrolling through the web pages. Since scrolling is inevitable for large

content such as the question, the scrolling is minimized and constrained within the question component itself.

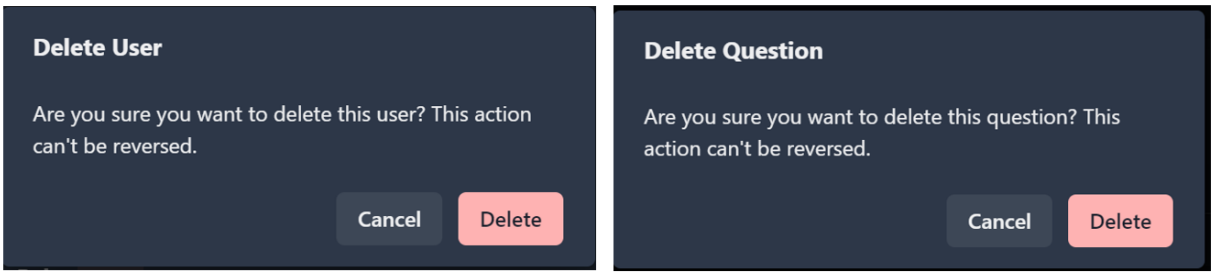

Figure 15: Alert Dialog for Deletion

The team has also taken into consideration when performing tasks such as deletion. Since deletion cannot be reversed, a confirmation alert dialog has been integrated to prevent accidental deletion. This can be observed in Figure 15.

4.6.2 User and Authentication Microservices

User Service is responsible for all actions relating to user accounts. All functionality that requires user information will have to be routed through User Service. It uses a Relational Database, PostgreSQL, to manage user accounts.

Authentication Service works with the API Gateway to authenticate and authorize all incoming requests. Authentication Service makes use of User Service to determine the validity of a user as well as their level of authorization. It uses JWT and BCrypt to enforce security for the PeerPrep application.

4.6.2.1 Tech Stack

The chosen technology stack and componentes for the user service includes:

Technology / Component	Role
Backend (Server)	Manages server-side logic and communication
PostgreSQL (database)	Relational database for storing user accounts

Table 15: User Service Tech Stack

The chosen technology stack and componentes for the authentication service includes:

Technology / Component	Role
Backend (Server)	Manages server-side logic and communication
BCrypt	Hash password for storage
JWT	Authenticates and Authorizes users
NGINX	Routes request to Authentication Service

Table 16: Authentication Service Tech Stack

4.6.2.2 Tech Stack Choice and Justification

Justification for User Service's tech stack.

Technology Choice	Role
Frontend-Backend Communication	
Socket.IO vs RESTful API	Considering the nature of User Service, the team does not expect requests to be made continuously from each client. However, since User Service is central to Peerprep, it is expected that many clients will query User Service. Hence, RESTful API is chosen for the following reasons:
	• **NF5.1 - Concurrent Users** • **NF5.3 - Component Adaptability** • **NF4.2 - Response Time**
	By using RESTful API, the User Service can be kept stateless. • **NF5.1** This brings the benefits of Horizontal Scaling that helps us to handle concurrent users. • **NF5.3** Being stateless can replace an instance of User Service without any problem. For example, if a pod crashes or performing rolling updates • **NF4.2** Using Horizontal Scaling can also ensure that our

	performance is maintained by keeping the User Service instance at a certain load.
PostgreSQL (Relational DB)	
	PostgreSQL was chosen for its alignment with the project's objectives and NFRs:
	Goal: Database Reliability, Familiarity
	NF7.2: Data Consistency
	PostgreSQL fulfills data reliability and consistency as it is an ACID (Atomicity, Consistency, Isolation, Durability) compliant database. What makes PostgreSQL standout from other ACID compliant databases is that most of our members are familiar with PostgreSQL.

Table 17: User Service Tech Stack Justification Table

Justification for Authentication Service's tech stack.

Technology Choice	Role
Frontend-Backend Communication	
Socket.IO vs RESTful API	Authentication Service also validates each request made to the Gateway so it expects even greater traffic than User Service. Hence, RESTful API is chosen for the following reasons:
	• **NF5.1 - Concurrent Users** • **NF5.3 - Component Adaptability** • **NF4.2 - Response Time**
	By using RESTful API, the team can keep our User Service stateless. • **NF5.1** This brings the benefits of Horizontal Scaling that helps us to handle concurrent users. • **NF5.3**

	Being stateless, the team can replace an instance of User Service without any problem. For example, if a pod crashes or performing rolling updates ● **NF4.2** Using Horizontal Scaling the team can also ensure that application performance is maintained by keeping the Authentication Service instance at a certain load.
API Gateway	
NGINX	NGINX was chosen for its alignment with the project's objectives and NFRs:
	Goal: Separate authentication from other services
	● **NF6.1:** User Registration for Access ● **NF5.1**: Concurrent Users ● **NF5.2**: Concurrent Collaborations
	NGINX supports the `auth_request` directive which allows us to extract authentication/authorization from other services. It forwards requests to Authentication Service to validate before forwarding to the intended service. ● **NF5.1** NGINX is mature, widely used and known for its high performance. Hence, a good choice to handle high number of concurrent user ● **NF5.2** NGINX as a reverse proxy supports websocket which is central to our collaboration functionalities. Together with its high performance, it helps to satisfy NF5.2.

Table 18: Authentication Service Tech Stack Justification Table

4.6.2.3 Architecture Diagram

Requests to API Gateway will be first forwarded to Authentication Service which will verify the token before telling the gateway to reject or proceed with the forwarding. There are some requests that do not need to be validated, for example, sign-in and sign-up. Before issuing tokens, the Authentication Service will query the User Service to check if the user is valid.

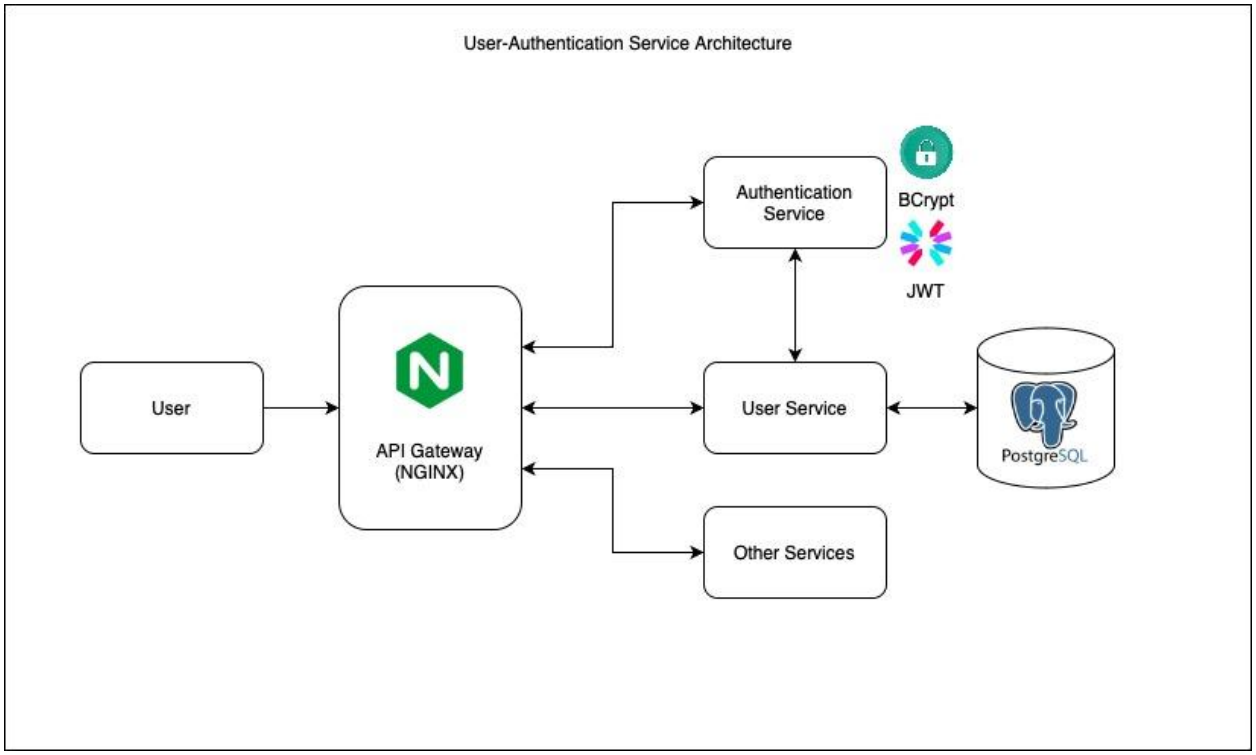

Figure 16: User and Authentication Service Architecture

4.6.2.4 Sequence Diagram

There are 3 main scenarios involving Authentication and User Service: (1) Sign-up, (2) Sign-in, (3) Validation. The other requests to User Service are part of other functionalities so the sequence diagram will be on the main three:

1. **Sign-up**
    - Component Involved: Authentication, User, API Gateway
    - Action: Sign-up
    - Description: When a user attempts to sign-up, the request is made via API Gateway to Authentication Service. The authentication will hash the password before handing it to User Service to store in the database

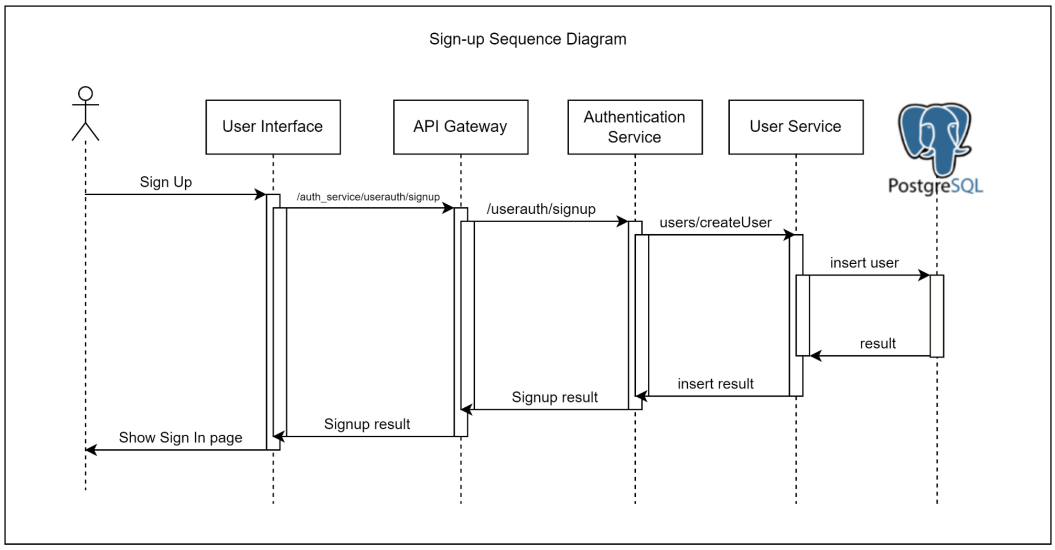

Figure 17: User Sign-up SD

2. Sign-in

- Component Involved: Authentication, User, API Gateway
- Action: Sign-in
- Description: When a user attempts to sign-in, the request is made via API Gateway to Authentication Service. As part of authentication, the Authentication Service will retrieve the user detail from User Service and verify before issuing the token

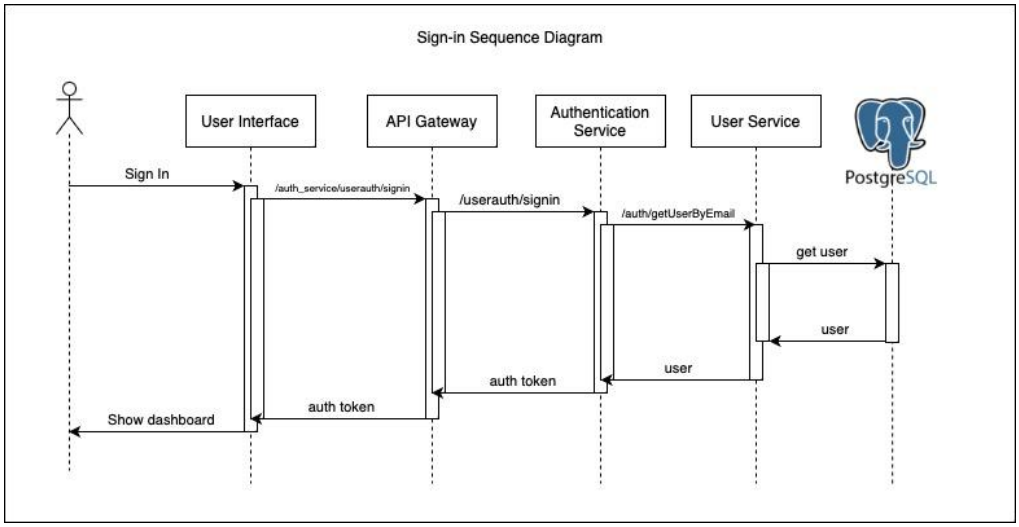

Figure 18: User Sign-in SD

3. **Validation**
    - Component Involved: Authentication, API Gateway
    - Action: Validation
    - Description: When a user makes a request to backend services, the API Gateway will forward the request to Authentication Service first. Once validated, the request is forwarded to the intended service.

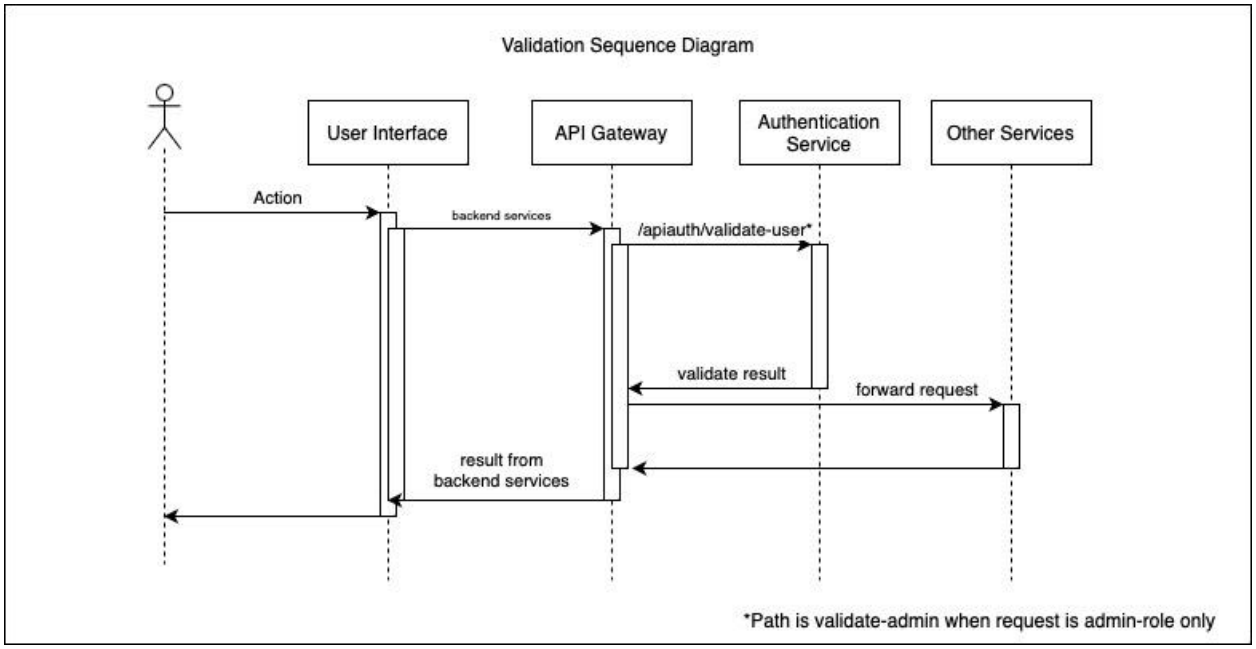

Figure 19: User Validation SD

4.6.2.5 Design Justification

Justification	Explanation
Performance	By separating Authentication and User Service, unnecessary load during validating requests to User Service can be reduced as seen in figure 19. If authentication were done for every microservices, then there is an additional computational load for each service. While the team can scale each service, the scaling only benefits the service itself. Having a centralized authentication, when scaled, benefits other services as the reduction in computation applies to all services.

Scalability	Since Authentication, User and Gateway are all stateless the team can make use of horizontal scaling autoscaler to handle increase in traffic. As the authentication / authorization is decoupled from other services, the team can develop them without being concerned about authentication / authorization.

Table 19: User and Authentication Service Design Architecture Justification Table

4.6.3 Question Service

The purpose of a question service is to provide the functionality to view coding questions to the users. Each question comprises of the following attributes:

- Question ID
- Question Title
- Question Description
- Question Category
- Question Complexity

These questions will be stored in a database and will be queried when other services such as collaboration and history services calls.

4.6.3.1 Tech Stack

The Tech Stack used throughout the implementation of question service includes the heavy use of a NoSQL database. In this implementation, the team has adopted the use of MongoDB.

To allow frontend users to make a HTTP call to the endpoint, axios library has been employed.

4.6.3.2 Tech Stack Choice and Justification

Using a NoSQL database such as MongoDB provides several advantages that coincide with the quality attributes.

Firstly, MongoDB provides features of horizontal scaling, fulfilling the quality attributes of scaling. The horizontal scaling is done through a “sharding” where data is being distributed across multiple databases in a cluster.

Secondly, through the process of sharding, the performance can also be improved through the distributed load and the stress on each of the databases. This process eventually improves the performance and helps to fulfill the quality attribute of performance.

4.6.3.3 Architecture Diagram

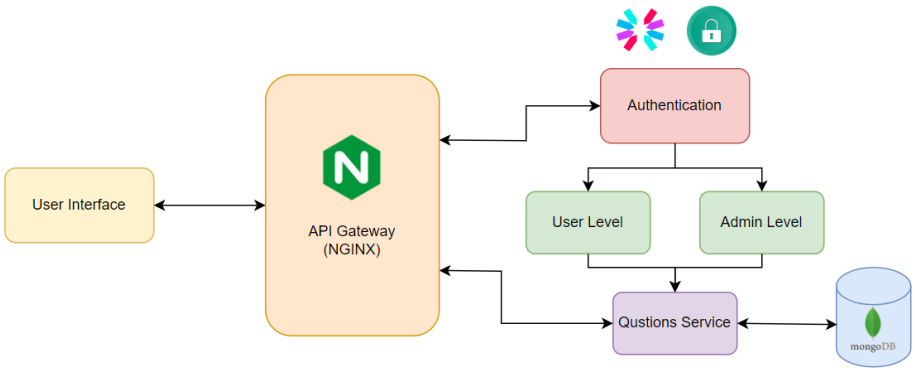

Figure 20: Question Service Architecture

4.6.3.4 Sequence Diagram

The sequence diagram shows a create question query to the database. (Create)

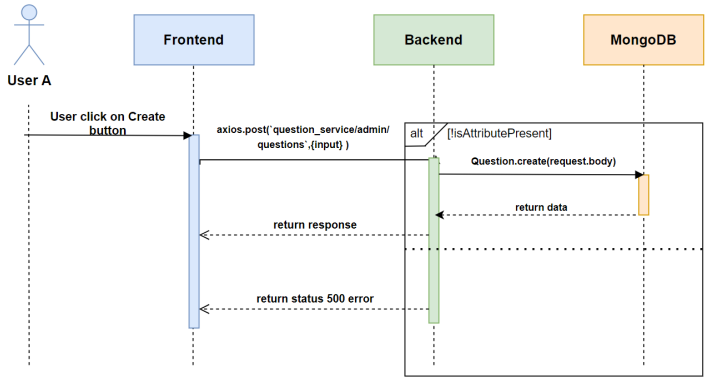

Figure 21: Question Creation SD

The sequence diagram below displays a fetch question query to the mongoDB. (Read)

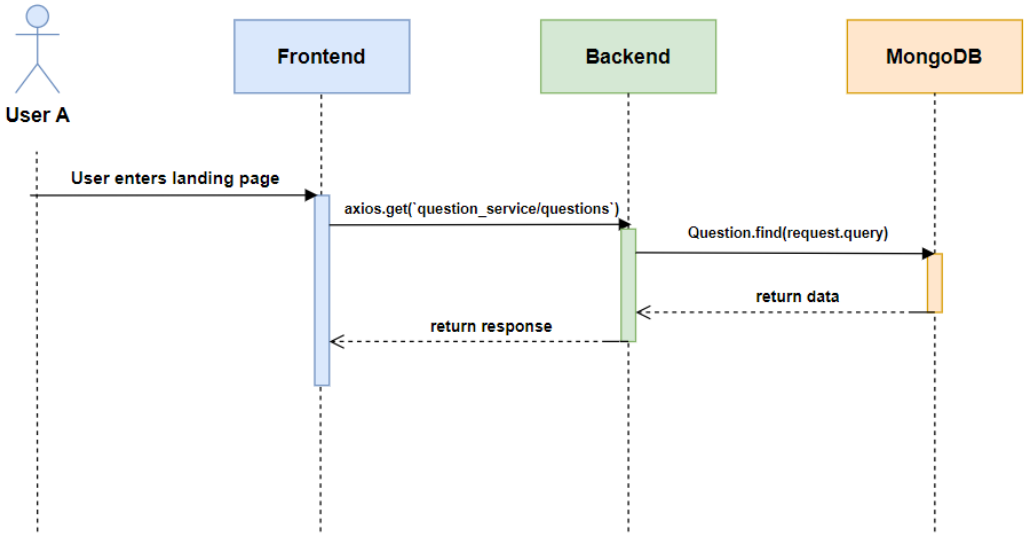

Figure 22: Question Fetching SD

The sequence diagram below updates a question query to the mongoDB. (Update)

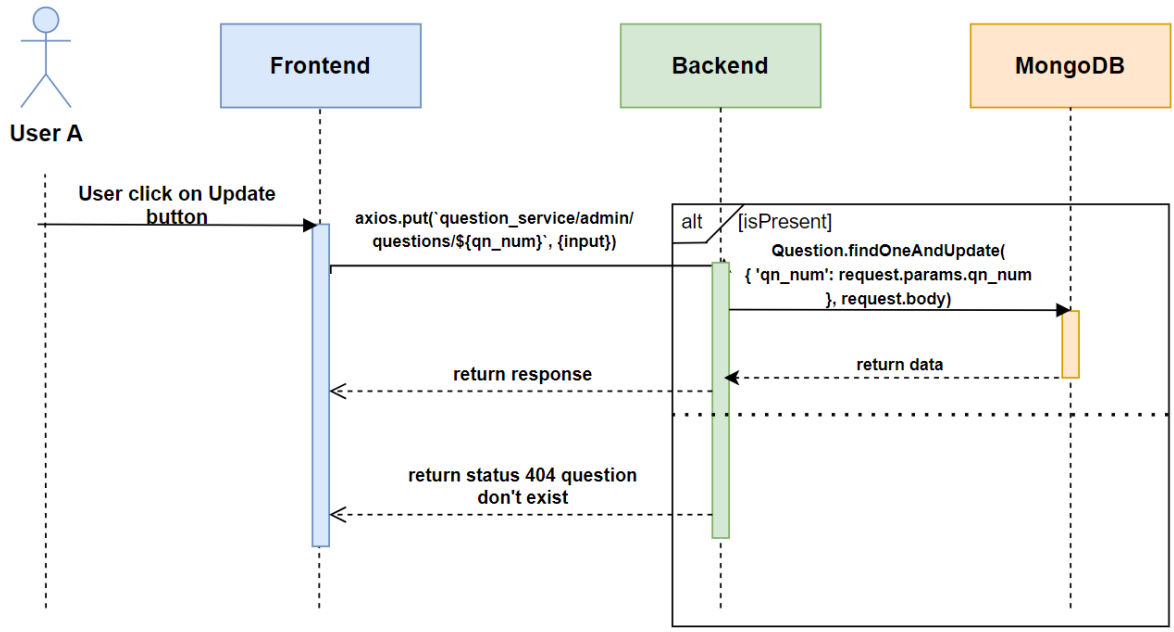

Figure 23: Question Update SD

The sequence diagram below deletes a question query to the mongoDB. (Delete)

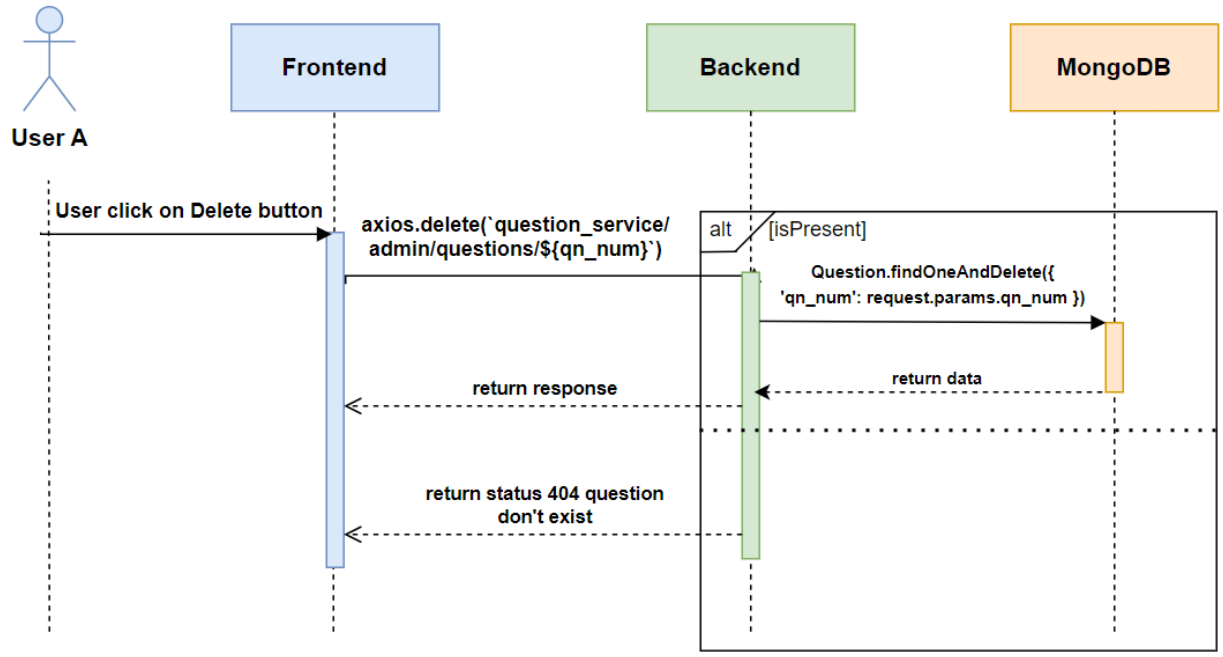

Figure 24: Question Deletion SD

4.6.4 Matching Service

The Matching Service is responsible for managing all aspects of user matching within PeerPrep. This includes the matchmaking of users based on selected difficulty levels, as well as the facilitation of match cancellations. The service operates within an event-driven architecture and relies on RabbitMQ as its message broker.

4.6.4.1 Tech Stack

The chosen technology stack and componentes for the matching service includes:

Technology / Component	Role
Frontend	Responsible for the user interface on the client side
Backend (Server)	Manages server-side logic and communication
Web Socket (Socket.IO)	Enables real-time communication between server and client

RabbitMQ	Acts as the message broker for asynchronous communication
Sequelize DB	Relational database for data storage
Matching Controller	Manages matching-related processes and logic

Table 20: Matching Service Tech Stack Table

4.6.4.2 Tech Stack Choice and Justification

In selecting the technologies for Matching Service, several factors were considered based on how well they align with the project's objectives and requirements. Below, justifications for each technology choice are shared, along with corresponding goals and NFRs that influenced these decisions.

Technology Choice	Role
Frontend-Backend Communication	
Socket.IO vs RESTful API	In a matching service, it is crucial to establish rapid real-time communication to meet the project's key objectives and NFRs. Socket.IO was chosen for the following reasons:
	• **NF3.3 - Efficient Interactions**
	• **NF4.2: Response Time**
	Socket.IO excels in real-time, bi-directional communication, offering prompt notifications for seamless user experiences during matching and match cancellations.
	On the other hand, RESTful API, designed for traditional request-response interactions, might introduce latency due to the need for clients to repeatedly poll for updates.
Message Broker Choice	
RabbitMQ vs Kafka	In a simplified matching service, RabbitMQ is the optimal choice due to the following considerations:

	• **Goal:** Efficient handling of match requests and cancellations.
	• **NF3.3 - Efficient Interactions**
	• **NF4.2: Response Time**
	• **NF5.3: Component Adaptability**
	RabbitMQ aligns with these goals and NFRs by providing low-latency real-time communication, asynchronous processing, and reliable message delivery, which ensures efficient handling of match requests and cancellations. The publish-subscribe model and message broker decoupling of RabbitMQ offer scalability, allowing components to function independently while streamlining component management and scaling. Also, due to **time constraints**, RabbitMQ which was **easier to adopt** was chosen.
Sequelize	
	Sequelize was chosen for its alignment with the project's objectives and NFRs:
	• **Goal:** Accelerate development and streamline the coding process.
	• **NF5.3: Component Adaptability**
	Sequelize offers an established community and extensive documentation, making it easier for the development team to work with. Its choice streamlines development and aligns with the project's goals of efficient development and collaboration.

Table 21: Matching Service Tech Stack Justification Table

4.6.4.3 Architecture Diagram

For Matching Service, event-driven pub/sub architecture is chosen. The architecture design and matching event role table are shown below.

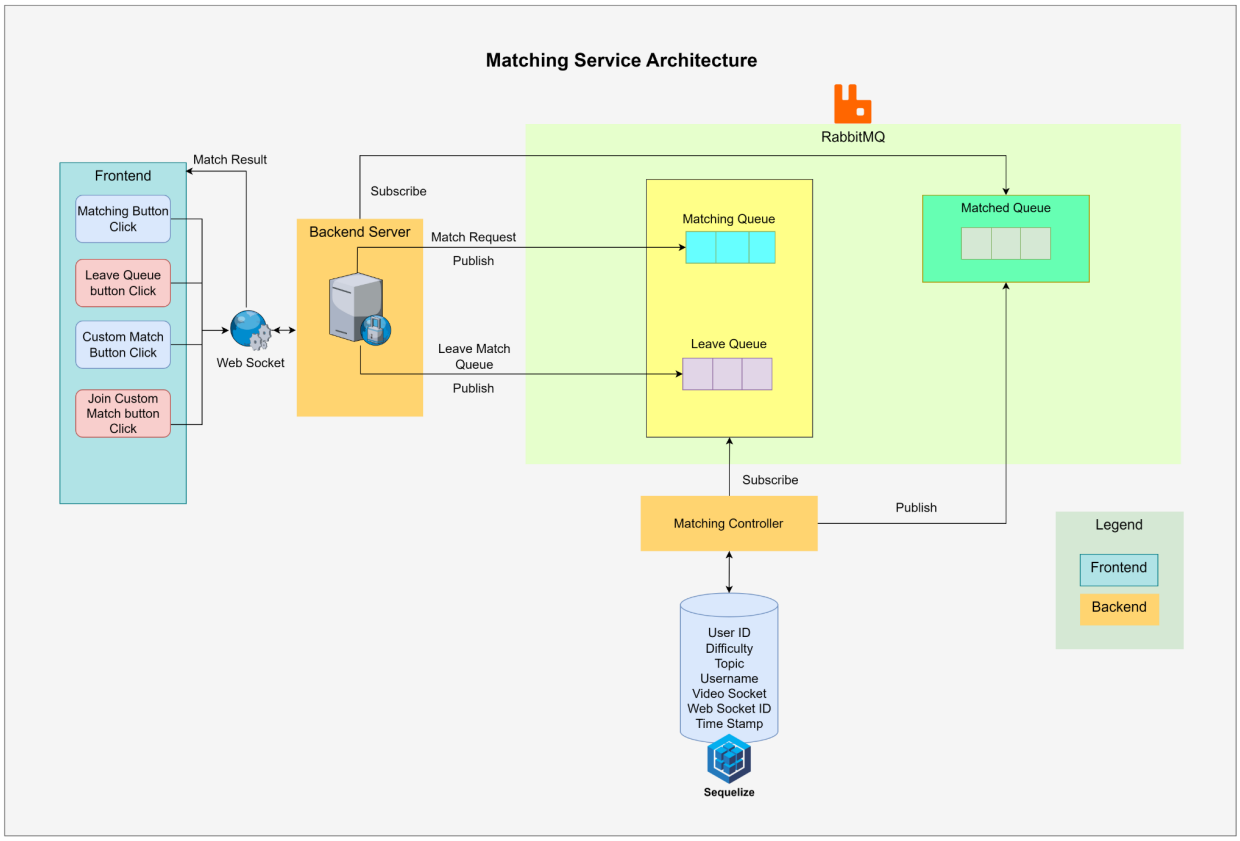

Figure 25: Matching Service Architecture

Event	Publisher	Subscriber	Message Queue (RabbitMQ)
Match Request	Backend Server	Matching Controller	Matching
Leave Queue (Cancel Match)	Backend Server	Matching Controller	Leave
Match Request Completed	Matching Controller	Backend Server	Matched

Table 22: Matching Event Role Table

4.6.4.4 Sequence Diagram

There are three possible key events during matching:

Match Request

1. UI will allow users to choose difficulty and match instances. The user will first select the question difficulty and click on the match button. The user is also allowed to create their own custom room based on their custom keyword to match with other users.
2. The user ID, selected difficulty, video socket for communication, current socket ID and custom keyword if any will be sent to the backend via socket.io.
3. Detects "match request" and publishes a message event to "Matching" queue in RabbitMQ.
4. Matching Controller subscribes to the "Matching" queue. It will consume the message where matching information is stored in the Sequelize database.

Match Request Completed

5. Then, pairing will be done based on selected difficulty by matching controller. Or in the case of custom matching, it will be done based on the custom keyword selected by the user.
6. Once pairing is completed, matching controller publishes “Matched” event to “Matched” queue in RabbitMQ.
7. Backend server will subscribe to “Matched” queue and send notification to both matched users via Socket.IO when matched.
8. If timeout 30s, socket.io will emit leave queue event which is equivalent to cancel match request.

Cancel Match

1. UI will allow users to cancel matching in the midst of matching.
2. Using socket id or custom keyword in the case of custom matching, identify matching instances through Sequelize DB and remove it.

To illustrate the three scenarios, a sequence diagram (SD) can be found below along with a legend for reference.

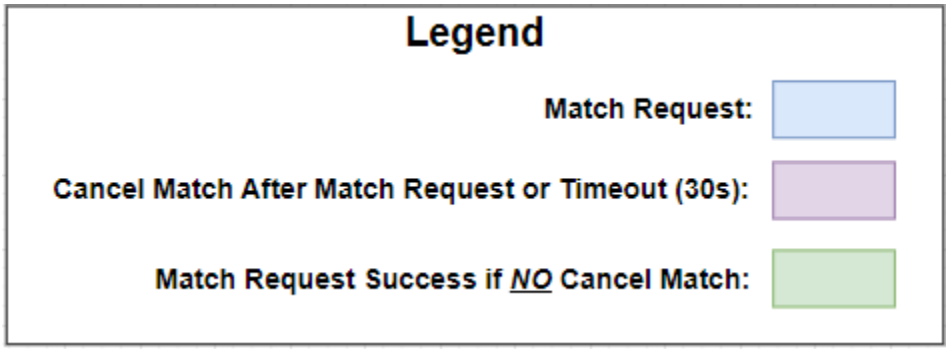

Figure 26: Legend for Matching Service SD

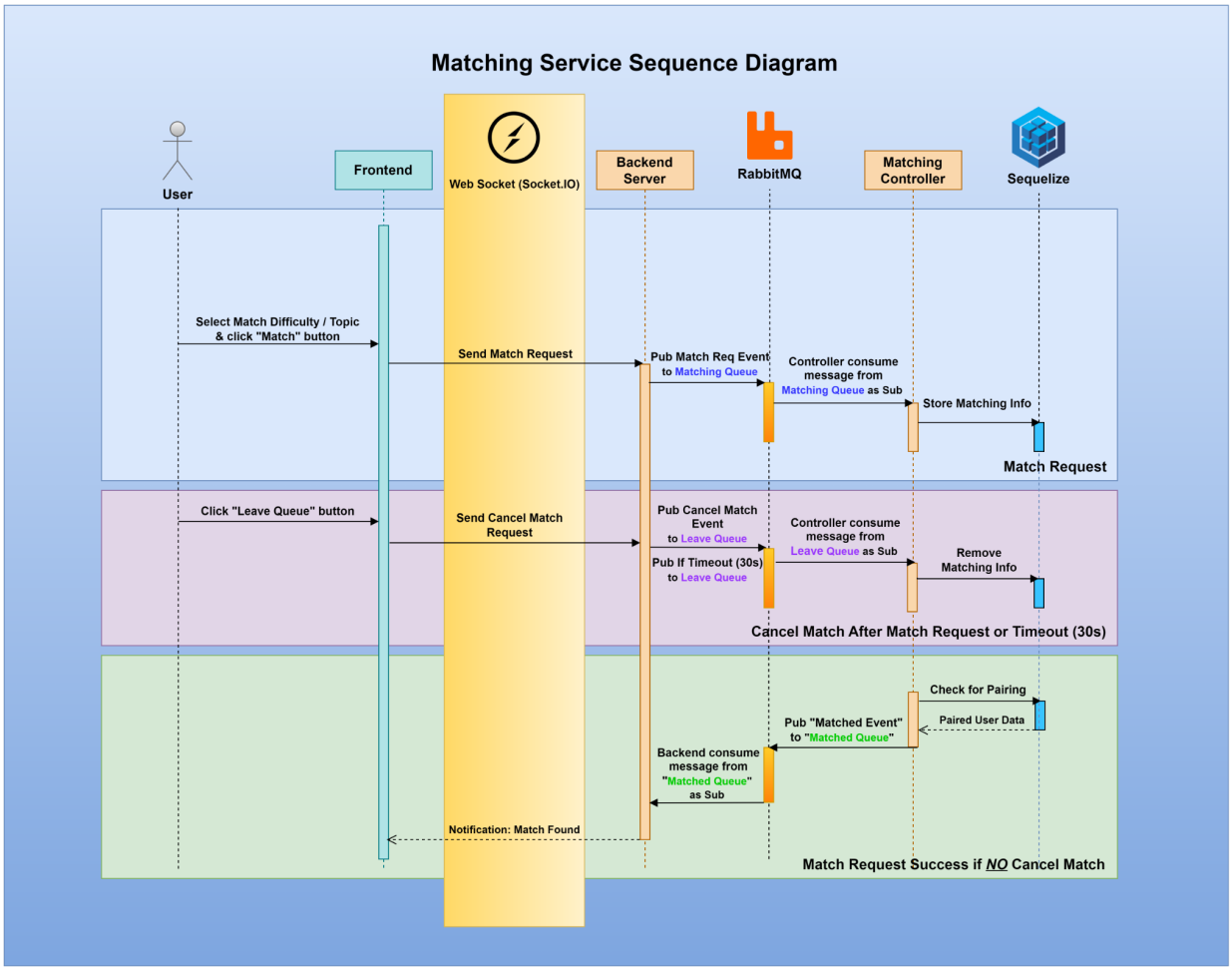

Figure 27: Matching Service SD

4.6.4.5 Design Justification

The event-driven pub/sub architecture is chosen for the following reasons:

Justification	Explanation
Performance	The event-driven architecture can contribute to improved performance in the system. It allows for asynchronous communication which aligns with the nature of matching service, reducing the need for synchronous blocking calls. This can lead to

	faster response times and better resource utilization. The pub/sub pattern helps ensure efficient message distribution to the appropriate subscribers, further enhancing performance.
Scalability	Event-driven architectures are inherently scalable. They allow for the decoupling of components, making it easier to add or remove services as needed. In this design, the use of a pub/sub pattern ensures that the system can efficiently scale to accommodate increased load. For instance, when more users want to use the service, horizontal scaling can be adopted by increasing the number of matching controllers and RabbitMQ message queues to sustain the increased load Furthermore, RabbitMQ offers topic exchange which is very compatible with the future planned feature “Match by Question Topic” mentioned later in section 5.

Table 23: Matching Service Design Architecture Justification Table

4.6.5 Collaborative Space and History Microservices

The design of the Collaborative Space involves the integration of several microservices, each serving a specific purpose in facilitating collaborative interactions. These microservices include the Collaboration Microservice (M4 and N5), the Communication Microservice with Video Call (N1) and the History Microservice (N2). In the collaborative workflow, these microservices work seamlessly to provide a comprehensive user experience, as outlined in section 2.4.

Collaboration Microservice (M4 and N5):

This microservice is central to the collaborative space, handling various aspects of real-time collaboration. It ensures synchronized code editing (M4) and manages collaboration events (N5). Users can actively collaborate on code, leveraging features such as shared code editing, language selection, and a shared timer. The Collaboration Microservice facilitates the dynamic exchange of information during collaboration sessions.

Communication Microservice with Video Call (N1):

The Communication Microservice focuses on enabling video calls (N1) within the collaborative space. It ensures seamless communication between users through video calls, enhancing the

collaborative experience. This microservice plays a vital role in supporting real-time visual communication, fostering effective collaboration between users.

History Microservice (N2):

The History Microservice is responsible for capturing and storing details of completed collaboration sessions. When a collaboration session concludes, session details are stored as historical data through the History Microservice (N2). This includes information about the users involved, the difficulty level, question details, and exchanged code during the collaboration. The stored history serves as a valuable record of past collaborative sessions.

Dashboard Integration:

When users return to the dashboard, the History Microservice retrieves and compiles past history sessions. It efficiently retrieves historical data and presents it to users in an organized manner. Users can review and reflect on past collaborative sessions, providing continuity and a sense of progress in their collaborative endeavors.

In summary, the Collaborative Space and associated microservices form a cohesive ecosystem, allowing users to engage in real-time collaboration, communicate through video calls, and maintain a historical record of their collaborative sessions. This design promotes a holistic and user-centric approach to collaborative coding experiences.

4.6.5.1 Tech Stack

The chosen technology stack and components for the collaborative space and associated microservices includes:

Applicable Microservice(s)	Technology / Component	Role
Collab Space (All)	Frontend	Responsible for the user interface on the client side
Collab Space (All)	Backend (Server)	Manages server-side logic and communication
Collaboration, Communication	Web Socket (Socket.IO)	Enables real-time communication between server and client
Video Communication	Peer	Facilitates peer-to-peer connections for video calls and real-time communication. Also, complements Socket.IO by providing the necessary infrastructure for establishing direct connections between clients.

History	MongoDB	Oversees the storage (when collab session ends) and retrieval of collaboration session details (dashboard), including timestamps, completed questions and code history, ensuring a comprehensive repository for users to review past sessions

Table 24: Collaborative and History Services Tech Stack Table

4.6.5.2 Tech Stack Choice and Justification

In selecting the technologies for collaborative space microservices, several factors were considered based on how well they align with the project's goals and requirements. Below, justifications for each technology choice are shared, along with sample project goals and NFRs that influenced these decisions.

Technology Choice	Role
Frontend - Backend Collaboration Server; Frontend - Backend Communication Server	
Socket.IO vs RESTful API	During the collaboration session, it is crucial to establish rapid real-time communication to meet the project's key objectives and NFRs. Socket.IO was chosen for the following reasons:
	Goal: To ensure rapid response times for user interactions such that there such be minimal latency during concurrent coding in code editor.
	NF4.2 - Response Time: Socket.IO excels in real-time, bi-directional communication, ensuring prompt notifications for seamless user experiences in collaborative spaces. This is crucial when components, such as code editor, language selection for syntax highlighting, shared timer, and code formatting, need to sync for both users. Additionally, Socket.IO enables immediate user notifications when someone leaves the room. These features underscore the necessity of fast, real-time communication that can be skillfully facilitated by Socket.IO.

	On the flip side, the RESTful API, tailored for conventional request-response interactions, is commonly utilized for data retrieval or storage. However, its use in the collaborative space context might introduce latency, as clients are required to repeatedly poll for updates, making it less suitable for real-time collaboration.
Frontend - Backend History Service	
Socket.IO vs RESTful API	**Scenario 1:** The history service comes into play when the collaborative session has ended.
	Scenario 2: Collaboration history data is retrieved and displayed on the dashboard for registered users.
	Consideration: For scenarios where the emphasis is on data storage or retrieval of collaboration history details, using a RESTful API may be a better choice. RESTful APIs are well-suited for handling data storage and retrieval operations without the need for real-time communication.
Video Communication Service	
Peer	**NF3.3 - Efficient Interactions:** Direct data transfer between peers promotes efficiency, reducing delays in collaborative actions. This contributes to a responsive and smooth user experience, enhancing the overall usability of the collaborative space
	NF4.2 - Response Time: By providing a direct peer-to-peer communication channel, Peer helps ensure that interactions, such as video calls or data sharing, have low response times. This aligns with the requirement of keeping response times within 1 second for each type of interaction.
	NF5 - Scalability: As new users join a collaborative session, Peer allows for dynamic establishment of connections between peers. This

	adaptability can contribute to scalability, allowing the system to handle varying numbers of participants without significant changes in the underlying architecture which will be mentioned later in section 5.
History Service	
MongoDB	
	NF4.2: Response Time: MongoDB's efficient handling of data, especially in scenarios involving collaboration history, contributes to maintaining a low response time. The ability to quickly retrieve and store collaboration history details meets the NFR of response time not exceeding 1s for each type of interaction.
	NF5.3: Component Adaptability: MongoDB supports flexible schema design, allowing for the addition or removal of components without disruptions. This aligns with the NFR.

Table 25: Collaborative and History Services Tech Stack Justification Table

4.6.5.3 Architecture Diagram

In the collaborative space, an event-driven architecture characterized by choreography is implemented. This architectural paradigm embraces a choreographed communication model, facilitating the decoupling of various backend microservices, the frontend and MongoDB. The essence of this architecture lies in its ability to ensure effective communication and integration among these components, fostering a dynamic, responsive, and loosely-coupled environment for users. By embracing choreography, the system achieves adaptability, scalability, and efficient responses, contributing to an enhanced user experience in real-time collaborative scenarios. Below is a simplified architecture diagram where blue represents frontend and orange represents collaborative space microservices:

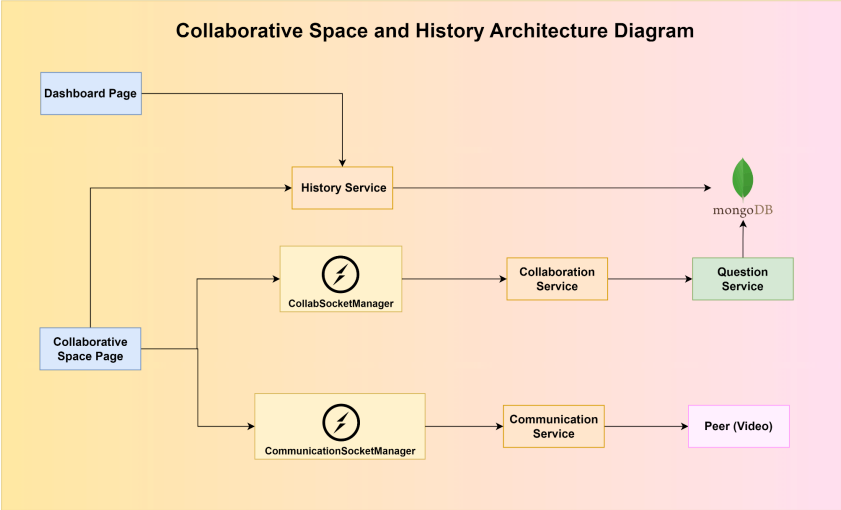

Figure 28: Simplified Collaborative Space and DashBoard Collaboration History Architecture

4.6.5.4 Sequence Diagram

The collaborative space is made up of four main components: (1) Question, (2) Shared Timer Feature, (3) Video Call and (4) Code Editor. Thus, the following sequence diagrams will be decomposed based on the four main components.

1. **Initial Page Enter:**
    - Component Involved: Question
    - Action: Fetch Question
    - Description: When paired users first enter the collaborative space, a random question of selected difficulty is fetched from the question repository. This fetched question is then displayed on the frontend.

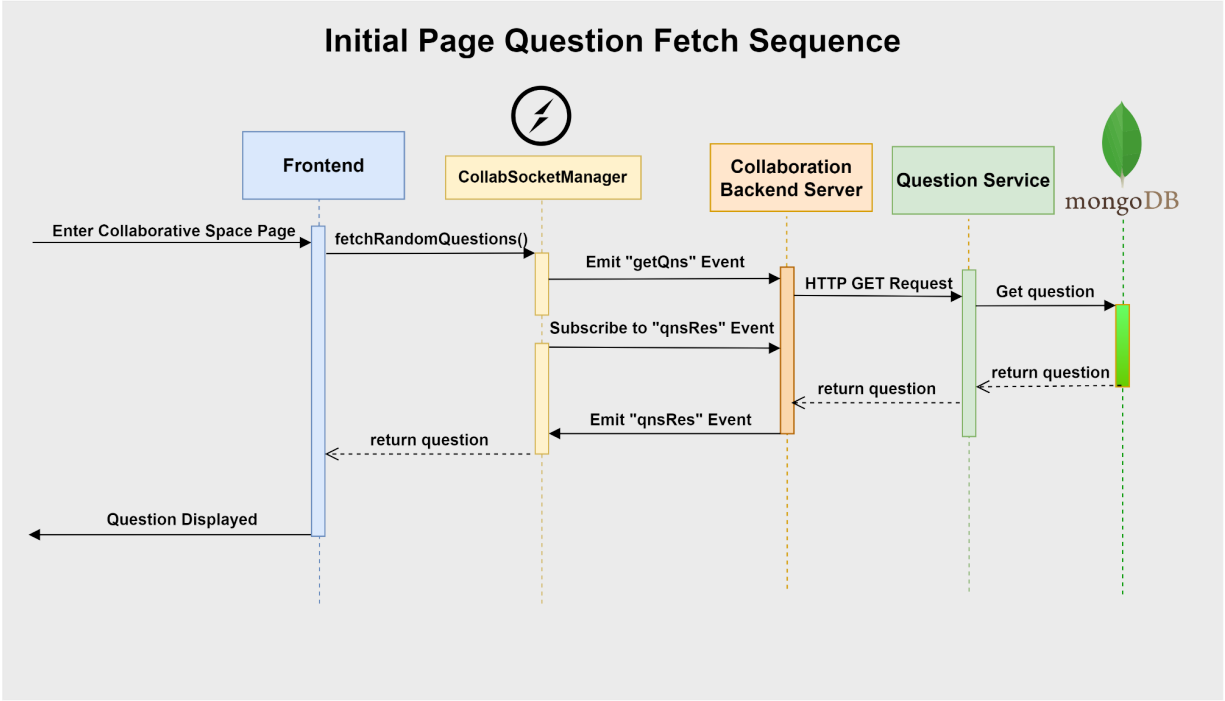

Figure 29: Initial Page Question Fetching SD

2. **During Collaboration:**
    - Components Involved: Shared Timer Feature, Video Call, Code Editor
    - Description: This section involves the ongoing collaboration process, including managing the code editing, shared timer and video call functionalities.
    - Actions:
        - Code Editor:
            - Code Change: Notifies other users about changes made to the code.
            - Format Code: Applies formatting changes to the code.
            - Language Change: Informs other users about a change in the coding language.
        - Shared Timer Feature:
            - Start Timer: Initiates the countdown for the shared timer.
            - Pause Timer: Pauses the ongoing timer.
            - Reset Timer: Resets the timer to its initial state.
        - Video Call:
            - Call User: Initiates a video call with the specified user.
            - Answer Call: Accepts an incoming video call.

Note: Below are the sequence diagrams for the different actions where user A is the action taker while user B receives all the feedback or response from the application. In actual setting, the sequence is applicable in both directions.

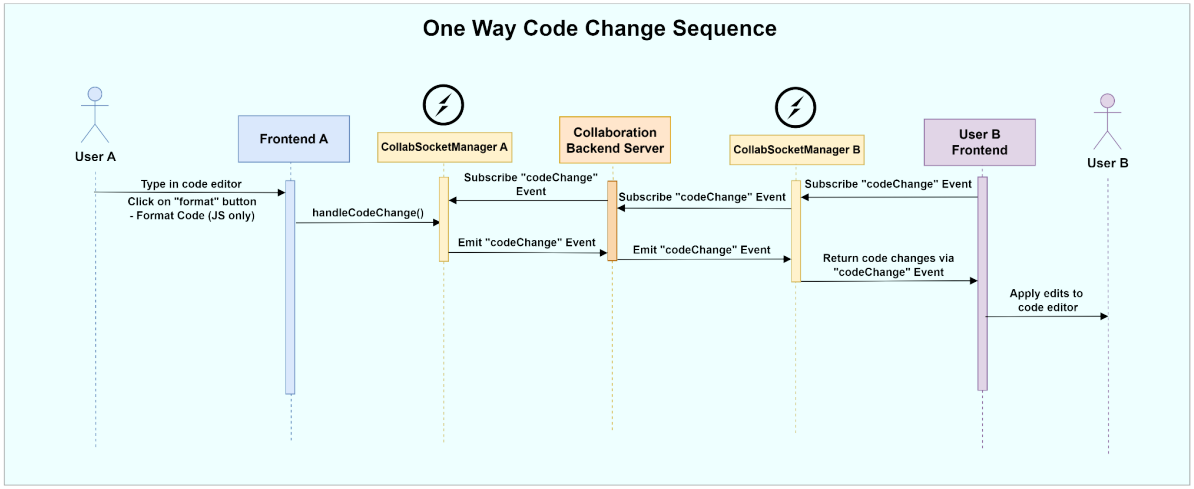

Figure 30: Code Change / Format Code SD

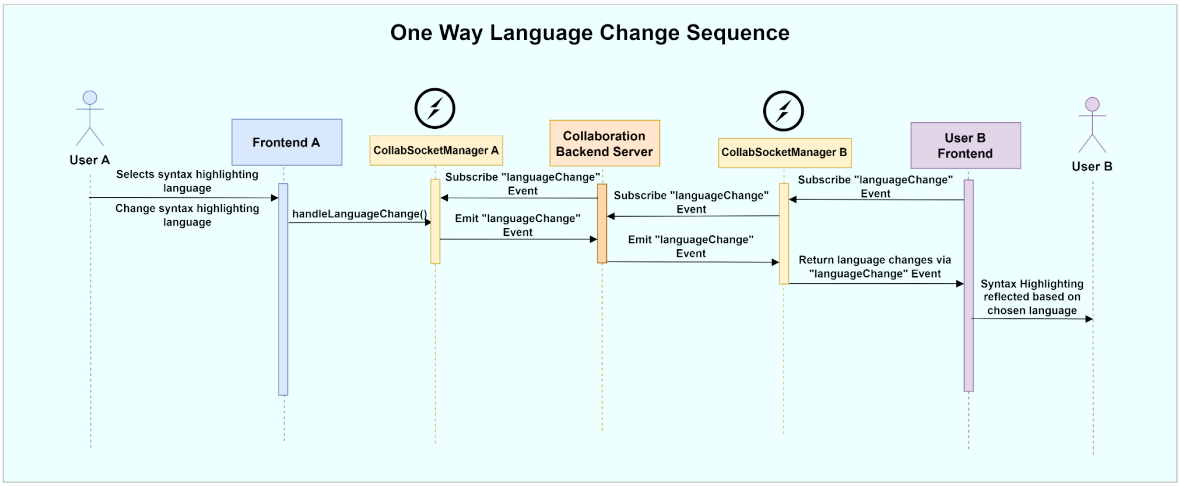

Figure 31: Language Change SD

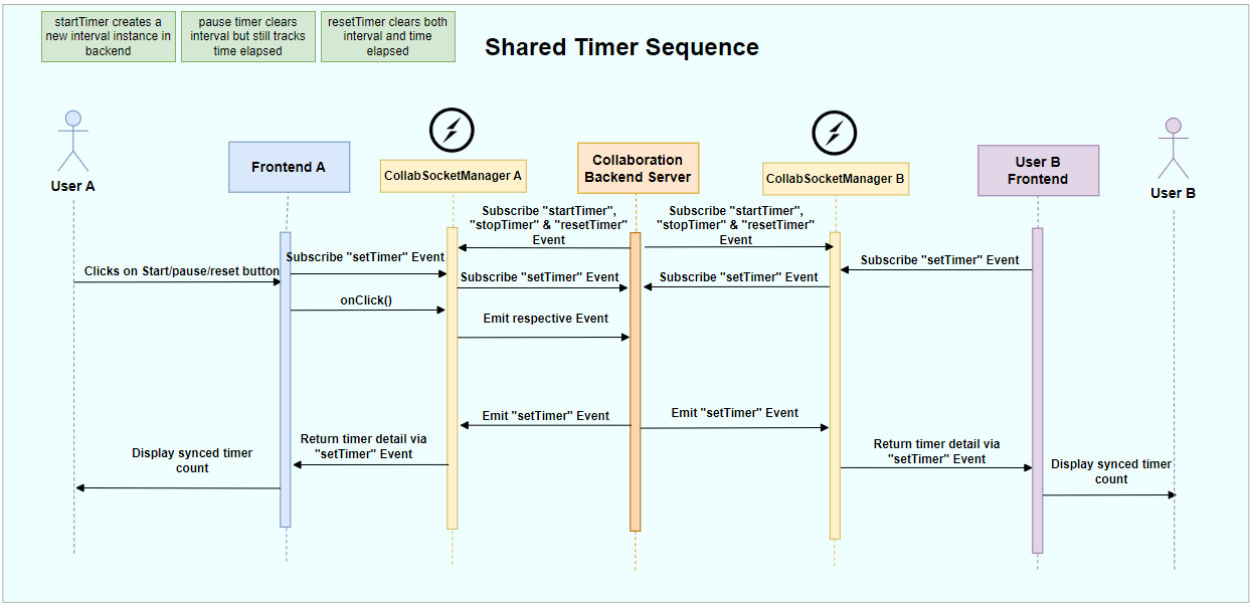

Figure 32: Shared Timer SD

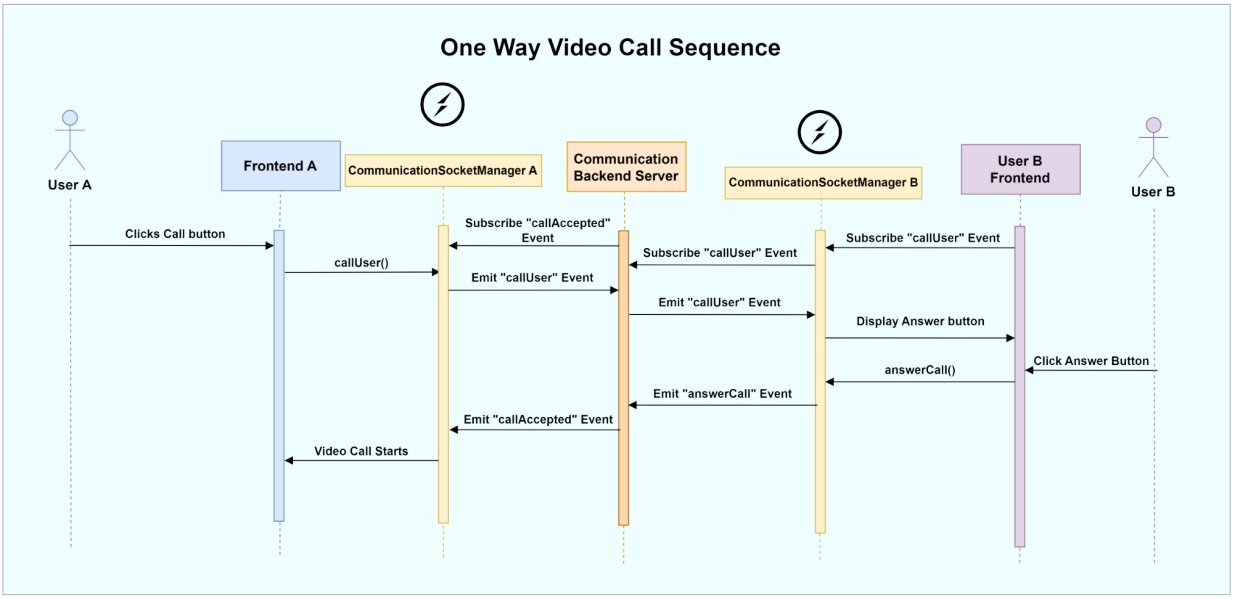

Figure 33: Video Call SD

3. **End Collaboration:**
    - Action: End Match
    - Description: When collaboration ends, the **handleHistory** function is triggered. This action involves saving relevant information about the collaboration session through history service, such as users involved, difficulty level, question details, and the code exchanged during the collaboration.

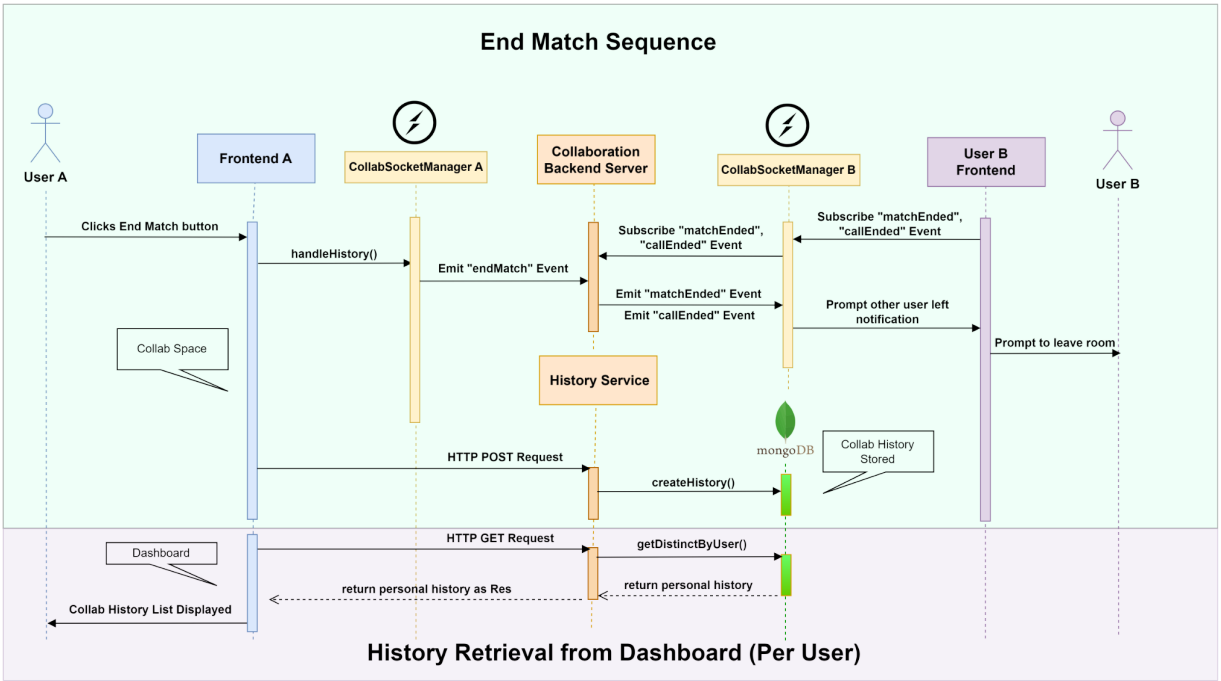

Figure 34: End Match SD

4. **History Retrieval from Dashboard:**
    - Action: Get personal collaboration history list
    - Description: When user returns to dashboard, application will retrieve history list via history service as illustrated in above sequence diagram.

4.6.5.5 Design Justification

The design decisions for the collaborative space service prioritize delivering a smooth and user-friendly experience, and the chosen architecture aligns with this objective. The decision to use an event-driven architecture is rooted in the need for asynchronous communication, enabling seamless interaction between users engaged in collaborative activities such as video calling, code editing, and timer setting changes.

Below is a justification table for the chosen design architecture.

Justification	Explanation

Usability	Event-driven architecture facilitates asynchronous communication, allowing actions on one user's side to be efficiently communicated and reflected on the other user's side. This promotes a real-time and responsive user experience.
Performance	Events enable immediate and live feedback between users. For instance, changes in code, timer settings, or video call status are communicated instantly, fostering a sense of collaboration and real-time interaction. Also, asynchronous communication minimizes delays and enhances performance. Users can perform collaborative actions without waiting for synchronous responses, leading to a smoother and more efficient interaction. Furthermore, the event-driven system enables dynamic responses to user actions. For example, changes in code are immediately broadcasted, and timer updates are communicated in real-time, ensuring that all users are synchronized with the latest state of the collaborative session.
Scalability	Event-driven systems are inherently loosely coupled. This loose coupling enhances scalability by allowing components to evolve independently, and new features can be added without significant impact on existing functionalities. The decoupling of components through events also allows for flexibility in how different parts of the collaborative space interact. Components can subscribe to events relevant to their functionalities, promoting a more flexible and adaptable system. This simplifies the process when adding new features to the collaborative space.

Table 26: Collaborative and History Services Architecture Design Justification Table

4.6.7 Deployment

The project is designed to be cloud-native, using Docker and Kubernetes technologies. Amazon Web Services was chosen to be our cloud provider, as they provide a conducive environment for developers to launch scalable applications with appropriate logging and security features.

Moreover, the team employs Continuous Deployment principles to enhance reliability and speed in our development process.

4.6.7.1 Tech Stack

The chosen technology stack for the application deployment includes:

Technology	Role
Docker	Enables containerisation of our code and provides a container engine to execute our containerisation code.
Kubernetes	Enables container orchestration. This allows us to easily scale up and increase performance of our project
Github Actions	Enables Continuous Deployment of our project, to improve reliability and development speed.
Amazon Elastic Container Registry	To securely and reliably store our Docker images for use by our cloud-based Kubernetes.
Amazon Elastic Kubernetes Service	Cloud-based Kubernetes provider. Enables us to run Kubernetes on top of Amazon EC2 instances.
Serverless Framework	Framework to automate the deployment of our serverless function.
Amazon Lambda	Serverless Function Provider. Enables us to host and invoke serverless functions on demand.

Table 27: Application Deployment Tech Stack Table

4.6.7.2 Tech Stack Choice and Justification

Justification for application deployment tech stack.

Technology Choice	Role
Container Engine	
Docker	To containerize our code, the team chose Docker as the project's container engine.
	Goal: To increase reliability and scalability, the team aims to

	support a microservices architecture. This was done by ensuring that each of the microservices are abstracted out into independent containers.
	NF5.3 - Component Adaptability Ensure that each microservice can be optimized independent of each other
	NF5: Scalability - Containers are designed to be easily scalable, which allows us to increase reliability and performance.
	The project uses Docker to containerise each microservice and the API Gateway. Each container is manually assigned with a specific outward facing port and communicates with the client through the API Gateway. In order to support development testing, the team also uses Docker Compose. This allows us to test containers in concert with each other.
Kubernetes	To support container orchestration, the team chose Kubernetes as the project's container orchestrator.
	Goal: To attain high scalability for the project.
	NF5: Scalability - Horizontal Scaling allows us to create new pods for load balancing. This means the project is able to react to increased load and support more concurrent requests on demand.
	Kubernetes aligns with these goals and NFRs by providing a reliable and developer friendly platform to orchestrate containers and to scale up the project. Kubernetes self-healing feature also allows the project to attain high reliability even during stressful traffic load on the microservices. Kubernetes easily supports scaling and modification to the containers without downtime, which allows us to provide timely enhancements to our users without interruption to their usage of PeerPrep.
Continuous Deployment	
Github Actions vs AWS Code Pipeline	To support Continuous Deployment, the team chose Github Actions as the project's Continuous Deployment platform.
	Goal: To automate the process of building the microservice images and deploying the images onto a cloud-based Docker Image Repository.
	The project aims to support Continuous Deployment to increase reliability and increase the speed of deployment and development.

	The team chose Github Actions over AWS Code Pipeline. The benefits of Github Actions are its ease of use and simple logging setup. This enables the project team to easily detect any issues on the Continuous Deployment process on the Github web platform itself. The benefits of AWS CodePipeline is its seamless integration with other AWS Elastic Container Registry and AWS Elastic Kubernetes Services. This improves code security and Continuous Deployment performance, since the Continuous Deployment process is isolated within AWS' environment. The team chose Github Actions over AWS CodePipeline in favour of its ease of use and quick feedback of the Continuous Deployment process. Since anonymity of the image building process is not a priority of the project, using Github Actions is a feasible alternative to AWS CodePipeline.
Cloud Deployment	
Amazon Elastic Container Registry vs Docker Hub	To store the project's images, the team chose Amazon Elastic Container Registry.
	Goal: To automate the Deployment process by hosting the project's compiled microservices images on a secure and reliable Docker Image hosting platform.
	The project aims to store the project's images onto a cloud based hosting problem to automate the Deployment process. This is to enable our Kubernetes server to quickly access the project's images without the manual intervention of the developers. The benefits of Amazon Elastic Container Registry is its seamless integration with Amazon Elastic Kubernetes Service. Since our project uses Amazon Elastic Kubernetes Service as our deployment platform, there is high incentive to also use Amazon Elastic Container Registry due to its seamless integration and its high security given the Deployment process is within the AWS environment. Moreover, this seamless integration allows us fine-tuned control over the tagging of the images hosted on Amazon Elastic Container Registry. The benefits of Docker Hub is its ease of use. Since the team is using Docker as the project's container engine, publishing our compiled image onto Docker Hub is easy and seamless. Moreover, the images on Docker Hub can easily be made accessible to my team since Docker is used by the team. The team chose Amazon Elastic Container Registry over Docker Hub. This is because the team prioritizes the performance and

	reliability of the project's deployment on Amazon Elastic Kubernetes Services. If there are issues with the Kubernetes Deployment, the time taken to redeploy a new Kubernetes Pod is faster with Amazon Elastic Container Register than Docker Hub. Moreover, using Amazon Elastic Container Registry ensures higher security over Docker Hub, since the Deployment process is isolated within the AWS environment.
Amazon Elastic Kubernetes Service vs Amazon Beanstalk	To deploy our project, the team chose Amazon Elastic Kubernetes Service.
	Goal: Our project should be launched on a platform that supports high performance and increased scalability.
	NF4: Performance - Amazon Elastic Kubernetes Services integrates Amazon services with Kubernetes. This seamless integration between the hosted image repository, compute instances and Kubernetes reaps high performance.
	NF5: Scalability - Amazon Elastic Kubernetes Services is built upon Kubernetes. Kubernetes supports both Vertical and Horizontal Pod Scaling, which supports the deployment of multiple replicated containers and load balancing, which allows us to attain high performance under stressful traffic load.
	The project aims to be scalable, reliable and attain high performance. This is dependent on the platform that the project is executed on. The benefits of Amazon Elastic Kubernetes Service are inherent high scalability with fine-tuned control. Kubernetes Service supports Pod Scaling, Service Discovery, API Gateways which consolidates the essential components for high-performance orchestration between multiple containers. Kubernetes also supports Infrastructure as Code (IaC) which allows reliable and quick deployment. The benefit of Amazon Beanstalk is the ease of use. Amazon abstracts out the complexities of scaling, networking and security, and the team would only have to be concerned about hosted image repository and deployment of the code. The team chose Amazon Elastic Kubernetes Service because it grants the team fine-tuned control. The team is able to plan for the optimal architecture of our project, optimizing for performance and minimizing resources used. Amazon Elastic Kubernetes Service also allows us to control the scaling logic, which allows us to deliver high performance to our users.

Serverless	
Serverless Framework with AWS Lambda	The team aims to build a Question Scraper that is constantly available, automatically scales and has low overhead.
	NF3.3 Efficient Interactions - By hosting a serverless function, PeerPrep administrators will enjoy ease of use to scrape questions into PeerPrep. By executing a POST HTTP request, this will invoke the serverless function behind the scenes without any user intervention.
	The project aims to provide seamless usability for the users of PeerPrep. Beyond manually adding questions into PeerPrep or writing a locally hosted script to automate the addition of questions into PeerPrep, the project builds in a serverless function to automate the process of question scraping for the user. The team chose to deploy our serverless function using Serverless Framework. Serverless Framework supports Continuous Deployment of our serverless function to any cloud-hosting provider, in which the team chose AWS Lambda. The Serverless Framework not only has a dedicated dashboard that provides information about each function invocation and its corresponding logs, the Serverless Framework also provides an endpoint to ease our invocation of the serverless function. By using Serverless Framework, the team is able to automate the deployment of the serverless function, facilitates the debugging and optimisation of our serverless function, and seamlessly integrate configuration changes with AWS Lambda.

Table 28: Application Deployment Tech Stack Justification Table

4.6.7.3 Architecture Diagram

The following are architecture diagrams for the different deployments.

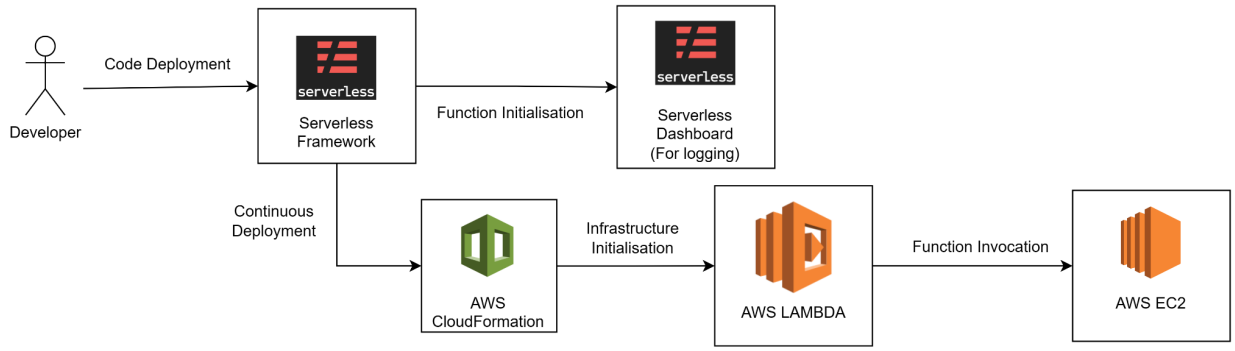

Figure 35: Serverless Function Deployment Architecture

To enhance the efficiency of our process in scraping new questions from Leetcode into PeerPrep, we opted for a serverless architecture using the Serverless Framework with AWS Lambda. This choice simplifies the challenging task of inserting new questions into PeerPrep, offering users a seamless and reliable platform while leveraging the benefits of cloud hosting. The Serverless Framework provides a cohesive solution, combining the functionalities of AWS Lambda with a custom dashboard that displays detailed logs, including the status of each function invocation and its duration.

AWS Lambda, a well-established serverless function platform, grants us fine-tuned control over the invocation of our Question Scraper. Specifically, for this application, we adjusted the HTTP timeout from the default 30 seconds to 1 minute to ensure smooth execution from start to finish, allowing the function to handle longer processes without interruption. This adjustment aligns with the need to maintain a robust and error-free execution of the scraping operation.

Upon deployment using the Serverless Framework, the dashboard on Serverless.com is updated, providing valuable insights into the health of the serverless function. Simultaneously, Serverless Framework employs AWS CloudFormation to deploy a new serverless function on AWS Lambda along with an API endpoint, adhering to best practices in Continuous Deployment. This approach streamlines the deployment process and ensures that updates are efficiently propagated.

After deployment completion, invoking the serverless function hosted on AWS Lambda is achieved through the assigned API endpoint. Behind the scenes, AWS Lambda automatically provides EC2 instances to run the code. It's noteworthy that, due to the cold start nature of new instances, the initial execution time may be slightly longer. However, over multiple consecutive invocations, this amortizes, providing a responsive and scalable solution for our Question Scraper. Overall, this serverless architecture optimally balances ease of deployment, reliability, and efficient execution for our specific use case.

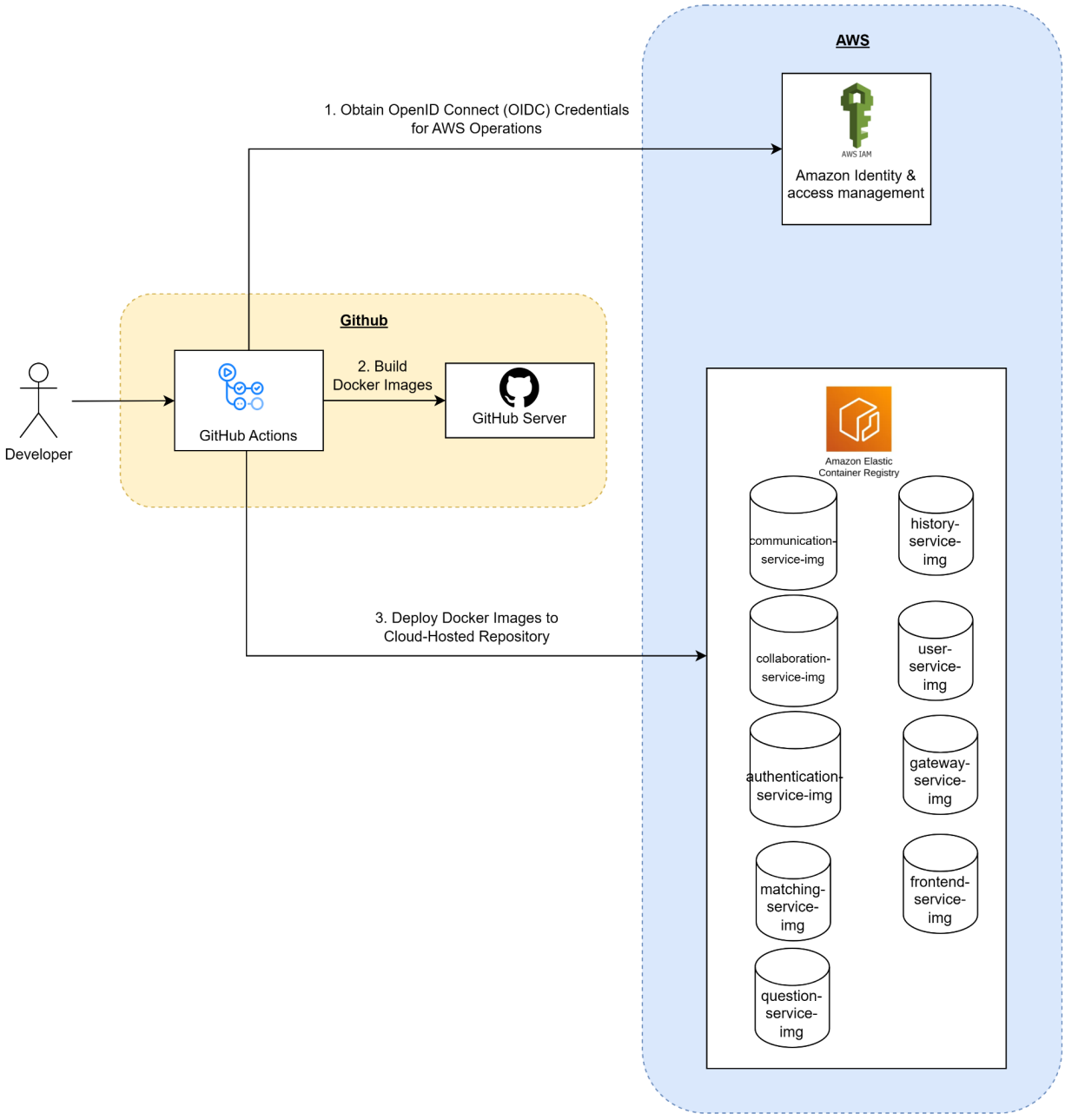

Figure 36: Continuous Deployment Architecture

To streamline our development process and enhance the reliability of our project, Continuous Deployment plays a pivotal role in automating the building and hosting of microservices' Docker images. This approach eliminates the manual labor associated with constructing Docker images for each microservice and managing their hosting. Leveraging Github Actions as our Continuous Deployment platform proves to be an excellent choice. Following each Pull Request, Github Actions seamlessly accesses our team's Github code repository and executes the Continuous Deployment process. Its user-friendly interface and rapid log generation provide developers with quick insights into the deployment process, allowing early identification of any issues.

Upon merging a new Pull Request, Github Actions initiates a secure login to Amazon Web Services via AWS Identity and Access Management (IAM) using the OpenID Connect protocol (OIDC). This authentication method significantly enhances security measures. Subsequently, Github Actions concurrently builds Docker images for each microservice using the robust infrastructure of Github backend servers. The final step involves uploading the built images to Amazon Elastic Container Repository (ECR). This integration with Amazon ECR ensures a seamless connection with Amazon Elastic Kubernetes Service (EKS), our chosen cloud-hosted Kubernetes service for this project.

Continuous Deployment guarantees that each merged Pull Request automatically reflects on the Amazon Elastic Container Repository Docker Image Repository. This automated synchronization ensures that the latest versions of our microservices are readily available for integration with other components, promoting consistency and reliability across the entire application. By employing Continuous Deployment with Github Actions and AWS services, we achieve a robust and efficient deployment pipeline that significantly accelerates our development speed while maintaining the highest standards of security and reliability.

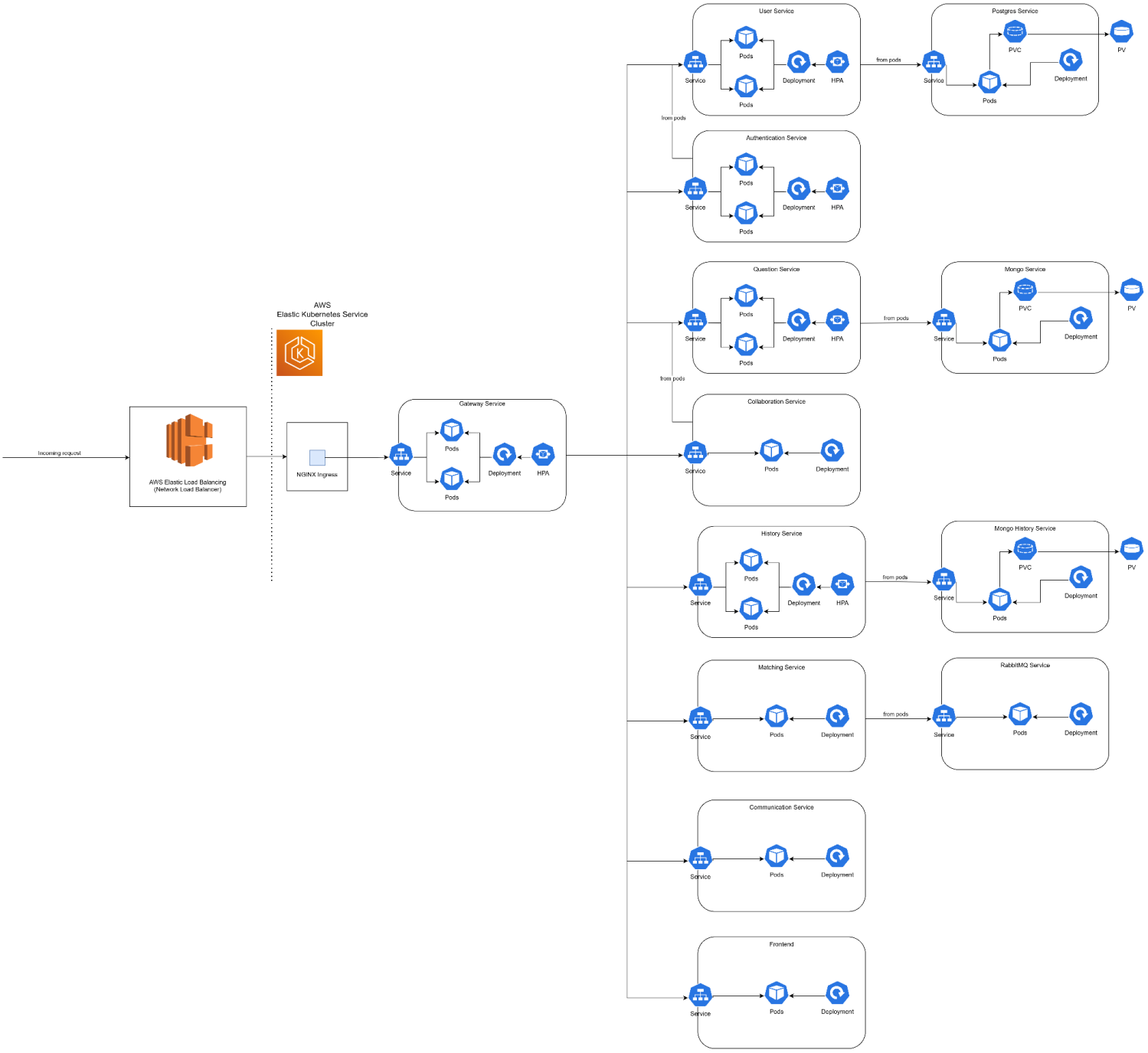

Figure 37: Kubernetes Deployment Architecture (simplified to show 1 pod for each microservice)

The overall architecture is the same as our diagram in 4.2 Overall Architecture, this diagram shows how Kubernetes components are utilized within each service. Each service uses a Kubernetes Deployment to manage the pods. Kubernetes Service is also established for each service to serve as an endpoint to the transient pods. For stateless services, Kubernetes HorizontalPodAutoscaler is set up which monitors load and increases replicas accordingly. For databases, we have Persistent Volume / Persistent Volume Claim to ensure that our data persists in case of any errors that may occur on the pods.

This design enables high performance and high scalability. Under high load, AWS Elastic Kubernetes Service is able to spin up new pods and/or pods of larger capacity to accommodate a stressful traffic load.

It is worth noting that the team have opted for AWS EC2 Managed Nodes instead of AWS Fargate Nodes. The difference is that this grants us fine-tuned control over the security, networking and access control of the cloud instances, instead of having AWS manage the nodes automatically for the team.

4.6.7.4 Design Justification

The design decisions for the collaborative space service prioritize delivering a smooth and user-friendly experience, and the chosen architecture aligns with this objective. The decision to use an event-driven architecture is rooted in the need for asynchronous communication, enabling seamless interaction between users engaged in collaborative activities such as video calling, code editing, and timer setting changes.

Below is a justification table for the chosen design architecture.

Justification	Explanation
Usability	1. **User-Friendly Experience:** Serverless architecture abstracts away infrastructure management, allowing developers to focus solely on code. This simplifies deployment and reduces complexity, leading to a more user-friendly experience for both developers and end-users. Additionally, AWS Lambda's Automatic Scaling ensures consistent performance, enhancing usability. 2. **Rapid Iterations:** Continuous deployment streamlines the release process, enabling quick iterations and updates. GitHub Actions automates the deployment pipeline, reducing manual interventions. IAM ensures secure access control, while AWS ECR provides a reliable and scalable

	container registry. This promotes a smooth and user-friendly update process. 3. **Container Orchestration**: Kubernetes on AWS EKS offers a robust container orchestration solution. The use of NGINX ingress enhances routing and load balancing capabilities. PeerPrep integration ensures efficient communication. This design choice optimizes resource utilization, leading to a user-friendly and responsive application.
Performance	1. **High Responsiveness**: AWS Lambda's automatic scaling ensures optimal resource allocation based on demand. This results in efficient use of resources, reduced latency, and improved overall performance. The serverless model allows for quick execution of functions, contributing to a high-performance application. 2. **Containerization**: Docker and AWS Elastic Container Repository facilitate containerization, ensuring consistent runtime environments. This ensures consistent performance across devices and enhances performance by providing a lightweight and efficient deployment process. Continuous deployment further accelerates the delivery pipeline, reducing time-to-market and enhancing performance. 3. **Scalable Orchestration**: Kubernetes on AWS EKS enables horizontal scaling, allowing the application to handle increased loads seamlessly. The use of AWS Network Load Balancer and NGINX ingress enhances routing efficiency. PeerPrep integration ensures scalable communication between microservices, contributing to high-performance levels.
Scalability	1. **Granular Scaling:** AWS Lambda's fine-grained scaling automatically adjusts resources based on individual functions. This granular scalability optimizes resource utilization, allowing the application to handle varying workloads efficiently. Serverless architecture inherently supports scalability without manual intervention, ensuring flexibility for future growth. 2. **Use of a cloud-based Docker Image Repository:** AWS ECR's scalability accommodates the growing need for containerized applications. Continuous deployment practices enable the application to scale rapidly, ensuring

	that it can adapt to changing demands and maintain high availability during peak usage periods. 3. **Horizontal Scaling:** Kubernetes on AWS EKS allows for easy horizontal scaling of microservices. The use of AWS Network Load Balancer and NGINX ingress enhances the distribution of incoming traffic. This empowers our project to support seamless communication across horizontally scaled components, ensuring the project scales efficiently.

Table 29: Deployment Architecture Design Justification Table

This design strategy leveraging Docker, Kubernetes, and serverless technologies in AWS Lambda, combined with continuous deployment practices, results in a cohesive approach that addresses usability, performance, and scalability aspects. The serverless model simplifies development and promotes a user-friendly experience, while containerization and Kubernetes orchestration enhance performance and scalability, making the application robust and adaptable to varying workloads. Continuous deployment practices further contribute to the overall efficiency and reliability of the development pipeline, ensuring a seamless and responsive user experience.

As for Kubernetes Deployment, the team wishes to highlight some notable design justifications.

- Kubernetes Service (Service Discovery N12)
    - As pods are transient, having a service object allows us to have a permanent address to the deployment which would not be affected even when our pods are replaced. It also allows us to have multiple pods in a deployment while keeping a single endpoint. This simplifies communication as the developer does not have to discover the address of individual pods. Kubernetes Service does automatic load balancing to its set of pods. Therefore, load balancing can be achieved by merely increasing the number of pods.
- Kubernetes Horizontal Pod Autoscaler (Scalability N10)
    - Rather than pre-defining the scale of our deployment, having HPA helps us to be resource efficient while ensuring performance of our application.
- Kubernetes Persistent Volume
    - While Kubernetes provides self-healing capability, there is a need to ensure that our data persists for the application to be reliable.

4.6.7.5 Local Deployment Guide

- Dependencies: Minikube, Docker, Kubectl
    - Minikube (https://minikube.sigs.k8s.io/docs/start/)
        - Use Docker Driver (https://minikube.sigs.k8s.io/docs/drivers/docker/)
    - Kubectl (https://kubernetes.io/docs/tasks/tools/)
- Steps to Setup Project:
    1. Start Minikube in cmd
        a. Run `minikube start`
    2. Ensure Metrics-Server is enabled in Minikube
        a. Run ` minikube addons enable metrics-server`
    3. Change directory to: `/development/kube_developement`
        a. Ensure `config.yaml` and `secret.yaml` is in this directory
    4. Depending on your operating system run the following command: (https://minikube.sigs.k8s.io/docs/handbook/pushing/)
        a. Linux / MacOS:
            `eval $(minikube docker-env)`
        b. Windows:
            i. Powershell: `& minikube -p minikube docker-env --shell powershell | Invoke-Expression`
            ii. CMD: `@FOR /f "tokens=*" %i IN ('minikube -p minikube docker-env --shell cmd') DO @%i`

        *switch Docker client's context to interact with Docker daemon within Minikube

    5. Run `docker compose build` - build the images (available to minikube due to previous command)
    6. Run `kubectl create namespace eks-peerprep` to create namespace
    7. `kubectl apply -f .` - Apply manifest in current directory to Kubernetes cluster
    8. Verify that all deployments are running
        a. `kubectl get all -n eks-peerprep`
    9. Run `minikube service gateway-service -n eks-peerprep`

5. Future Enhancements and Features

The following table outlines forthcoming enhancements and features targeted for future implementation or unimplemented features due to time constraints. These endeavors aim to further increase the project's prioritized quality attributes and NFR such as usability, user experience, performance and scalability.

Future Enhancement	Description
Question Filter: Topic Selection	To enhance the question filtering system, the ability to filter questions by topics will be introduced. Currently, the only available filter is based on question difficulty, limiting the user's ability to focus on specific subject areas. The proposed improvement aims to empower users by allowing them to refine their practice sessions to particular topics of interest. This modification aligns with the user-centric goal of providing a more targeted and personalized learning experience.
Match by Question Topic	Similar to topic selection question filtering, users would likely prefer to simulate technical interviews based on specific topics rather than just a randomized selection. Including an option for users to match for collaboration based on question topic enhances the user experience by providing a more targeted and relevant practice environment.
Match Confirmation Prompt (F2.3)	Due to time constraints, the implementation of this feature was not completed within the designated timeframe and is now earmarked as a future enhancement.
Increase Custom Room Limit	To accommodate practical technical interview settings where there might be more than one interviewer, the custom room is planned to increase its room limit from two users to support multiple participants. This expansion enables the room to function not only for paired collaboration but also as a discussion room, leveraging the capabilities of Peer, as mentioned in section 4.6.5.
IntelliSense for more languages (F4.2.2+)	Currently the project only supports auto-complete keyword suggestions for JavaScript among our 4 supported languages. It would greatly enhance the user's experience if similar capabilities are also implemented for the remaining 3 languages, C++, Java and Python.

Distinguish user by cursor colour in code editor (F4.1.3)	Due to time constraints, the implementation of this feature was not completed within the designated timeframe and is now earmarked as a future enhancement.

Table 30: Future Enhancement Table

6. Reflection and Learning Points

The team started out with the project with an ambitious stance. As the team was too focused on acquiring skills from this project, many of the nice-to-have has been listed as a checklist. However, due to the diverse background and experience of the team, there have been several miscommunications which delayed the progress of the team.

Moreover, the team also was too focused with the task at hand which resulted in the lack of planning, resulting in the bottleneck in the progression. With the differences in priority, the team eventually has to regroup and align the expectations. The meeting with the team eventually helped the team to view the project realistically and to lower their expectations.

In retrospect, there are several learning points in the group. Firstly, scheduling can be included as a factor in a subgroup. In the subgroups, two out of three of the members are heavily involved in their capstone project. Hence, the remaining members are required to accommodate to the schedule while ensuring tasking continues. By separately members with heavy workload, the progression will not be bottleneck during evaluation season for members with high workload.

The team can also employ a member to set deadlines. By having a team member acting as a timekeeper, the team can be ensured to be on track.

Additionally, having members understand different subgroups implementation on surface level is also helpful when integrating. For example, the implementation on frontend was able to rerender on save. However, during the integration process, frontend was containerized with the other services. This resulted in frontend implementation having to rebuild the container for the changes to reflect. This severely resulted in a delay in the efficiency of frontend implementation. By allowing each member to have understanding of each other’s implementation, less confusion will be caused and efficiency will also be enhanced.

7. Appendices

- PrepPeer Git Repository: https://github.com/CS3219-AY2324S1/ay2324s1-course-assessment-g09

8. References

Report Format:

- https://www.batimes.com/articles/exploring-quality-attribute-requirements/
- https://treeckle.com/treeckle_2.0_td.pdf

Tech Stack Icon Reference:

- https://icon-icons.com/
- https://worldvectorlogo.com/
- https://vecta.io/symbols/9/aws-compute/32/aws-elastic-beanstalk
- https://vecta.io/symbols/9/aws-compute/12/amazon-ecr