CS3219 Software Engineering Principles and Patterns

PeerPrep - Final Report

G06

Project Team:

Student Name(s):	Student ID(s):	E-mail(s):
Ang Jun Kang	A0235183H	angjunkang@u.nus.edu
Chong Yijing Isabel	A0241158J	isabel.chong@u.nus.edu
Kalaris Ng Yun Shan	A0239510L	kalarisng@u.nus.edu
Loo Wei Zhe Evan	A0231054W	E0701818@u.nus.edu
Wong Qin Yao	A0235155J	wong.qy@u.nus.edu

Table of Contents

Summary

In this comprehensive report, our team delves into the intricacies of PeerPrep, a one-stop solution for a dynamic peer matching system with a technical interview preparation platform. This initiative is part of the National University of Singapore's (NUS) CS3219 course on Software Engineering Principles and Patterns, and it stands as a practical application that provides meaningful explanations for the types of software principles and patterns learned and applied throughout the course.

The document encapsulates the journey of conceptualising, designing, and implementing PeerPrep, providing a detailed account of our team's approach, challenges encountered, and the solutions engineered. From the context and purpose of PeerPrep to the intricacies of the chosen microservices architecture, this report serves as a testament to our dedication and innovation in addressing the need for a collaborative space. This is where individuals, regardless of their background, can come together to enhance their technical interview skills and enjoy the application's benefits. The report offers an in-depth exploration of the project's management, and the thoughtful considerations behind feature choices, making it accessible and beneficial to a wide range of users.

Acknowledgement

We would like to express our heartfelt appreciation to our Teaching Assistant - Gaurav Gandhi, who has been an invaluable source of guidance and support throughout the development of PeerPrep. Gaurav played a pivotal role in steering our project in the right direction, offering useful insights, feedback, and unwavering encouragement. His expertise and responsiveness have been instrumental in helping us overcome challenges and ensuring the successful progress of our project.

In addition, we extend our appreciation to our esteemed lecturers, Dr. Akshay Narayan and Assoc. Prof Bimlesh Wadhwa. Their dedication to teaching us industry-relevant software engineering principles and patterns has been instrumental in preparing us for the role of a software engineer. Their insights, knowledge, and commitment to our education have equipped us with the necessary skills and knowledge to tackle real-world software engineering challenges.

We are grateful for the valuable contributions and support from Gaurav and our dedicated lecturers, which have greatly enriched our learning experience and project development.

Chapter 1. Introduction

1.1. Problem Statement and Context

PeerPrep is part of the National University of Singapore's CS3219 course on Software Engineering Principles and Patterns, and is a practical application intended to solidify the theoretical knowledge gained during the course. In this application, we address the need for a technical interview preparation platform, the challenges individuals face in preparing for such interviews and the necessity for a collaborative solution. This application aims to benefit students, professionals or anyone seeking to enhance their technical interview skills.

In the software engineering industry, technical interview skills are paramount, serving as a key evaluation metric for candidates' problem-solving abilities, core knowledge, coding proficiency, adaptability and communication skills. With the industry's competitive landscape and diverse formats of technical interviews, there is a growing demand for better preparation methods. This is where PeerPrep comes in. With our innovative approach of using peer matching systems and microservices architecture coupled with diverse resources, we effectively foster collaboration and skill enhancement all in one setting.

1.2. Purpose

PeerPrep acts as a centralised hub, offering an array of resources specifically curated to meet the diverse and evolving requirements of individuals preparing for technical interviews. The platform's repository encompasses an extensive range of choices such as difficulty of questions and topics to prepare for to support candidates across various interests inherent in technical interviews.

At its core, PeerPrep translates classroom collaboration into a real-world application. By introducing the element of collaboration like what we often do in schools, PeerPrep aims to foster an environment where individuals are encouraged to apply the acquired theoretical principles to hands-on scenarios, together with a peer who has the same interests as them. In addition, PeerPrep also provides a space for matched peers to ask each other technical questions. These functionalities aim to help users to develop a deeper understanding of the functionality and significance of the topic in real-world technical interview settings.

Chapter 2. Choice of Architecture

2.1. Microservices Architecture

Our team has chosen to adhere to the microservices architecture to be able deploy independent applications without the services interfering with each other. This ensures high cohesion and loose coupling in our application which provides us with easier to maintain and better designed groups of code to develop features in parallel more effectively. PeerPrep is orchestrated through a set of distinct services, each playing a pivotal role in delivering a comprehensive and effective technical interview preparation platform. Each service is organised around the main business domains - user, questions, matching.

The User Service takes charge of user profile management context and user history context, ensuring a seamless and personalised experience for each participant.

Our Matching Service is strategically designed to match users based on diverse criteria, including the difficulty level of questions and specific topics. This service offers the flexibility to incorporate multiple matching criteria, tailoring the user experience to individual preferences.

The Question Service is dedicated to curating and managing a robust question repository, meticulously indexed by difficulty level and other relevant criteria such as specific topics. This repository serves as the foundation for the technical interview preparation process.

To provide a user-friendly entry point, the user interface (UI) for user interaction serves as the interface for users to access and navigate the application seamlessly.

Lastly, the deployment of the application is addressed with the option to either deploy it on a local machine using Docker Compose or on Google Cloud Run leveraging on a Docker-based solution.

These services collectively form the backbone of PeerPrep, ensuring a well-rounded, feature-rich, and user-centric platform for technical interview preparation.

2.2. Backend Architecture

2.2.1. Microservices and RESTful API

The backend architecture of PeerPrep is structured around microservices, each representing a self-contained unit responsible for distinct business functionalities. These microservices communicate with each other through a RESTful API, adhering to the principles of Representational State Transfer (REST). This architectural choice facilitates the seamless interaction between different components, allowing for scalability, flexibility, and maintainability.

2.2.2. Model-View-Controller (MVC) Design Pattern

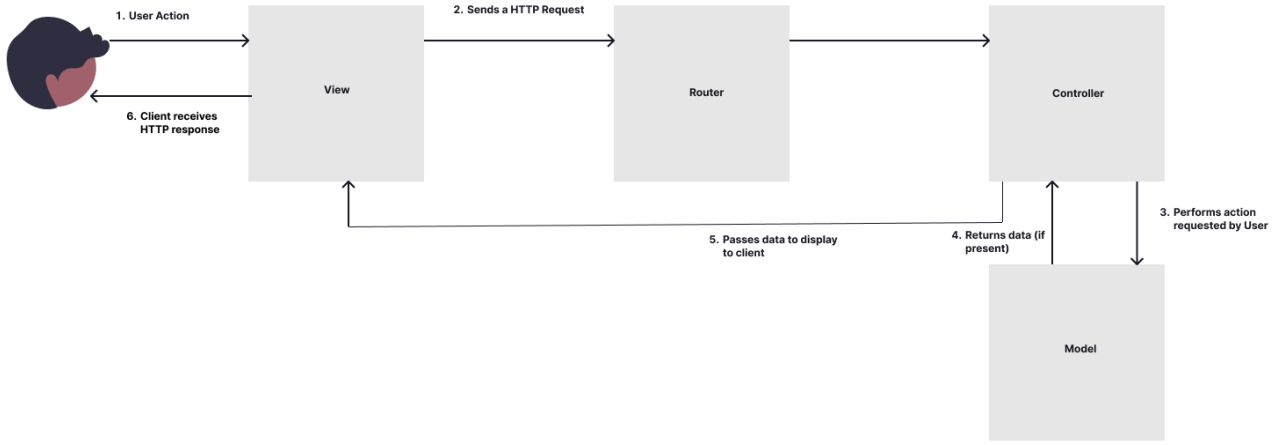

Figure 2.1. PeerPrep's Backend MVC Pattern

Within each microservice, the Model-View-Controller (MVC) design pattern is employed to organise and separate concerns, enhancing the overall maintainability and readability of the codebase. Each microservice consists of mainly the view, controller and model layers.

2.2.3. MVC Layer Description

View

While the view is in the frontend, it communicates with the router to execute backend related APIs. The client sends a HTTP request to the server through a specified endpoint and the router pings the controller to invoke the user action. When necessary, the controller updates the model per user action. The data, if requested, is

sent back from the model to the controller and back to the view to display to the client.

Router

The router acts as the intermediary between the outside world (client requests) and the internal MVC components. Routers define the URL endpoints and map them to the corresponding controller functions. Routers direct incoming requests to the appropriate controller action.

Controller

The controller serves as a bridge between the router and model. Its main role is to read data from request parameters or body and select the model for processing. Most of the preparation of the view is done here through utilising the model and the model's API.

Model

Models help encapsulate data and contain methods for manipulating states to be sent back to the controller and/or the database. They help define the structure of our database tables.

2.2.4. Benefits of MVC Pattern

Some benefits of adopting MVC over layered styles would be the modularity as Separation of Concerns principle is strictly followed. MVC facilitates extensibility where a new view and controller pair can be added for a new interface medium or even a new functionality can be easily added to the model independently of other components. For instance, when developing our User History, we are able to add an additional controller in user-services with endpoints to analyse user data after user-services main functionality has been developed.

2.2.5. Our Microservices

Backend services are structured into three microservices, each catering to a specific aspect of PeerPrep's functionality, they are:

1. **User Service**: This microservice handles the logic for user profile management, authentication, authorization and user history. It acts as the central hub for user data and user history. It works in tandem with our Cloud SQL for relational database to handle user data.
2. **Question Service**: This microservice is responsible for handling all question-related operations. It works in conjunction with MongoDB for flexible storage.
3. **Matching Service**: This microservice handles the logic behind user-matching, ensuring that users are paired accurately for collaborative sessions based on their set criterias.

Chapter 3. Tools and Services

This chapter encapsulates the tools and services used throughout the planning and execution of the PeerPrep application. It serves as a brief description of the tools. Further description of its usage with external libraries is described in Chapter 7.

3.1. Project Management

Jira and Scrum Management

Jira is a powerful project management and issue tracking tool that we used to plan, track, and manage our project efficiently. From initial ideation to the final delivery, Jira is essential in serving as the cornerstone of our project management and collaboration efforts. With Jira, we are able to enhance teamwork, prioritise tasks, and maintain a clear overview of our project's progress, leading to improved productivity and the successful achievement of our project goals.

In the screenshot below, it is evident why we chose Jira for issue and scrum management. The screenshot shows the planned deadlines and timeline for each assignment. And a kanban board is used to assign smaller issues inside each assignment where a member would be responsible for. Inside Jira, we can also link all relevant documentations such as the Product Backlog.

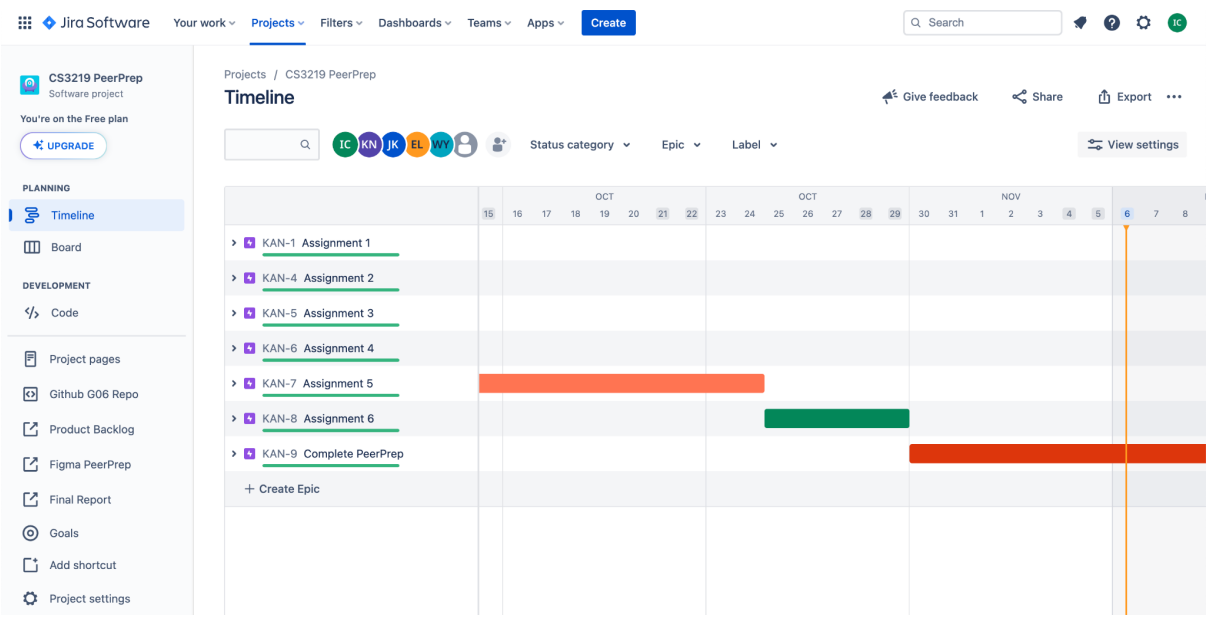

Figure 3.1. Jira for Issues Management

Aligned with our commitment to adopting Scrum as our chosen agile framework for managing the project's progress, we have embraced a structured and flexible approach to project management. Scrum places a strong emphasis on iterative and incremental development, fostering continuous collaboration and adaptability to changing requirements.

In this framework, Jira stands out as our comprehensive platform for project management, serving as a one-stop solution to seamlessly integrate and facilitate Scrum methodologies. By leveraging Jira's capabilities, we ensure that our project management processes align with Scrum principles, allowing for effective planning, continuous improvement, and transparent communication among team members.

3.2. User Interface Design

Figma

Figma is an essential tool we used to plan out our frontend user interfaces. Due to its ability to facilitate collaborative design and interactive prototyping, Figma allows us to create clickable prototypes, in turn helping us to continuously refine our user experiences. The tool also offers real-time previews, streamlining our design process and ensuring consistency across the pages. Furthermore, its design handoff capabilities and support for responsive design make it an invaluable asset for efficient UI planning, saving time in the design phase.

Once the design on Figma was created, it served as a valuable reference point for the entire team. It allowed everyone to visualise the user interface, layout, and overall aesthetics of the project. This alignment on a template design was crucial for ensuring a consistent and cohesive user experience across the frontend development. With the Figma design as a guide, we could make informed decisions about the visual elements, colour schemes, typography, and the placement of different UI components without miscommunication.

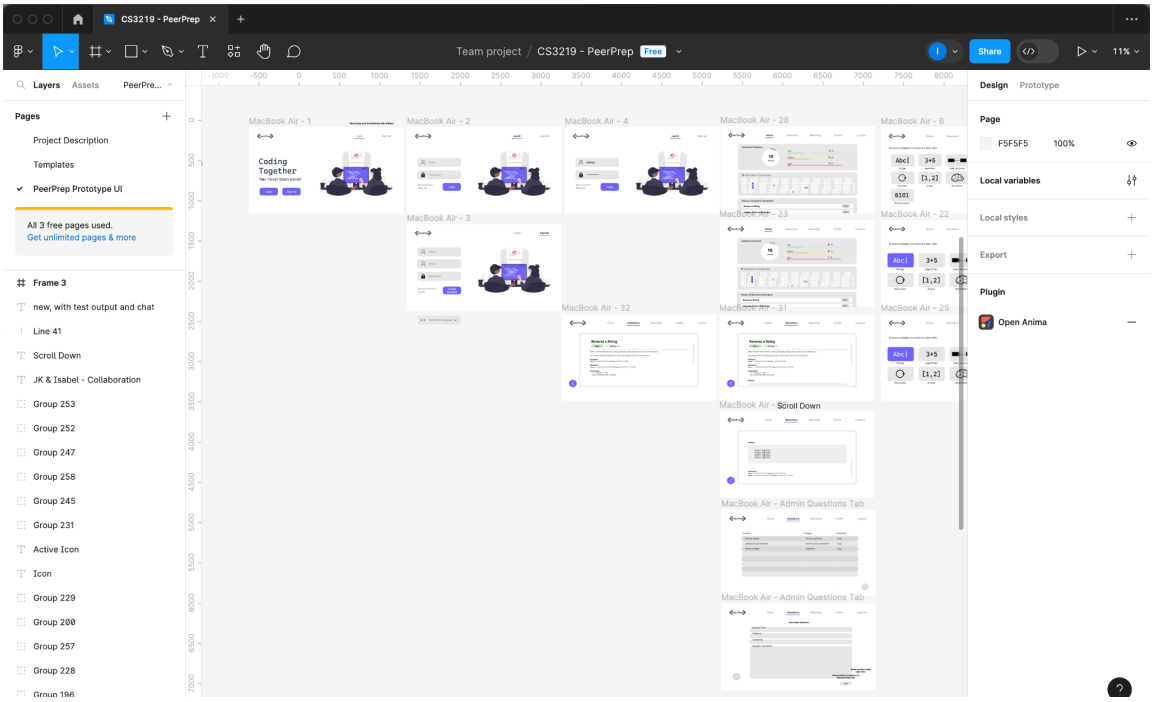

Figure 3.2. Figma Interface for PeerPrep Design

3.3. Full-Stack Development

React

React is used for building the frontend of our interactive web application. React's component-based architecture promotes code reusability, making it efficient and easy for us to create dynamic user interfaces. It also has a strong developer community and a rich ecosystem of libraries and tools which we had easily leveraged on for our project.

Node.js

Node.js is used for our backend server-side JavaScript development. Its non-blocking, event-driven I/O model makes it suitable for us to employ it to build scalable and real-time applications. This helps us to simplify sharing code between the client and server.

Express

Express is used as a web application framework with Node.js. It simplifies routing, middleware, and request handling, making it a lightweight framework that speeds up our web application development, especially with the APIs and web services we have built.

3.4. Cloud and Local Database Management

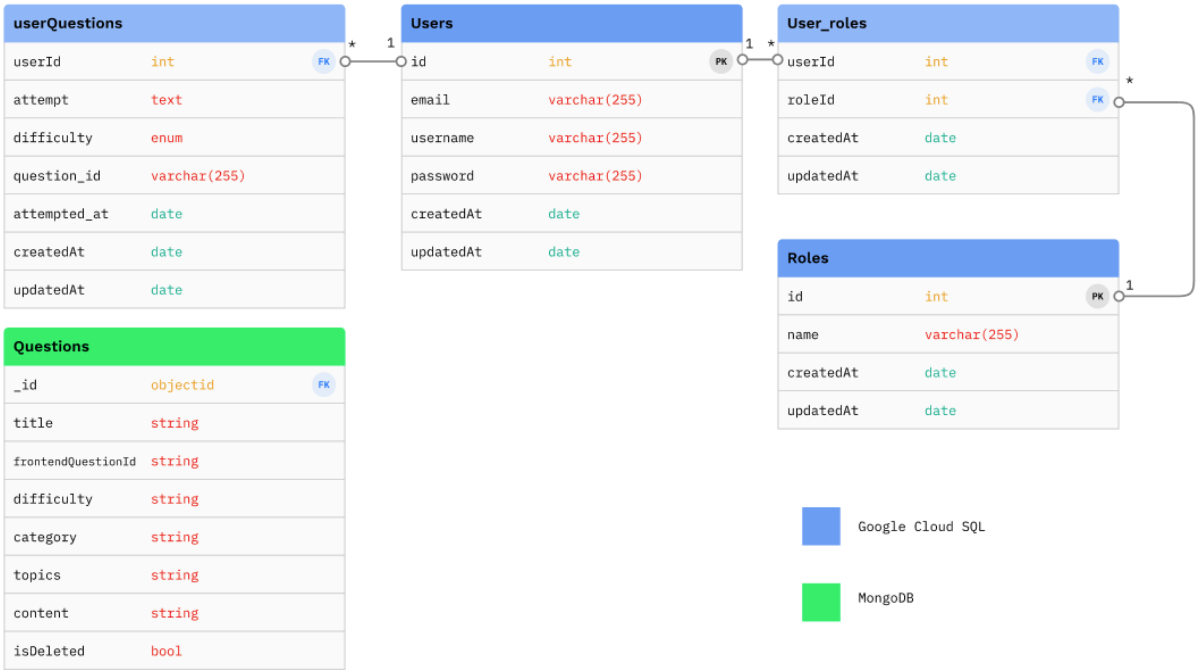

Figure 3.3. Entity Relationship Diagram of Databases

MongoDB

MongoDB serves as the cornerstone of our application, specifically in storing and managing the multitude of questions that fuel our platform's core functionality. The design choice to employ MongoDB is rooted in its exceptional suitability for our use case. Its flexible, schema-less data model allows us to seamlessly accommodate a diverse range of question types and formats, making it ideal for our dynamic and evolving question database.

MongoDB Atlas

Moreover, MongoDB Atlas ensures that our data is not just stored but is also managed efficiently without the hassle of infrastructure maintenance. The platform guarantees data availability, automatic backups, and scalable capabilities, assuring a robust and reliable foundation for our application. With MongoDB, we can provide our users with a seamless, responsive, and scalable experience, underpinning the success of our question-centric application.

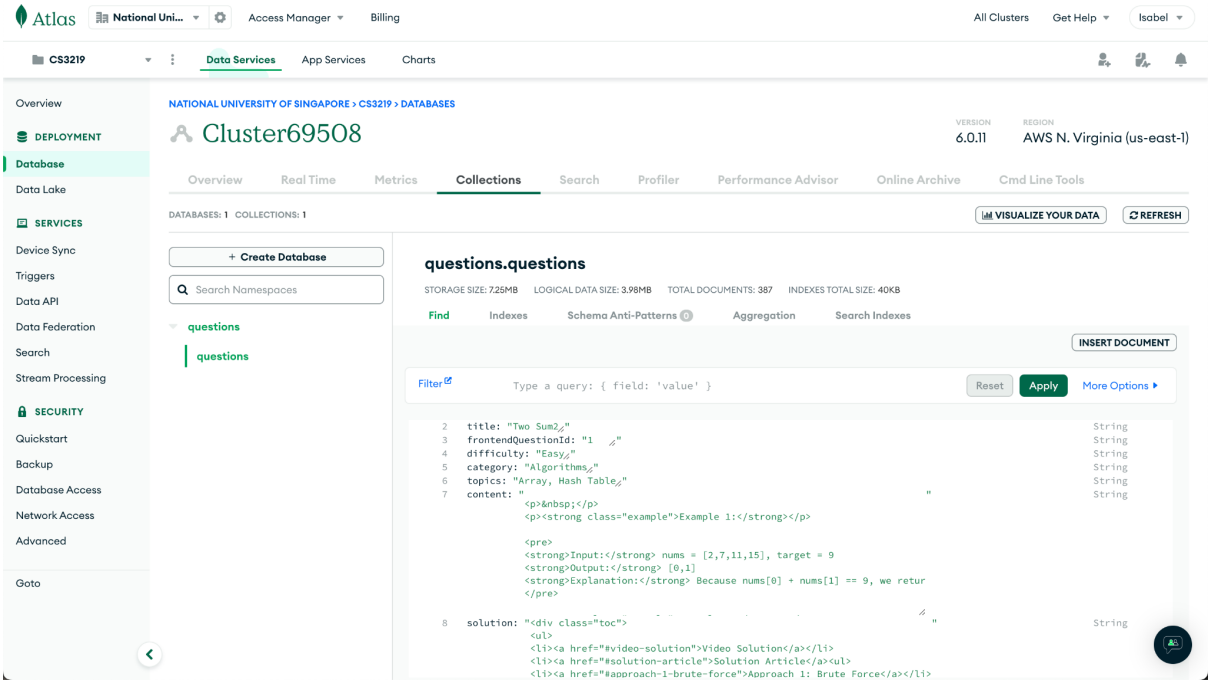

Figure 3.4. MongoDB Atlas Interface Showing Technical Questions

MySQL Workbench

MySQL Workbench is used for MySQL database design and administration. In the development of our application, we required a reliable and powerful SQL environment for database design, management, and testing. MySQL Workbench is compatible with various versions of MySQL and supports different operating systems. This compatibility and portability allowed our team to work in diverse development environments without compatibility issues. The visual design interface also allowed simplified database schema creation, management and querying, which has helped us with debugging and refining our application.

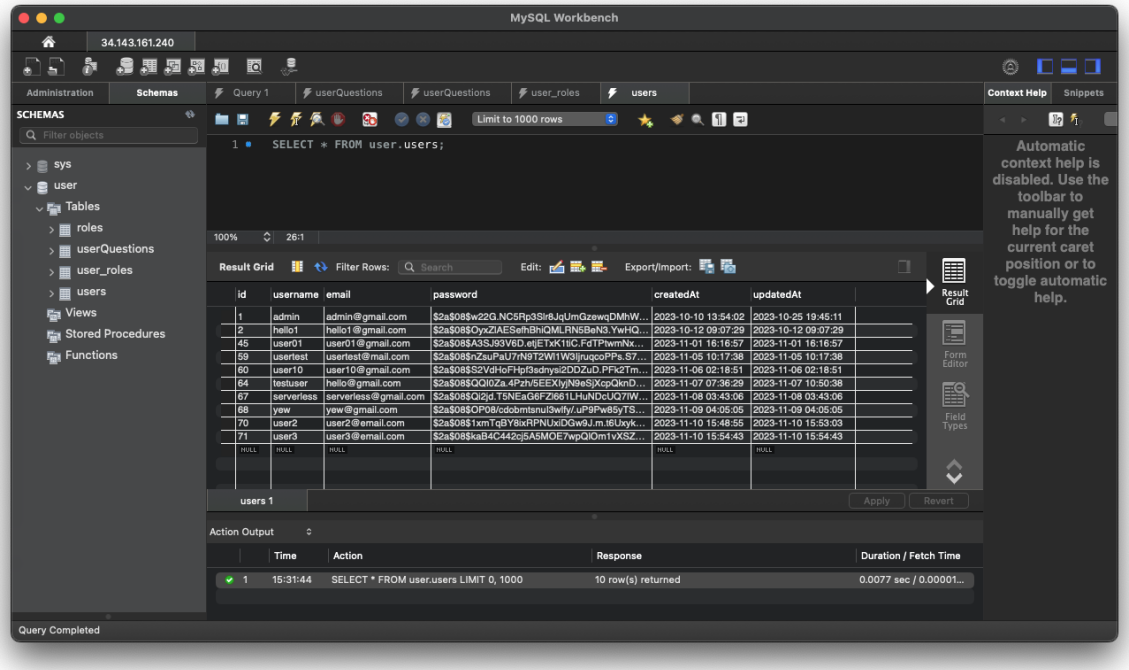

Figure 3.5. MySQL Workbench Interface Showing User Management

3.5. Matching and Code Collaboration Tools

Socket.IO

Socket.IO plays a pivotal and indispensable role in our web application, as the primary objective of our project is to create an environment where users can seamlessly collaborate with one another. By harnessing the power of Socket.IO, we can implement vital collaborative features, such as real-time chat and shared code spaces. This dynamic, bidirectional communication technology ensures that users can interact with each other instantaneously, facilitating lively discussions, code reviews, and real-time collaboration.

Socket.IO not only elevates user engagement and interaction within our application but also serves as the backbone of our platform's core functionality, making it a vital component in our quest to foster collaboration and teamwork among our users.

CodeMirror

CodeMirror shines in collaborative coding by offering a versatile and feature-rich code editing environment that supports real-time collaboration. Its capabilities, such as syntax highlighting, autocompletion, and code linting, ensure that collaborative coding sessions are efficient and free from errors. The tool's extensibility allows for

customisations tailored to our team's specific needs, whether it's building a collaborative code editor or a code playground.

With CodeMirror, we can create interactive and responsive code editing interfaces, facilitating a smooth and productive coding experience for both local and remote team members. It serves as a powerful resource that significantly enhances web development projects and fosters effective teamwork among users, making it a go-to choice for collaborative coding endeavours.

OneCompiler

OneCompiler provides a seamless platform for writing, compiling, and running code online. This online tool offers a convenient and accessible way to write and test code in multiple programming languages directly in the browser, making it a valuable resource for learning, testing, and sharing code snippets. Leveraging a powerful cloud infrastructure, it enables swift code execution without the constraints of traditional setups, enhancing the coding experience.

RapidAPI

RapidAPI serves as a bridge between our application and the OneCompiler code execution service. Through RapidAPI, we can effortlessly integrate and access the functionalities provided by OneCompiler's API. This streamlined integration ensures that our users can execute and test code within the application, fostering a convenient and interactive coding environment.

These software tools and services cater to different aspects of software development, project management, and collaboration. They streamline workflows, enhance productivity, and contribute to the successful development and management of PeerPrep.

3.6. DevOps

Our DevOps strategy is designed to streamline development, testing, and deployment processes, ensuring consistency across different environments. Leveraging Docker for local development, we address the "it works on my machine" challenge. Docker Compose facilitates multi-container applications for efficient testing, and Google Cloud Platform (GCP) serves as the deployment platform.

3.6.1. Local Development Using Docker

Docker

One of Docker's primary advantages is the guarantee that if it works on one developer's machine, it will work on all others. This portability ensures consistency across different environments, from development to production, eliminating the "it works on my machine" problem. Containers also have less overhead than virtual machines, and Docker optimises this with its lightweight runtime and efficient use of system resources.

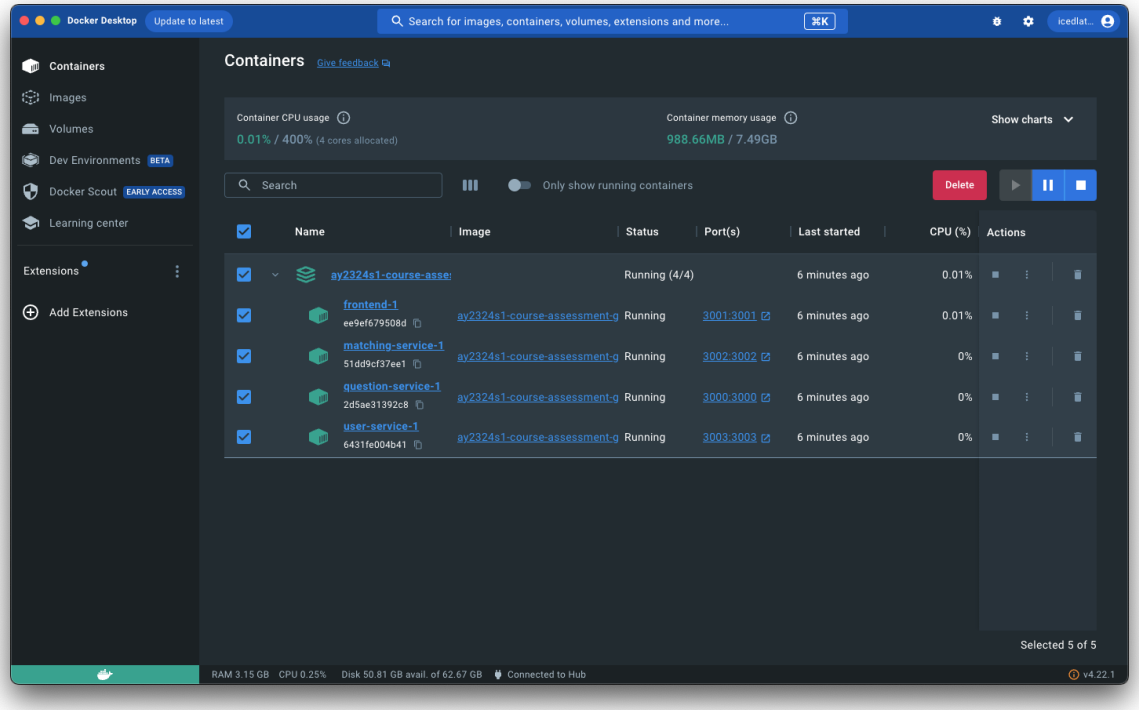

Figure 3.6. Docker Desktop Interface Showing PeerPrep Microservices Running

For local development and testing, Docker Compose allows us to define and run multi-container Docker applications and a virtual network so that services can communicate with each other through service name and designated port number. By using a YAML file to configure the application's services, we have been able to launch our entire stack with a single command. The use of Dockerfiles provides a clear, version-controlled specification of how our images are built. Each Dockerfile contains all the commands a user could call on the command line to assemble an image, ensuring our builds are transparent and repeatable. With Docker, we can replicate the Cloud Run environment locally, which aids in testing and debugging. This ensures that any issues can be caught early in the development cycle before deployment.

Snippet from docker-compose.yml

```yaml
version: "3"

services:
  user-service:
    build:
      context: ./user-service
    environment:
      - USR_SVC_PORT=${USR_SVC_PORT}
      - SQL_HOST=${SQL_HOST}
      - SQL_USERNAME=${SQL_USERNAME}
      - SQL_PASSWORD=${SQL_PASSWORD}
      - JWT_SECRET=${JWT_SECRET}
    ports:
      - 3003:3003
    networks:
      - service

  question-service:
    build:
      context: ./question-service
    environment:
      - USR_SVC_AUTH=http://user-service:3003/api/auth
      - QNS_SVC_PORT=${QNS_SVC_PORT}
      - MONGO_HOST=${MONGO_HOST}
...
```

Figure 3.7. Code Snippet Of docker-compose-yml

3.6.2. Deployment on Google Cloud Platform (GCP)

Google Cloud Platform (GCP)

To run containers on cloud run, we need to configure key information about the containers we want to use, such as Docker images used, CPU and memory allocation, with environment variables. Services automatically manage the scaling of instances based on the traffic the application receives.

Google Container Registry (GCR)

GCR is a fully managed container registry service provided by GCP. It allows pushing of production builds to GCR which can then be deployed on Google Cloud Run. By storing images in the container registry, we are able to rollback deployments to previous iterations.

Overview of Deployment

Our application consists of three microservices and one frontend website. For each of these, we have one Cloud Run instance for it and scaling is done automatically.

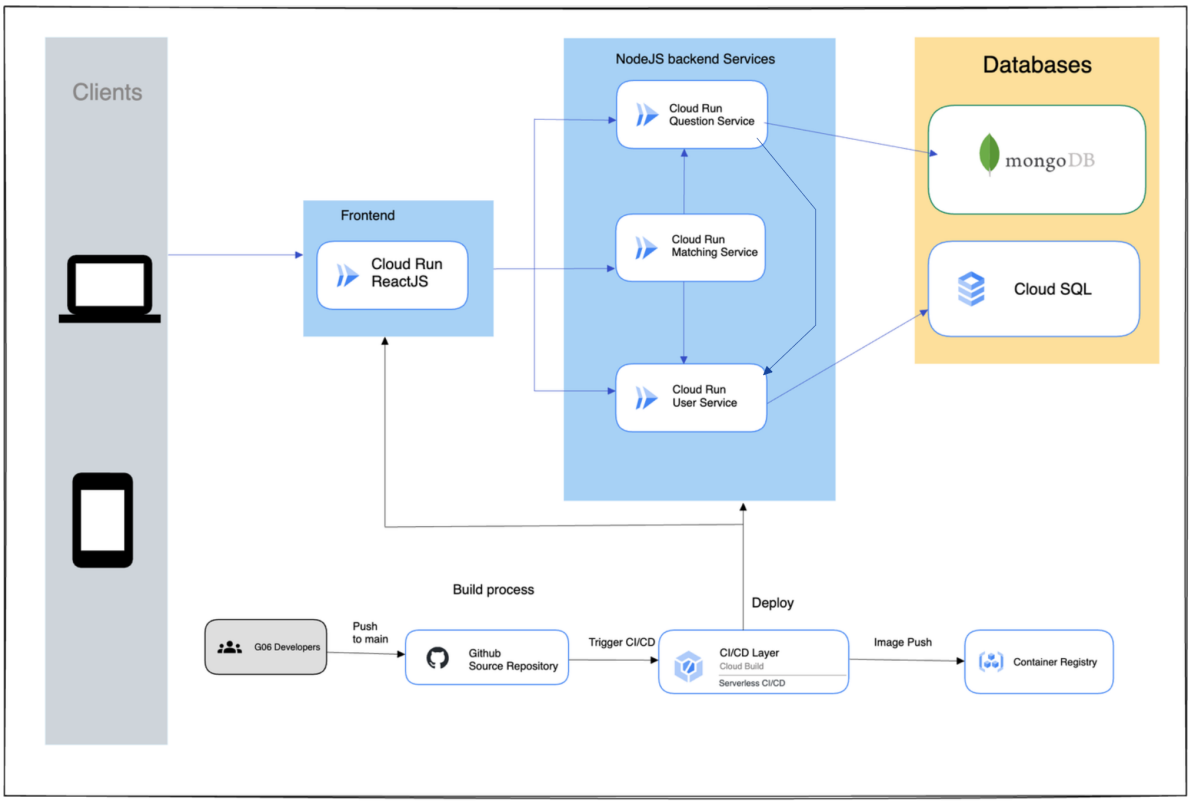

Figure 3.8. High Level Overview of Architecture Diagram

In our application architecture, as presented in the diagram, we have designed a robust and scalable infrastructure that ensures a seamless and responsive experience for our clients. Our clients, who can access our platform through a variety of devices such as desktop computers and smartphones, interact primarily with our ReactJS-based Frontend. Hosted on Google Cloud Run, our Frontend serves as the interactive face of our application, designed to be an engaging and user-friendly yet

decoupled system where we can rapidly iterate to push updates, test new features, roll out changes without impacting backend services.

Google Cloud Run

Google Cloud Run stands as a cornerstone for PeerPrep's deployment, offering a serverless execution environment for our containerized applications. Our decision to utilise Google Cloud Run was catalysed by its fully managed nature, removing the burden of server management from our team and allowing us to focus on development and innovation.

Once our application is containerized using Docker, we can deploy it effortlessly to Google Cloud Run, this eliminates the need for adaptation or modification of the container to suit the cloud environment. Docker also ensures that the application will run in the same environment during development and production in Cloud Run. This consistency helps prevent discrepancies and potential bugs that might arise from environment differences.

One of the critical deployment decisions we made was whether to deploy multiple services in a single Virtual Machine or Engine Compute instance versus adopting a containerized deployment across multiple cloud instances. We ultimately chose the latter, leveraging containerization for its benefits in terms of scalability, flexibility, and ease of management. This decision aligns with industry best practices and facilitates efficient resource utilisation in a cloud-native environment, and is further elaborated on below.

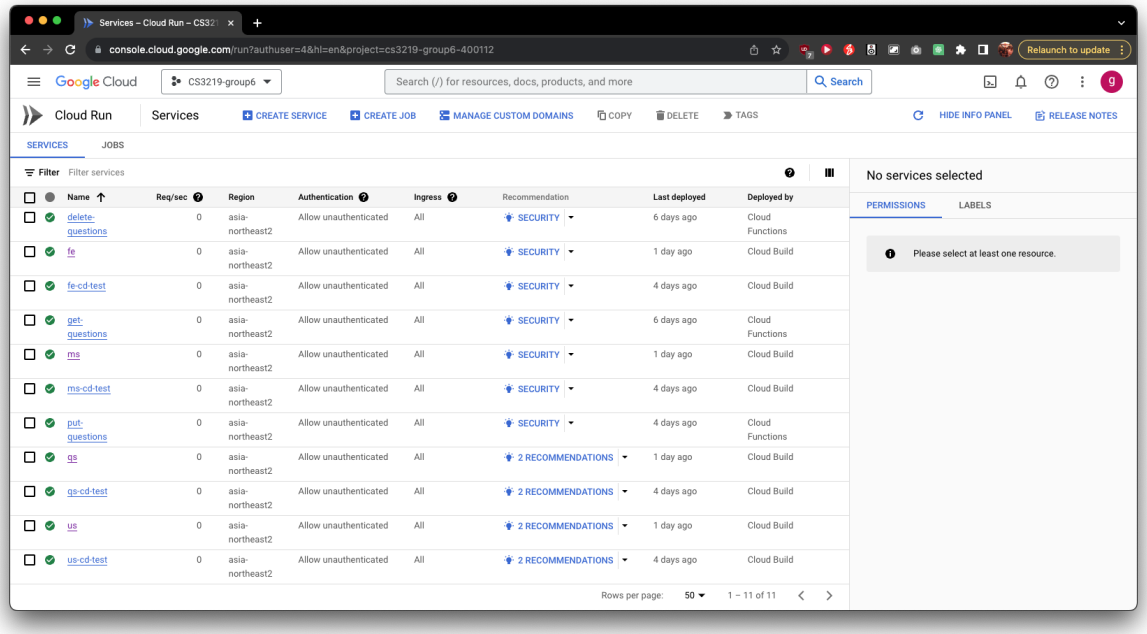

Figure 3.9. Google Cloud Run Interface Showing PeerPrep Microservices Running

We opted for containerized deployment across multiple cloud instances for several compelling reasons. Firstly, containers offer remarkable resource efficiency. They are lighter than Virtual Machines, containing only essential OS processes. This streamlined approach significantly reduces the time required to deploy instances.

Moreover, the portability of containers is a key advantage. Applications are encapsulated during development, ensuring they run consistently across platforms. This flexibility is especially evident when utilising containerization platforms like Docker, allowing seamless operation on any platform that supports containers.

In addition to portability, containers exhibit excellent scalability. By adjusting resources allocated to each service based on metrics provided by GCP, we can easily adapt to changing demands. Furthermore, Cloud Run enhances cost-efficiency by provisioning instances based on actual usage. This feature is especially beneficial for student developers, as it helps minimise costs when services are not actively in use.

In summary, our choice of containerized deployment is driven by the trifecta of resource efficiency, portability, and scalability, all of which align seamlessly with modern DevOps practices and cost-effective cloud utilisation.

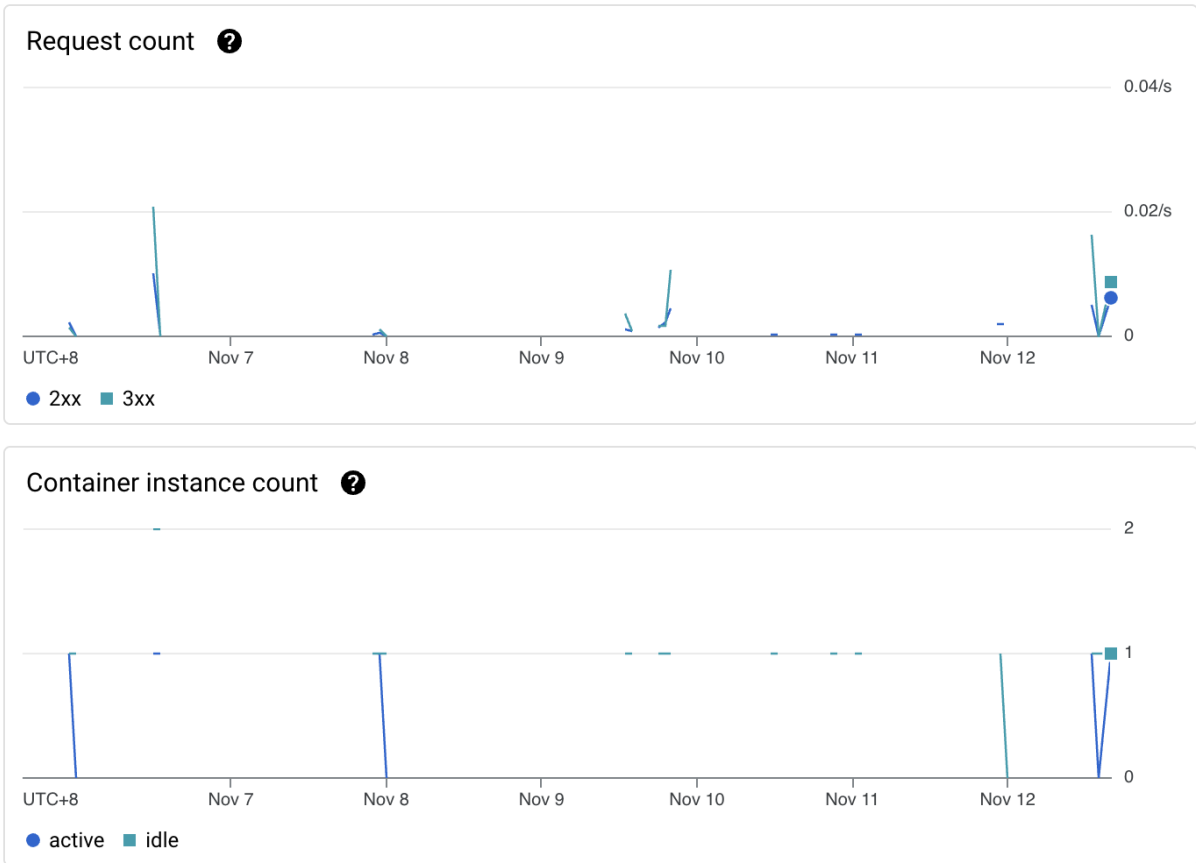

Figure 3.10. Google Cloud Run Interface Logs Showing Instances Used

Google Cloud Build

In our project, Google Cloud Build has been pivotal in automating the build and deployment process of our applications. This choice was primarily driven by its seamless integration with Google Cloud Platform services, enabling us to establish a robust Continuous Deployment (CD) pipeline. With Google Cloud Build, each code to the Github main branch update triggers an automated build process, ensuring that our application gets deployed onto Google Cloud Run seamlessly. This automated process saves valuable development time and resources, allowing our team to focus on more strategic tasks.

Snippet from cloudbuild.yaml

```yaml
steps:
# Build the user-service
- name: 'gcr.io/cloud-builders/docker'
args: ['build', '-t', 'gcr.io/$PROJECT_ID/user-service:$COMMIT_SHA',
'-f', './user-service/Dockerfile.prod',
'./user-service']

# Push the user-service image to Container Registry
- name: 'gcr.io/cloud-builders/docker'
args: ['push', 'gcr.io/$PROJECT_ID/user-service:$COMMIT_SHA']

# Deploy the user-service to Cloud Run
- name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
entrypoint: gcloud
args: ['run', 'deploy', '${_USR_CLOUD_RUN_NAME}',
'--image', 'gcr.io/$PROJECT_ID/user-service:$COMMIT_SHA',
'--region', '${_REGION}',
'--port', '${_USER_PORT}',
"--allow-unauthenticated"
]


# Build the question-service
- name: 'gcr.io/cloud-builders/docker'
args: ['build', '-t', 'gcr.io/$PROJECT_ID/question-service:$COMMIT_SHA',
'-f', './question-service/Dockerfile.prod',
'./question-service']

# Push the question-service image to Container Registry
- name: 'gcr.io/cloud-builders/docker'
args: ['push', 'gcr.io/$PROJECT_ID/question-service:$COMMIT_SHA']
...
```

Figure 3.11. Code Snippet From Cloudbuild.yaml

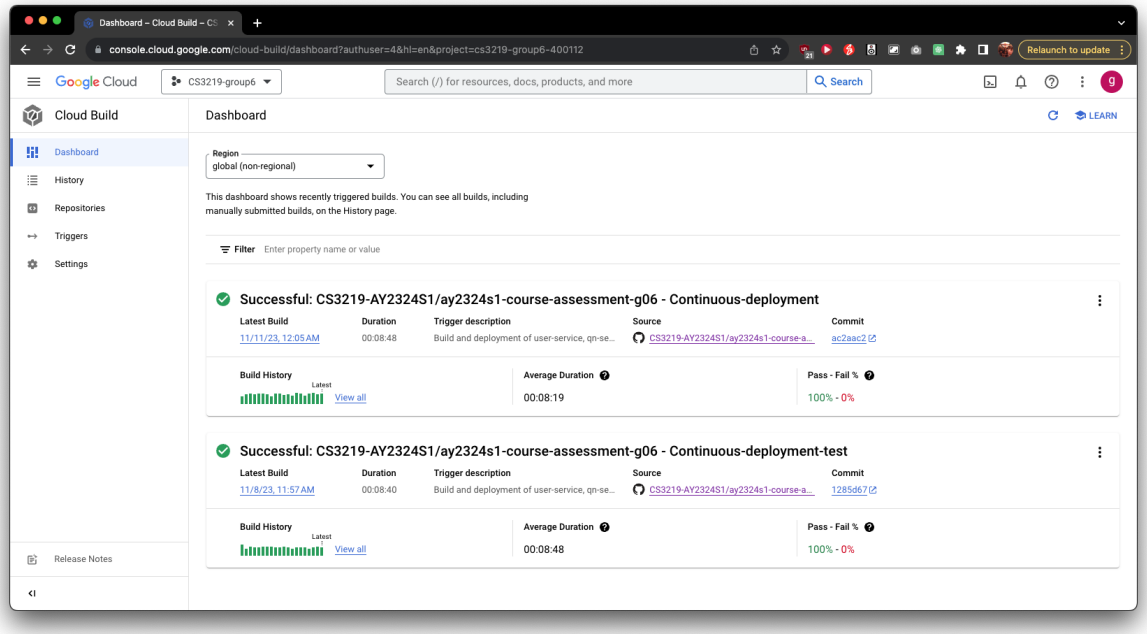

Figure 3.12. Google Cloud Build Interface

A standout feature of Google Cloud Build is its customizable build environments, enabling us to tailor our build processes to meet specific project requirements. This flexibility allows us to define custom build steps, use various programming languages,

and specify different environments and configurations, making it adaptable to our diverse development needs.

Continuous Deployment (CD) is an integral part of our workflow, where each time our codebase undergoes a change in the main branch, Google Cloud Build compiles the source code and builds the Docker image, which is then stored in Google's Container Registry. From there, Google Cloud Run retrieves the latest image and deploy them, ensuring our services are up-to-date with our latest codebase. We also utilised Cloud Build to execute deployment tests on various branches.

Google Cloud SQL

The choice of Google Cloud SQL as our database management service was driven by several key factors that align with our project's requirements, particularly with scalability and manageability in mind. Google Cloud SQL offers a fully managed database service, which means that the overhead of database administration, such as updates, patches, and backups, is handled by Google. This allows our development team to focus more on application development rather than database maintenance. Google Cloud SQL offers scalability and high availability through the features Google Cloud SQL provides, which is crucial for maintaining peak performance during peak usage, as well as ensuring our application remains accessible and responsive even during infrastructure failures, maintenance windows and unexpected traffic surges.

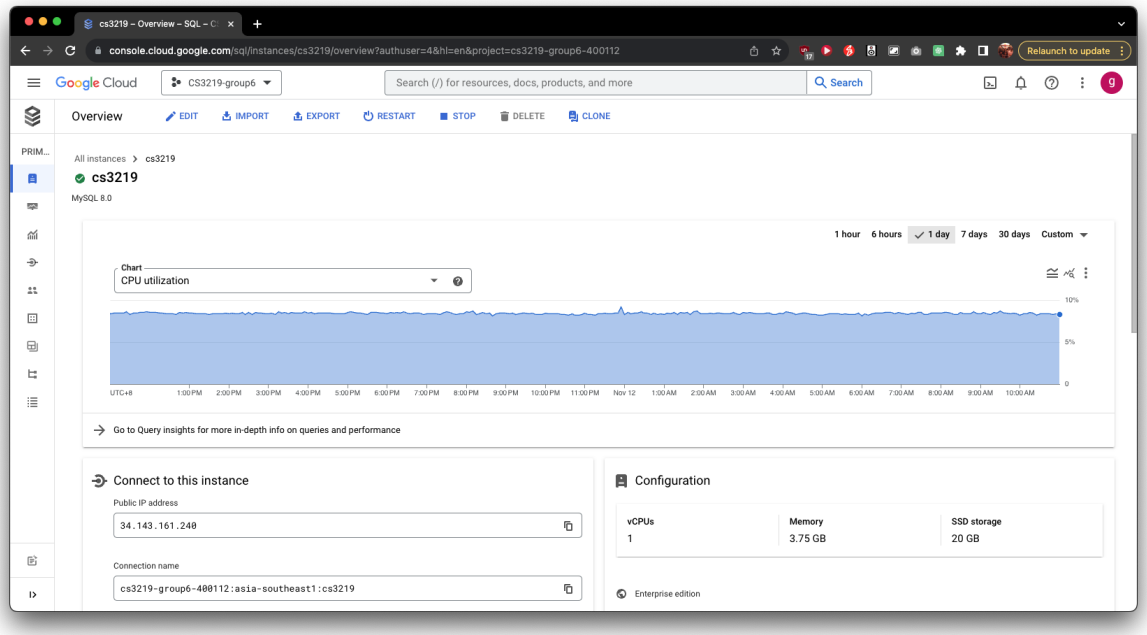

Figure 3.13. Google Cloud SQL Interface

Google Cloud SQL supports standard SQL, which means that there is no need for our team to learn new database languages or adapt existing SQL-based applications significantly. The database service not only provides performance insights and recommendations through Cloud SQL Insights, making it easier to optimise query performance and resource utilisation without the need for deep database expertise, it also provides options to locate databases in regions close to the user base, which minimises latency and improves the responsiveness of the application.

3.7. API Testing

Postman

Postman has been an invaluable tool in testing the APIs we developed for PeerPrep in their respective microservices domain. It provides a comprehensive platform for quickly creating, sharing, testing, and documenting APIs. This has enabled us to ensure that our APIs perform as expected before they are integrated into our application. By utilising Postman, we can simulate client requests and analyse the responses from our server endpoints without the need for a frontend interface.

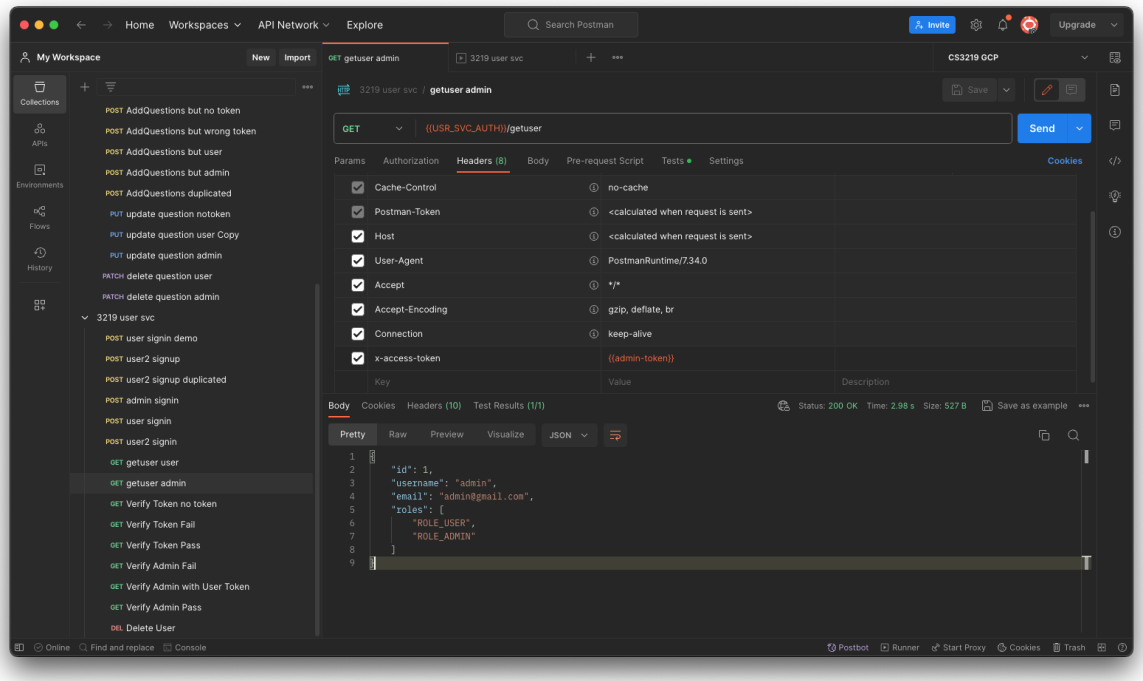

Figure 3.14. Postman Interface Showing API Calls to user-service

In our testing, we heavily utilised the environment feature. By setting up different environments for local and cloud deployment, we can manage and switch between different sets of variables, including JWT Tokens and URLs, ensuring that the correct

parameters are used for each environment, maintaining consistency and accuracy in our API testing process across various stages of deployment. Postman also has automated API testing runs, where we can monitor our deployment and work accordingly.

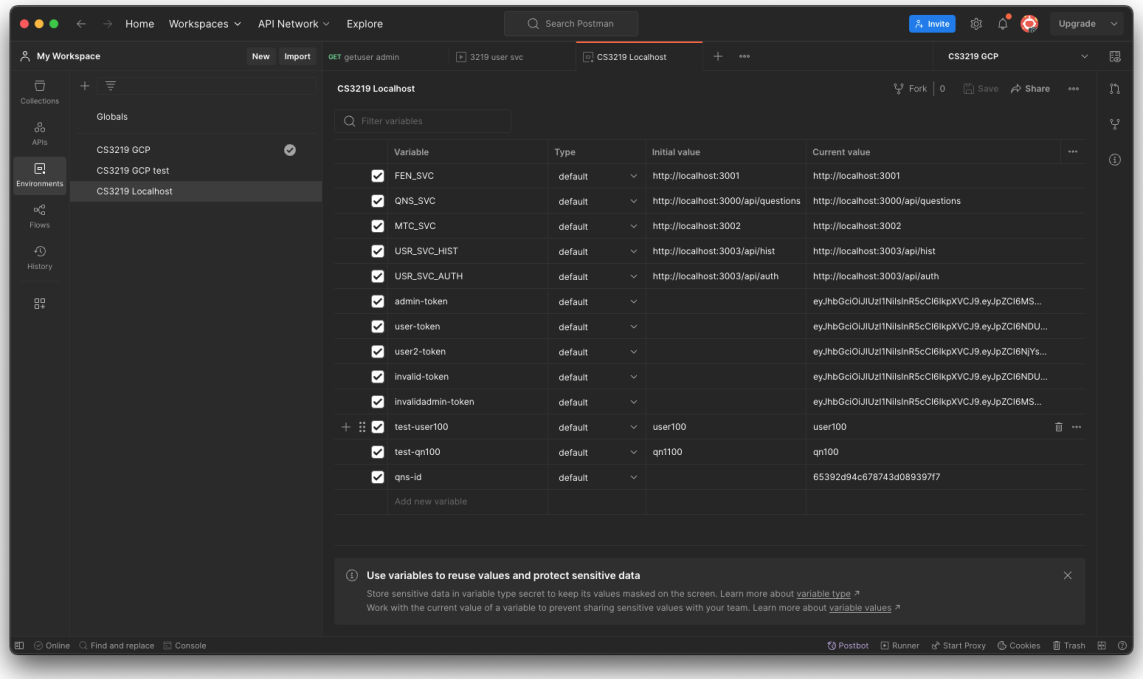

Figure 3.15. Postman Environment Showing Different Stored Environment Variables

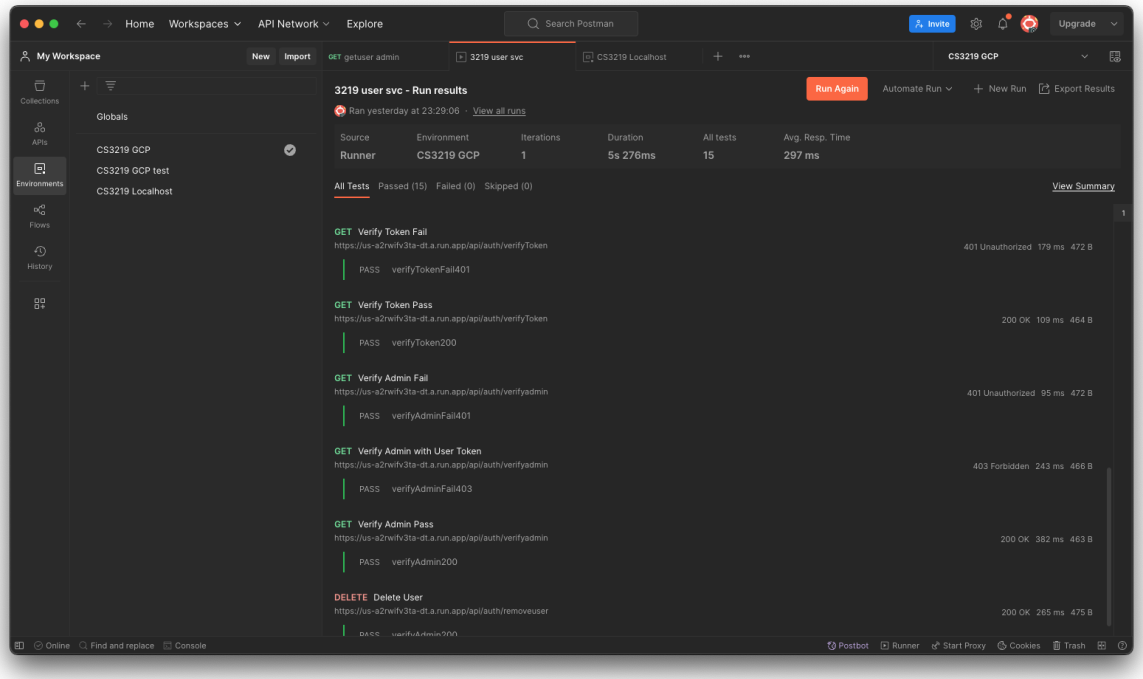

Figure 3.16. Postman Interface Showing Automated Testing of API Routes

Chapter 4. Project and People Management

4.1. Organization of Team

Our team is organised into two sub-groups, where each group handles a set of features highlighted from the must-have and a subset of handpicked nice-to-have features as referenced from the CS3219 project instructions. However, there is no strict boundary on each member's responsibilities within the subgroup, and our team has made great progress by lending a helping hand to one who requires help if they are more knowledgeable in the field. Below are the sub-groups and the features they are mainly responsible for:

Group	Names	Must-Have Features Responsible For	Nice-To-Have Features Responsible For
Sub-Group 1	Loo Wei Zhe Evan	M1, M5, M6	N2, N5, N9
	Wong Qin Yao		
Sub-Group 2	Ang Jun Kang	M2, M3, M4, M5	N1, N3, N4
	Chong Yijing Isabel		
	Kalaris Ng Yun Shan		

Figure 4.1. Sub-Group And Features

With the issues assigned, the feature management is managed through a popular project management software - Jira (Refer to Chapter 3.1). This software is chosen as it has an easy to use interface, providing us with versatile planning and tracking capabilities that aligns with the agile methodologies adopted by our development team. Jira aided us with backlog management and prioritisation, playing a pivotal role in streamlining our feature management process and ensuring they align with our project's goals and timelines. Our team used Jira to maintain a transparent and organised workflow and enhance collaboration, enabling us to deliver quality features in a timely manner.

4.2. Milestones Planning

As part of the CS3219 module, the management of the speed of PeerPrep's progress is done with compliance with the module's milestone deadline. The module milestones are as follows.

Milestone	Completion Date	Expectations
1	End of Week 7	Delivering minimum viable product (MVP or proof of concept) containing the User Service, Question Service and Matching Service.
2	End of Week 9	Complete development of collaboration service and one nice-to-have per sub group. Mandatory project meeting where we demonstrate to mentor.
3	15 Nov 2023	Final implementation along with documentation and demonstration. Deliverables include all the must have and nice to have features implemented, integrated, tested and deployed following Mx and selected Nx requirements

Figure 4.2. Module Milestones

4.3. Individual Contribution

This section showcases individual contributions of each member in our team.

Entity/Developer	Ang Jun Kang	Chong Yijing Isabel	Kalaris Ng Yun Shan	Loo Wei Zhe Evan	Wong Qin Yao
Frontend	✔	✔	✔	✔	✔
Basic User Service			✔	✔	✔
Basic Question Service	✔	✔	✔		✔
Basic Matching Service	✔	✔	✔		
Chat	✔	✔			
Code History		✔	✔	✔	
Code Execution		✔			
Improved Question Management	✔		✔		✔
Improved Code Editor	✔			✔	

Docker / Deployment				✔	✔
Assignment Video Recording					✔
Assignment Submission	✔			✔	✔
Documentation Management		✔	✔	✔	

Figure 4.3. Individual Contributions

Chapter 5. PeerPrep Features

This chapter showcases a comprehensive high-level view of the features implemented in PeerPrep, followed by the functional, and non-functional requirements of the application.

5.1. Must-Have Feature

As part of the course requirements, our project is anchored by these essential features.

Feature Number	Must-Have Feature
M1	User Service - responsible for user profile management.
M2	Matching Service - responsible for matching users based on some criteria (e.g. difficulty, level of questions, topics, proficiency level of the users, etc.) This service can potentially be developed by offering multiple matching criteria.
M3	Question Service - responsible for maintaining a question repository indexed by difficulty level (and any other indexing criteria - e.g. specific topics)
M4	Collaboration Service - provides the mechanism for real-time collaboration (e.g. concurrent code editing) between the authenticated and matched users in the collaborative space.
M5	Basic UI for user interaction - to access the app that you develop.
M6	Deploying the application on your local machine (e.g. laptop) using native technology stack or on a (local) staging environment (e.g. Docker-based).

Figure 5.1. Chosen Must-Have Features

5.2. Nice-To-Have Features

As part of the course's requirements, we have chosen 6 extra nice-to-have features (Refer to Appendix A) on top of the must-have features . The chosen nice-to-have features are listed below.

Feature Number	Chosen Nice-To-Have Feature
N1	Communication: Implement a mechanism to facilitate communication among the participants in the collaborative space (other than the shared workspace). With progressive levels of difficulty: text-based chat service, voice calling service, video (+voice) calling service.
N2	History: Maintain a record of the questions attempted by the user. With progressive levels of difficulty: maintain a list of questions attempted along with the date-time of attempt, maintain a list of questions along with the attempt, maintain a list of questions along with the attempt and retrieve correct answers.
N3	Code execution: Implement a mechanism to execute code in a sandboxed environment, and retrieve+present the results in the collaborative workspace.
N4	Enhance question service to enable managing questions, for example, tagging (by topic, popularity, etc.,), retrieving questions on the fly during a session initiation.
N5	Enhance collaboration service by providing an improved code editor with code formatting, syntax highlighting for one language, syntax highlighting for multiple languages.
N9	Deployment of the app on the production system (AWS/GCP cloud platform).

Figure 5.2. Chosen Nice-To-Have Features

It is important to also mention that our team has chosen to add on with an additional feature of Continuous Deployment (CD), which is a partial feature under the Nice-To-Haves as defined by the course instructions. Further details on why the features are chosen and how they are implemented will be explained under Chapter 7.

5.3. Functional Requirements

The functional requirements specify PeerPrep's expected behaviour and functions and are crucial in our software development and system design process as they guide us in building the necessary functionalities to meet user or business needs.

Product Name: Peerprep		
	Functional Specification	Priority
F1	User Authentication / Service	
	F1.1 The application should allow registration of new users.	H
	F1.2 The application should ensure that every account has a unique username and email.	H
	F1.3 The application should allow users to login.	H
	F1.4 The application should allow users to log out of their account.	H
	F1.5 The application should allow users to delete their account.	H
	F1.6 The application should allow users to stay logged in when they refresh the browser.	H
	F.1.7 The application should block logins when wrong credentials are used.	H
	F1.8 The application should keep track of a record of questions attempted by the user in descending order.	H
	F1.9 The application should distinguish between an admin and a regular user.	H
	F1.10 The application should allow users to change their password.	M
	F1.11 The application should allow users to retrieve their past attempts.	M
	F1.12 The application should be able to churn out statistics on past submissions done by a user.	M
	F1.13 The application should validate user registration attempts for correct email format and secure password criteria.	L
F2	Matching Service	
	F2.1 The application should allow users with the same language, difficulty level and topic to be matched together.	H
	F2.2 The application should decide how users with similar priority will be chosen for the 2-person match.	H
	F2.3 The application should inform the user and allow a rematch if no match is found after a fixed time.	H

Product Name: Peerprep		
	Functional Specification	Priority
	F2.4 The application should provide a timer to indicate how long the user has been in the matching queue.	H
	F2.5 The application should only show topics available for matching.	H
	F2.5 The application should give the matched users the same question to work on.	H
F3	Question Service	
	F3.1 An admin should be able to perform CRUD (Create, Read, Update, Delete) operations on the application.	H
	F3.2 The application should showcase a database of available questions for both the admin and users to read/view.	M
	F3.3 The application should allow users to choose to attempt questions of different difficulty levels (easy, medium, hard).	M
	F3.4 The application should allow users to select the topic of the question they want to attempt.	H
	F3.5 The application should randomise the questions given for each pair.	M
	F3.6 The application should be seeded with a list of questions.	M
F4	UI for User Interaction	
	F4.1 The application's UI should be intuitive and easy to use.	H
	F4.2 The application's home page should display relevant statistics for the user to view on the home page.	M
	F4.3 The application's home page should display the questions that have been attempted by the users.	M
	F4.4 The application's home page should display history statistics (eg. number of questions attempted) of the user.	M
	F4.5 The matching interface should be intuitive to the chosen selections for topics and selections.	H
F5	Collaboration System	
	F5.1 The application should provide concurrent code editing between matched users.	H

Product Name: Peerprep		
	Functional Specification	Priority
	F5.2 The application should load the question up in the code space.	M
	F5.3 The application should allow users to quit the collaboration session.	H
	F5.4 The application should inform the paired user when the other user chooses to quit the collaborative space.	M
	F5.5 The application should allow users to run their code in the code space with no noticeable delay.	M

Figure 5.3. Functional Requirements

5.4. Non-Functional Requirements

Non-functional requirements complement the functional aspects of PeerPrep by detailing the characteristics, constraints, and qualities that shape the overall performance and user experience of the system. These requirements encompass key factors such as usability, performance, and deployability. While functional requirements address what the system should do, non-functional requirements define how well it should do it. Considering these non-functional aspects is integral to ensuring that PeerPrep not only meets user needs but also excels in terms of reliability, efficiency, and overall quality.

Product Name: Peerprep		
	Non-Functional Specification	Priority
NF1	Usability	
	NF1.1 The UI should have a consistent colour scheme.	M
	NF1.2 The application should be accessible via a public URL.	H
	NF1.3 The application should have a responsive UI.	H
	NF1.4 The application should support a minimum resolution of 1024 x 768 px.	H
NF2	Performance	
	NF2.1 The application should be able to handle at least 2 people using it at once.	H

Product Name: Peerprep		
	Non-Functional Specification	Priority
	NF2.2 The application should take less than 5 seconds to load in the questions for the match.	H
	NF2.3 The application should take no longer than 5 seconds to log in.	H
	NF2.4 The system should respond to each request within 5 seconds.	M
NF3	Deployability	
	NF3.1 The application should be able to be rolled back to previous versions.	H
	NF3.2 The application should be deployed on the cloud.	H
	NF3.3 The application should be automatically deployed based on the latest version.	M
	NF3.4 The application should be deployed within 15 minutes after updating to the latest version.	L
	NF3.5 The application should be deployed with health checks and monitoring capabilities.	L

Figure 5.4. Non-Functional Requirements

In order to systematically address the non-functional requirements (NFRs) identified for PeerPrep, it is imperative to establish a clear hierarchy of priorities. This involves assigning scores to each quality attribute, thereby providing a roadmap for development that aligns with the project's goals. A comprehensive evaluation was conducted, taking into account factors such as user expectations, project constraints, and possible business objectives.

Below is the prioritised list of NFRs along with corresponding scores, reflecting their significance in the context of our project.

Attribute	Score	Availability	Integrity	Performance	Deployability	Scalability	Security	Usability	Compatibility
Availability	4		<	^	^	<	<	^	<
Integrity	3			^	^	<	<	^	<
Performance	6				<	<	<	^	<
Deployability	5					<	<	^	<
Scalability	2						<	^	<
Security	1							^	<
Usability	7								<
Compatibility	0								

Figure 5.5. Quality Attribute Prioritisation of PeerPrep

1. **Usability**: At the forefront of PeerPrep's quality attribute is usability, where we strive to create an intuitive and frictionless user experience, ensuring that PeerPrep is easy to navigate and user-friendly. A high usability score signifies a focus on creating an interface that facilitates easy navigation, content discovery, and seamless collaboration, enhancing the overall user experience
2. **Performance**: Performance is a close second, given that PeerPrep is a technical interview preparation platform, performance is critical for providing a seamless and responsive user experience. Optimal execution speed ensures that users can navigate the platform efficiently, access resources quickly, and engage in real-time collaborative activities without latency.
3. **Deployability**: Deployability is key to our ability to quickly and reliably roll out new features and updates to PeerPrep. A robust score in deployability underscores PeerPrep's agility in responding to user feedback and fixing issues without disrupting the user experience. While not the most critical attribute at present, deployability is crucial for PeerPrep in its early stages, where frequent iterations and enhancements are necessary to refine the platform. This quality attribute ensures that the platform can evolve rapidly, seamlessly integrating new functionalities and improvements in response to the changing needs of users
4. **Availability**: In the context of PeerPrep, availability is paramount to ensure that users can access the platform whenever they need to prepare for technical interviews. A substantial score in availability underscores the importance of

PeerPrep being operational 24/7, accommodating possible users across different time zones and schedules.

5. **Integrity**: While integrity may have a lower score, it remains crucial in PeerPrep's context. Maintaining data integrity is essential to ensure that the learning materials, user profiles, and collaborative histories are accurate and reliable. The balance struck reflects the need to uphold integrity without compromising other critical attributes.
6. **Scalability**: Acknowledging the need for scalability is vital for PeerPrep as it may experience varying levels of user activity. While not the top priority, the lower score indicates an awareness of scalability requirements to accommodate potential growth in user numbers and resource demands should the project grow in the future.
7. **Security**: Security, though crucial, is assigned a lower score to balance with other competing priorities. In PeerPrep, ensuring the security of user data, login credentials, and collaborative interactions is important but is balanced against the need for an accessible and performant platform. We still managed to implement Secure Communication (HTTPS) on all our cloud services to prevent CSRF attacks and Authentication to prevent unauthorised deleting of questions.
8. **Compatibility**: With the lowest score, compatibility considerations are recognized but deemed less critical in the current project landscape. In the context of PeerPrep, while compatibility is acknowledged, the emphasis is placed on other attributes given the nature of the platform as an interview preparation tool.

We have performed a comprehensive testing or showcasing of the NFRs in Appendix B which you can find at the end of the report. This testing process involved evaluating various aspects such as production cloud deployment to ensure that PeerPrep meets the specified non-functional criteria. The results and insights gained from this testing phase significantly contribute to the robustness and reliability of our system, affirming our commitment to delivering a high-quality and well-performing application.

Chapter 6. Application Walkthrough

This chapter provides a step-by-step exploration of the PeerPrep application. We guide users through the essential components and features of the application, enabling users to become familiar with the application's user journey from the very beginning.

1. Welcome, Sign Up and Login Page

Upon accessing the application, users are greeted by our inviting welcome page, designed to be the central starting point for their journey. Here, users are provided with clear options to either sign up as a new user or log into the application if they already have an account. This uncomplicated entry point ensures that users can swiftly embark on their interaction with the platform.

Figure 6.1. PeerPrep Welcome Page

Upon clicking on the Login tab from the navigation bar or the button from the welcome page, users are directed to input their username and password for login.

Figure 6.2. PeerPrep Login Page

And on the “Sign Up” page, users are guided by the system to set up appropriate username, emails and password. Some restrictions are in place here, such as a fixed length of 6 and 40 characters for the password. Users identify each other by their usernames in the application, and hence checking for unique usernames are also in place.

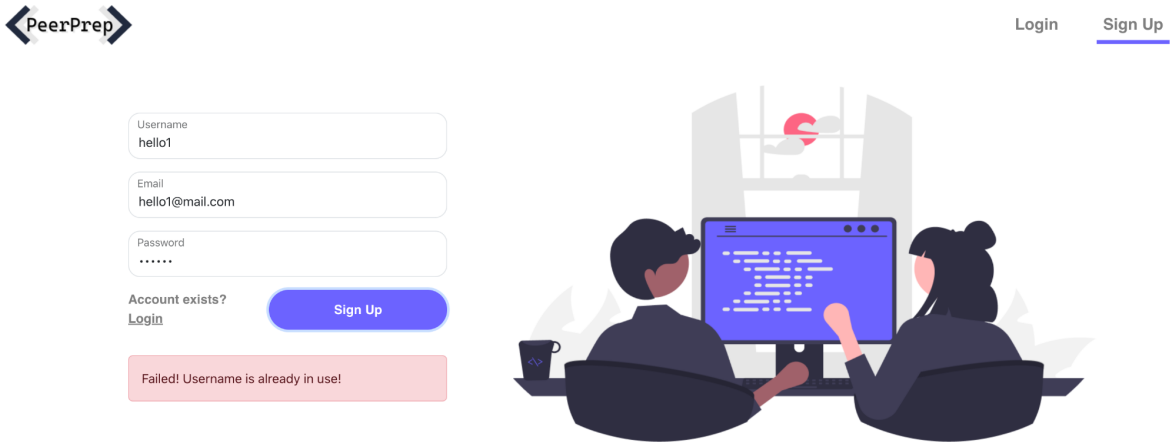

Figure 6.3. PeerPrep Sign Up Page Checking for Duplicate Username

2. Home Page

Upon login, users are greeted by the "Home" page for a comprehensive overview of their training efforts. This page showcases their progress through informative question statistics and a heatmap, providing insights into their performance. Additionally, the question history section displays details of their past attempts, allowing users to review their progress and access their latest solutions for specific questions. This

comprehensive view enhances the learning experience and enables users to track their growth over time.

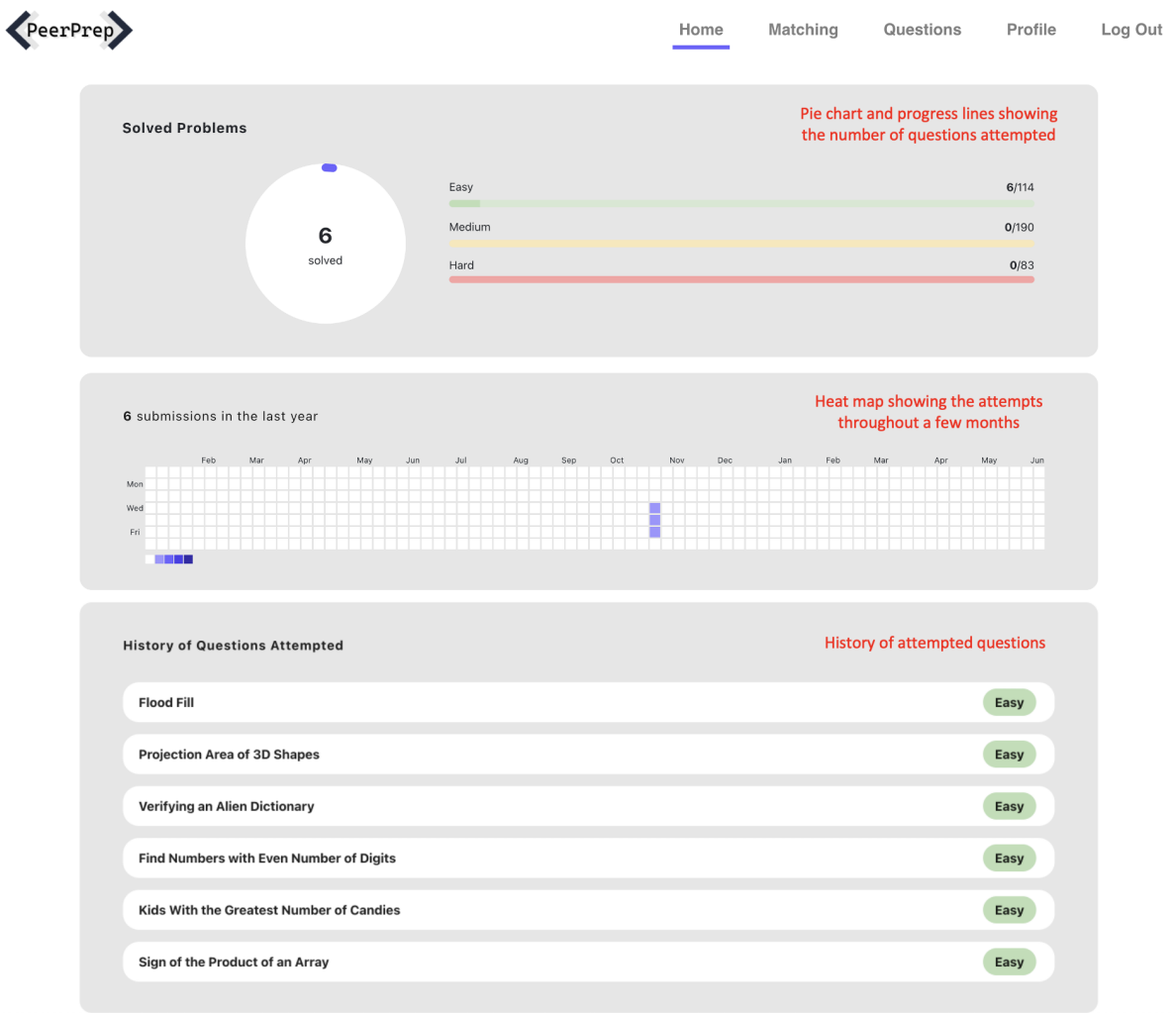

Figure 6.4. PeerPrep User Home Page

3. Matching and Collaboration Page

The "Matching" tab introduces an interactive and engaging experience. Here, users have the exciting opportunity to tailor their coding practice to their preferences. They can fine-tune their experience by choosing specific topics and setting their desired difficulty level for coding challenges. After configuring these preferences, users need only click on the "Match" button, initiating a dynamic process in which the application seeks out a compatible peer with similar interests and skill levels. This feature adds a layer of gamification and excitement to the user experience, further encouraging collaboration and learning through peer-to-peer interaction. It's essential to note that if there's no immediate match within 30 seconds, users have the option to explore

alternative topics and difficulty levels to broaden their chances of finding a suitable peer.

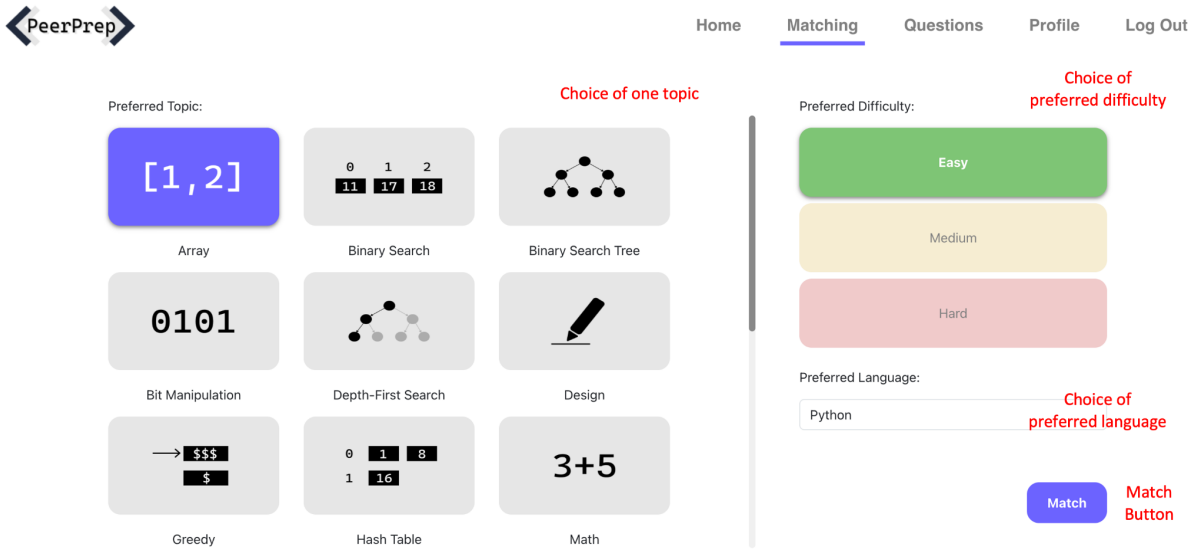

Figure 6.5. PeerPrep User Matching Page

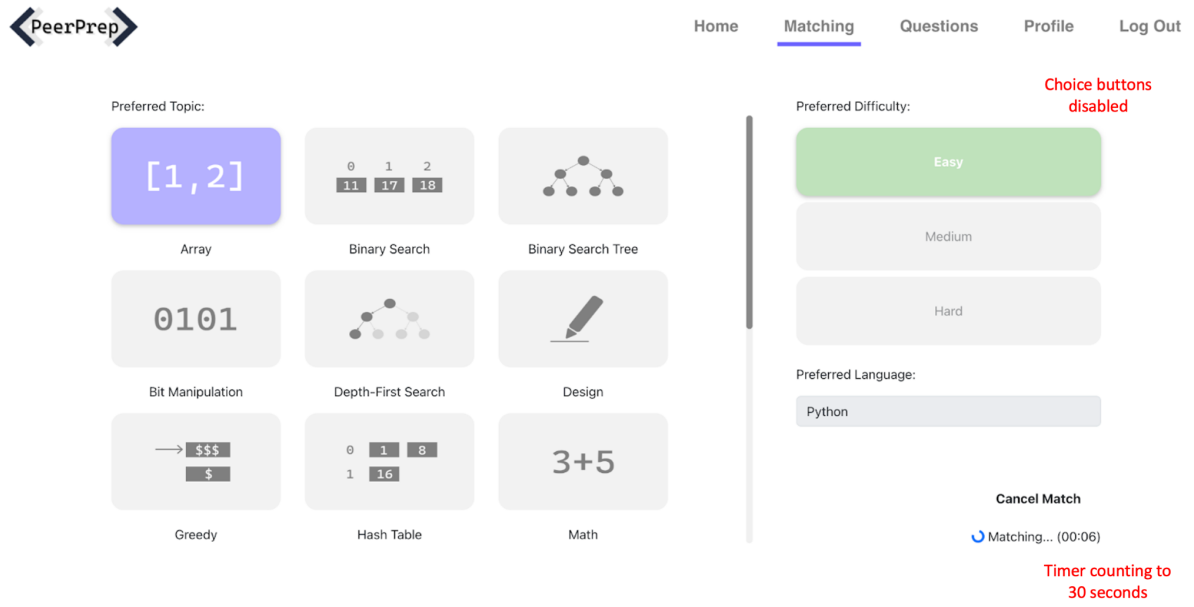

Figure 6.6. PeerPrep User Matching Page During Matching Phase

Upon successful matching, both users are seamlessly transported to a dedicated and private room. Here, they can communicate directly through an integrated chat interface and concurrently work on coding challenges within a shared code execution environment. The real-time code execution feature allows users to run their code, with results visible to both parties in the room. Once they are content with their code, they

can opt to submit their collaborative attempt, with both users agreeing on the submission. In the event one user decides to exit the session prematurely, the other user has the choice to either continue coding independently or leave the session.

In the spirit of fostering a productive and collaborative coding experience, the timer also acts as a tool for users to manage their time effectively. The addition of a timer in the collaborative coding session introduces an element of urgency and gamification. This timer creates a dynamic environment where users must complete their coding challenge within a specified time frame. Once the timer runs out, users are presented with a choice: they can either submit their code or choose not to, and subsequently, they can exit the session. This time-constrained aspect not only adds an element of excitement but also encourages users to develop valuable skills in time management and collaboration, all while making the learning experience engaging and interactive.

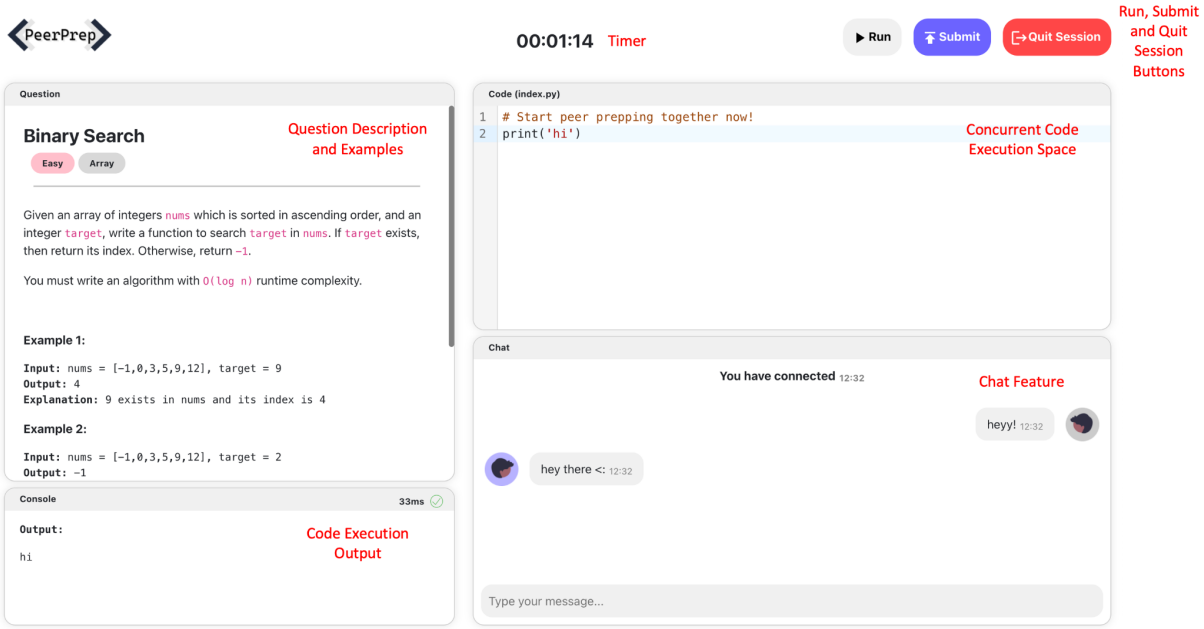

Figure 6.7. PeerPrep User Collaboration Space

After the user chooses to submit the question, they can view their latest attempt at the question from their home page as shown in Figure 6.4. Upon clicking on a question they have attempted, the code attempt can be seen at the bottom of the page.

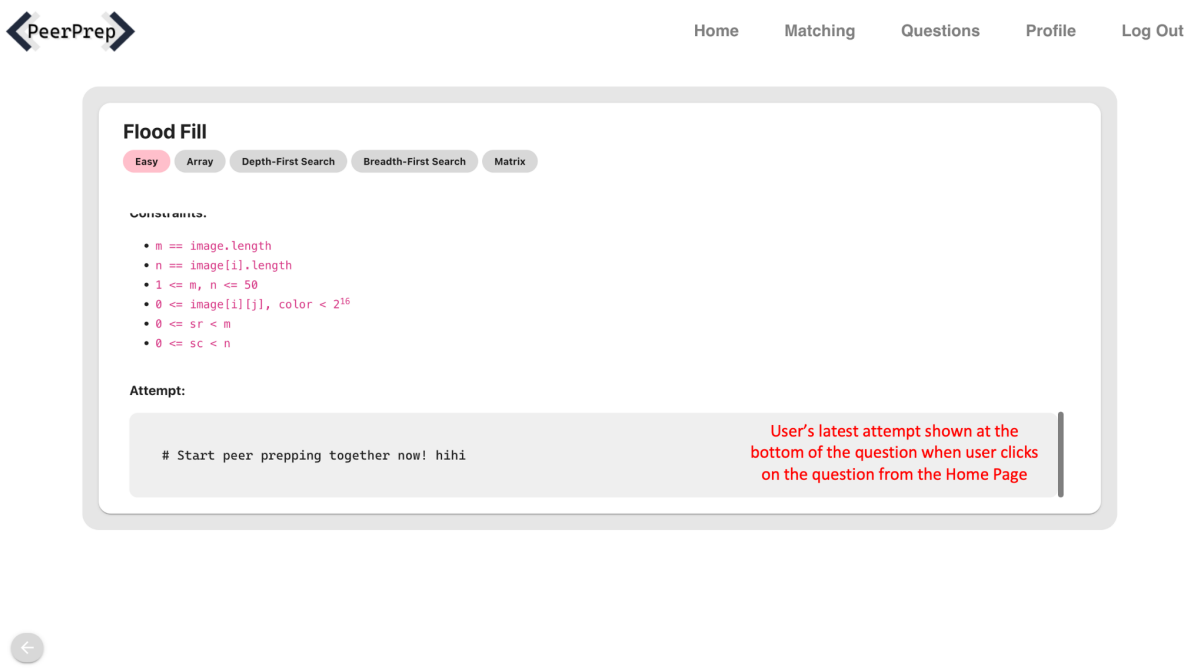

Figure 6.8. PeerPrep User History Attempt

4. Questions Page

Additionally, users can enter the richly featured "Questions" tab, a treasure trove of coding challenges and exercises. In this section, they have the freedom to explore a vast library of questions, each thoughtfully categorised and tagged for easy navigation. Users can browse through the collection to pinpoint questions that align precisely with their interests and objectives. Notably, when an admin is logged in, they wield enhanced privileges within this tab. Admins possess the authority to create, update, and, if necessary, delete questions. This empowerment bestows upon them a pivotal role in content management, ensuring the quality and relevance of the application's challenges.

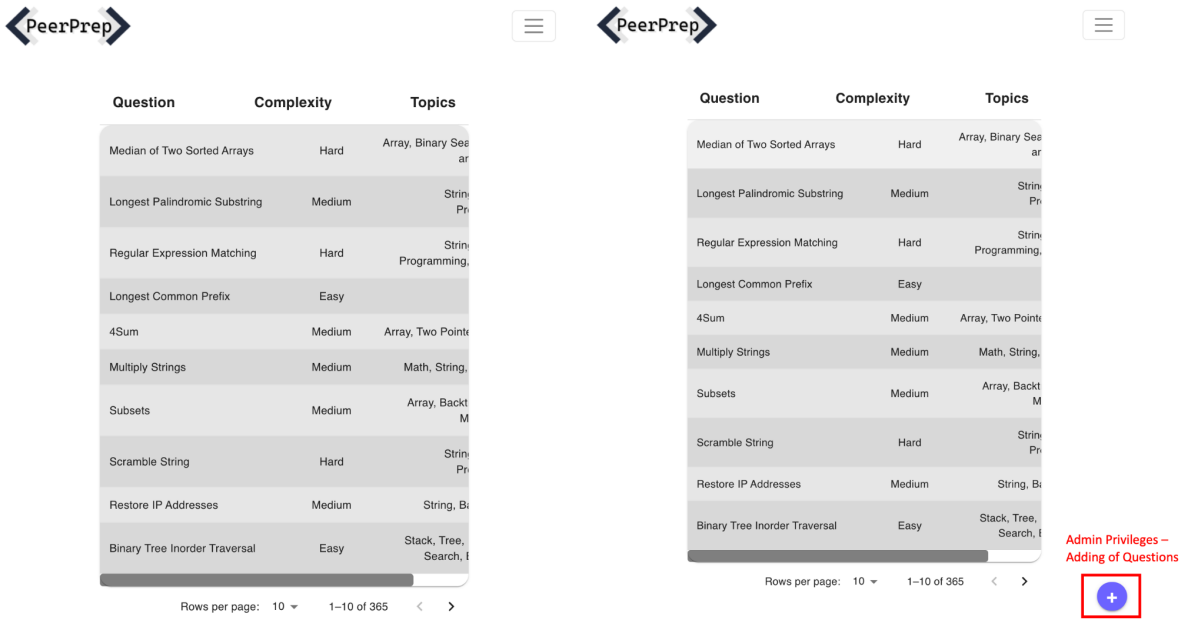

Figure 6.9. Comparison of User and Admin Privileges on Question Tab

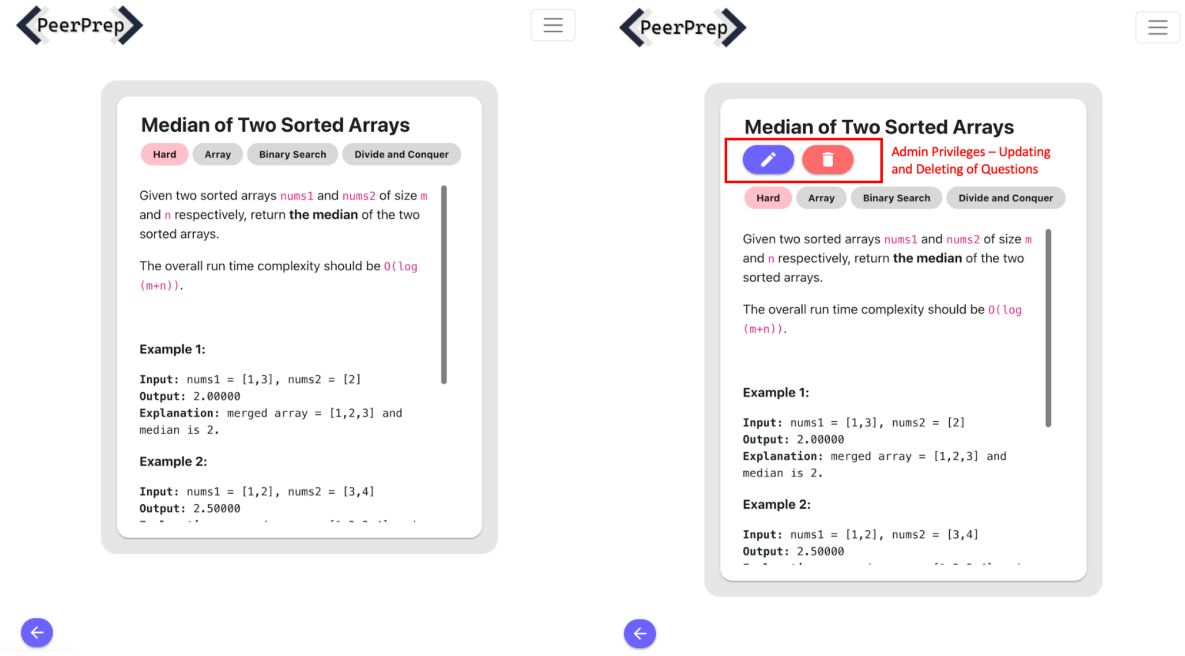

Figure 6.10. Comparison of User and Admin Privileges on Question View

In our question editor, it is equipped with a feature-rich text editor so that the admin can beautify the questions with syntax such as bolding and italicising, adding of questions and links, and so on.

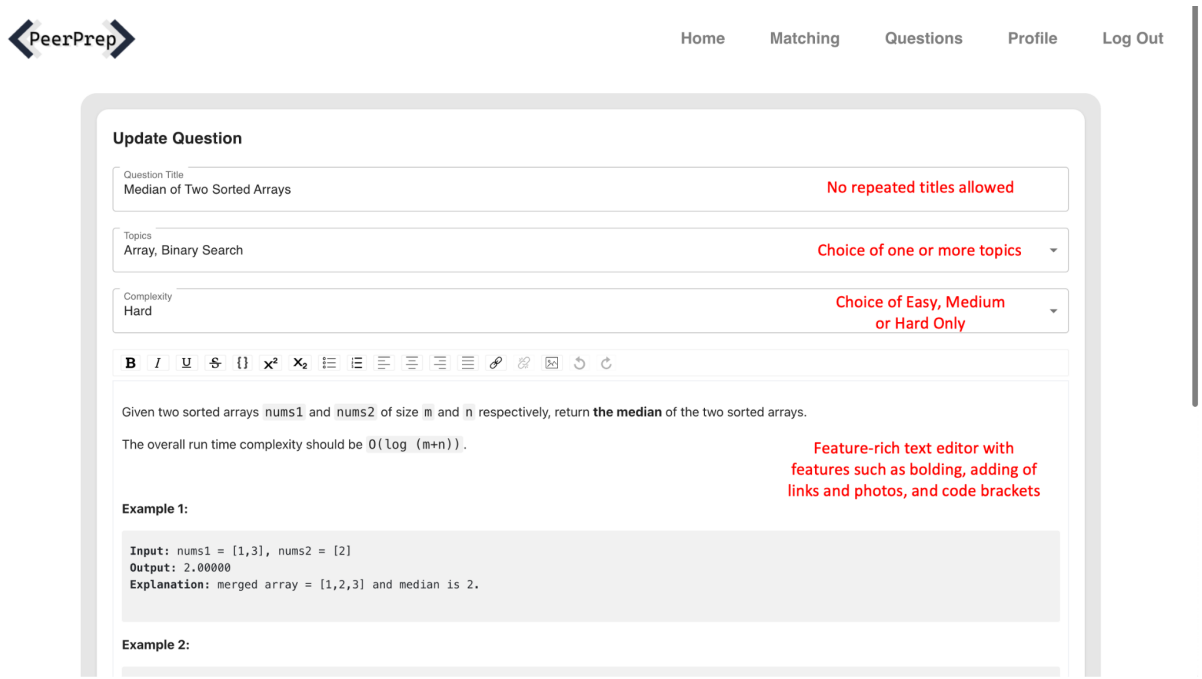

Figure 6.11. PeerPrep Update Questions Form

And as a last check, we have also implemented various prompts throughout the application such as when an admin chooses to delete a question. These prompts will allow the admin to confirm, or rethink their decision to delete a question.

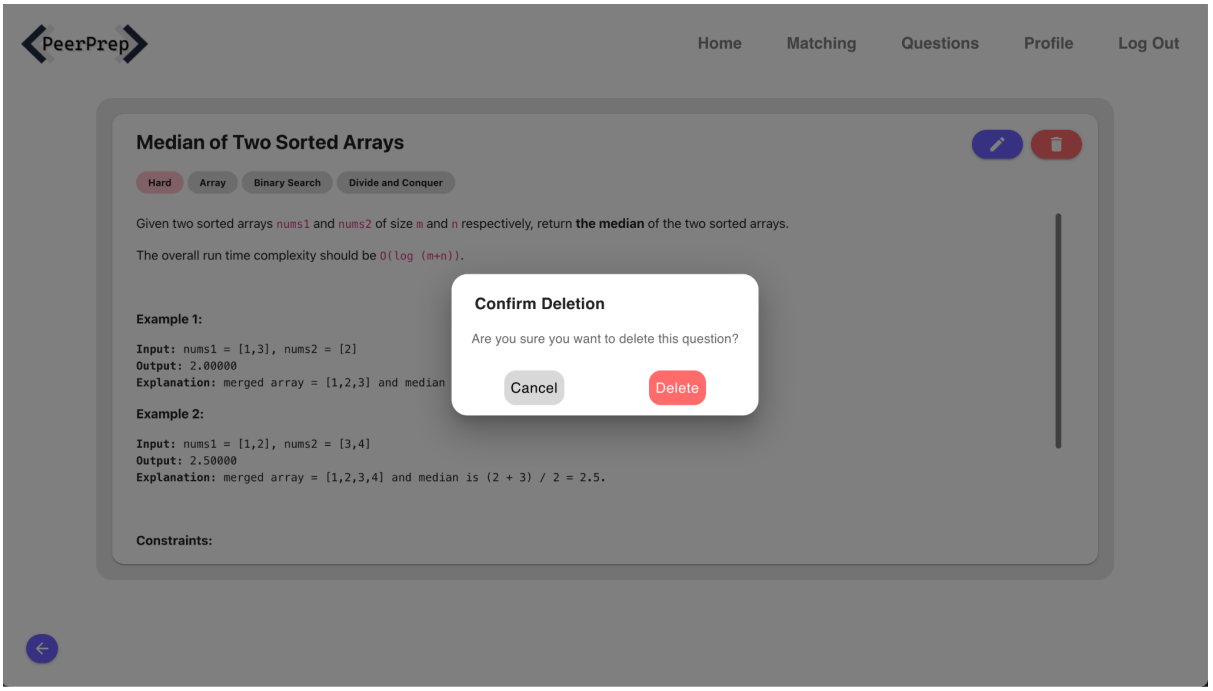

Figure 6.12. Application Prompt on Admin Question Deletion

5. Unauthorised Access

The PeerPrep application robustly addresses unauthorised access scenarios by implementing stringent access controls. Unauthorised users attempting to access member privileges or non-admins attempting to gain admin rights are effectively managed.

Member privileges encompass a range of features, including the ability to view homepage analytics, review attempted question history, access a comprehensive list of all questions, view and edit their profile page, and facilitate collaboration by matching with other users for coding collaborations.

On the other hand, admin privileges extend beyond member privileges. Admins not only enjoy all member privileges but also possess additional capabilities. They have the authority to edit, update, and create questions, thereby ensuring comprehensive control and oversight over the platform's content and functionality.

This careful delineation of member and admin privileges enhances security and control, ensuring that only authorised users have access to specific functionalities, maintaining the integrity and purpose of the PeerPrep application.

When a non-member tries to access a page requiring member privileges, they are met with a “401 Unauthorised” error page. This is handled by the backend and as such blocked on the frontend.

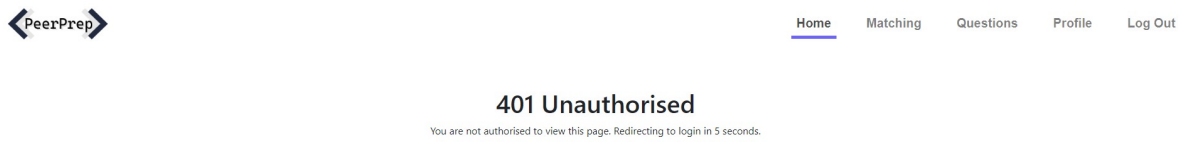

Figure 6.13. HTTP 401 Unauthorised Error For User Accessing “Matching Page” With Invalid Token / Non-Member Privileges

Similarly, when a member tries to access admin pages such as adding a question, they are met with a “403 Forbidden” error page informing users that they are accessing an action or page which they are not authorised to do so. This is because only the admin should handle an important feature question repository management.

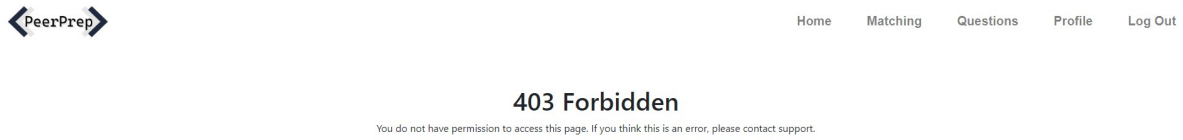

Figure 6.14. HTTP 403 Forbidden Error For User Accessing “Add Question” Page Requiring Admin Privileges

Chapter 7. Service Integration Libraries and Decisions

This chapter offers a comprehensive exploration of the services integrated into the application's architecture, delving into the underlying decisions that govern the selection of libraries for each service. It not only catalogues the libraries employed but also provides an insightful analysis of why certain libraries have been specifically chosen for individual services.

7.1. User Interface

Bootstrap

Bootstrap plays a pivotal role in shaping the overall aesthetics and responsiveness of our user interface. Leveraging Bootstrap's extensive collection of pre-designed components and a responsive grid system, we ensure a consistent and visually appealing design across various devices and screen sizes. The ease of customization and the robust set of styling options provided by Bootstrap contribute to the seamless creation of a modern and user-friendly interface.

Material-UI (MUI)

Complementing Bootstrap, we integrate Material-UI (MUI) to enhance the visual appeal and user interaction of our application. Material-UI, built on the principles of Google's Material Design, offers a rich set of React components that bring a polished and intuitive design language to our user interface. The use of Material-UI components not only adds aesthetic consistency but also introduces interactive elements that align with modern design standards, elevating the overall user experience.

The decision to blend Bootstrap and Material-UI is rooted in a thoughtful consideration of their respective strengths. Bootstrap provides a robust foundation for layout and responsiveness, while Material-UI contributes a sleek and interactive design layer. This synergistic approach allows us to harness the strengths of both libraries, resulting in a visually cohesive, responsive, and user-centric interface.

7.2. User Service

Our team first deliberated on the choice of database and methods of authentication, ultimately settling on Sequelize for database management and JWT tokens for secure authentication.

7.2.1. User Profile and History Storage

Sequelize

We initially chose Sequelize to store users' information such as their username, email and password. Later on, we extended Sequelize to also store user history information, such as individual user attempts. Sequelize offers a promise-based Node.js ORM (Object-Relational Mapping) for SQL databases and provides a robust framework for database access and manipulation. These benefits allowed us to ultimately settle on the use of Sequelize for storing various users' information:

1. **Abstraction / Portability**: Sequelize abstracts the database system, allowing us to write uniform code that can operate across SQL databases like MySQL, SQLite, PostgreSQL, and MSSQL. The abstraction layer provided by Sequelize will come in handy if we decide to scale our application to a new database in the future.
2. **Connection Pooling**: It efficiently manages a pool of database connections that can be reused, rather than creating a new connection for every database request, which is essential for high-load scenarios.
3. **Modelling Relationships**: It allows complex associations between different data models, facilitating the management of relationships like one-to-one, one-to-many, and many-to-many.
4. **Lazy Loading**: Sequelize supports lazy loading of associated data, which can reduce the initial load on the database, allowing for more scalable data fetching strategies. Leveraging on this, we are able to reduce waiting times when fetching data to display on our frontend.
5. **Validation and Constraints**: Sequelize comes with built-in validators and constraints that can be used to ensure data integrity at the model level.

7.2.2. User Authentication

JSON Web Tokens (JWT)

In our web application, JWT tokens play a crucial role in ensuring secure and efficient data transmission. These tokens are not only compact but also self-contained, which means they are not only transportable through various methods but also do not necessitate server storage. Their compact nature makes them particularly well-suited for our web app, as they enable streamlined communication between different parts of our system without incurring excess data overhead. These key areas of JWT tokens makes them suitable for our web application:

1. **Statelessness**: JWTs are self-contained, carrying all the necessary information about a user. This stateless approach aligns with the RESTful principles our services adhere to, removing the need for server-side session storage and thereby reducing server load and complexity.
2. **Security**: JWTs provide robust security features. With a strong signing algorithm, JWTs assure that the tokens are protected from tampering and forgery.
3. **Flexibility**: The compact structure of JWTs makes them easy to transmit through URL parameters, headers, and HTTP methods, offering flexibility in how we pass secure information between clients and servers. We put the JWT Token in the header, making them exceptionally adaptable to the various services we have defined.
4. **Cross-Domain Access**: JWTs facilitate secure cross-origin resource sharing, which is vital as our application consumes multiple APIs that may not reside on the same domain.
5. **Ease of Access Control**: JWTs can encode user roles and permissions, providing fine-grained access control. The payload of a JWT can include specific claims that denote the user's capabilities, allowing the server to ascertain whether a request should be authorised based on the token's encoded permissions.

Formik

We used form management libraries to offer a simpler and more organised way to handle form state, form submissions and validation. Specifically, we used Formik and Yup to handle form management. Formik simplifies the management of form state

within React components. It efficiently handles values, submission states and validation statuses, providing a robust way to handle form validation, supporting both synchronous and asynchronous validation. It also handles form submissions and submission-related functionalities, such as handling submission events, form resets and submission success or error handling.

Yup

Using Yup allows us to create schemas to define the structure and validation rules for data. These schemas can be used to validate and parse values, and we are able to specify rules for different data types such as strings, numbers, objects, arrays and more. Yup also supports a wide range of validation rules, including required fields, string length constraints and other custom validation logic.

7.2.3. User History

UIW React Heat Map

To visualise the user's history statistics on our home page, we used the UIW React Heat Map component. This component offers a heatmap visualisation feature, providing an efficient way to display data in graphical format. By importing the HeatMap module from 'uiw' and integrating it within various components, the project leverages this heat map to present data on when a user attempted and submitted a code.

Using a pre-built component like UIW React Heat Map instead of building our own heat map from scratch offers several advantages, such as reducing development time and allowing us to focus on other aspects of our project in view of the limited time. As this component is also established and has undergone rigorous testing and bug fixes, the functionality is also more reliable and robust compared to a custom solution. Based on what kind of data we want to display to the users, we can also heavily customise the heat map component.

Circular Progress Bar

The 'react-circular-progressbar' library is used to implement the circular progress indicator. It is visually intuitive and user-friendly, offering a different and more engaging way to represent progress or completion of questions. Users can quickly

interpret progress status by looking at the circular motion. ‘react-circular-progressbar’ provides a ready-to-use and customisable React component that allows us to implement the circular progress indicators easily and efficiently. Its intuitive API makes it straightforward to integrate the circular progress bar into the UI. The library also offers a range of customisable options, such as varying stroke widths, colours and text display, allowing us to create visually appealing and brand-consistent circular progress indicators.

Using this library for our circular progress bar eliminates the need for us to build this UI element from scratch. The library is likely to have undergone testing and maintenance, providing a more reliable and stable solution compared to a custom implementation.

7.3. Retrieving Database of Questions

GraphQL

In pursuit of creating a comprehensive pool of questions for our project, we explored alternative avenues to the provided example questions. Our team's decision was to extract questions directly from LeetCode.

When contemplating the process of retrieving questions from LeetCode for our project, we deliberated on various methods, including utilising GraphQL, downloading questions for direct addition to our database, or automating the process through web scraping since LeetCode does not offer a dedicated API for developers.

We settled on leveraging GraphQL to query questions from LeetCode. The dynamic nature of GraphQL queries ensures adaptability to changes in LeetCode's data structure, future-proofing the system. Additionally, GraphQL's ability to retrieve all necessary data in a single request minimises network data transfer, promoting efficiency compared to resource-intensive alternatives like downloading large datasets or web scraping multiple pages. Using GraphQL, our goal was to collect essential attributes such as the question's title, difficulty level, category, associated topics, content, and solution.

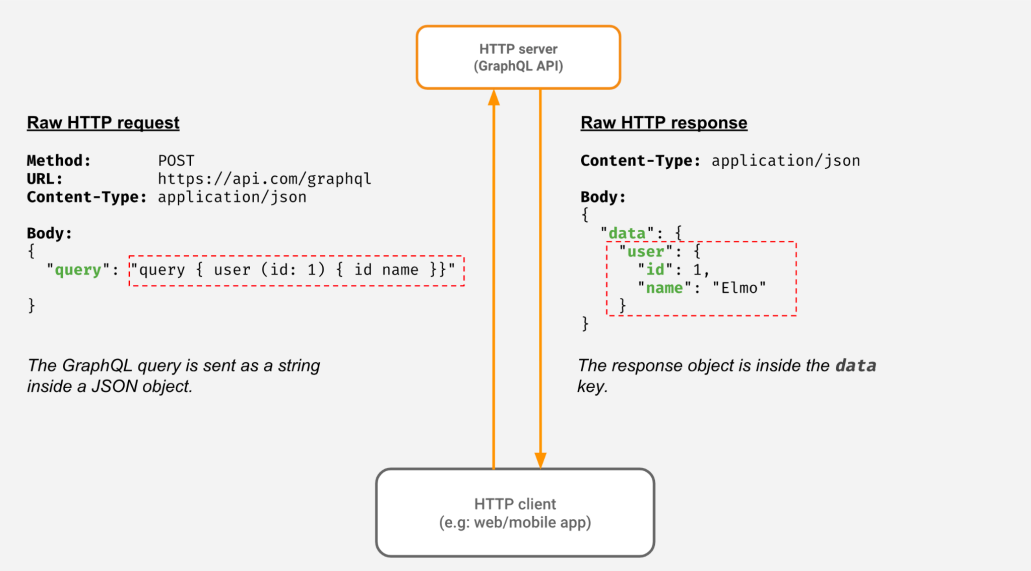

Figure 7.1. GraphQL Query and Store to Web Client

Image obtained from Hasuracon 2023 (https://graphql-engine-cdn.hasura.io/learn-hasura/assets/graphql-react/graphql-on-http.png)

```
_id: ObjectId('652f9ad8660830a5560d7edc')
title: "Next Permutation"
frontendQuestionId: "31"
difficulty: "Medium"
category: "Algorithms"
topics: "Array, Two Pointers"
content: "<p>A <strong>permutation</strong> of an array of integers is an arrang…"
solution: "<div class="toc">
          <ul>
          <li><a href="#video-solution">Video Solution</a…"
isDeleted: true
```

Figure 7.2. MongoDB document structure

As part of our data gathering process, we implemented filters to ensure that we only included questions with both content and a solution. However, we encountered challenges with the formatting of the solutions obtained from LeetCode. These solutions required an iframe to be displayed properly, rendering them incompatible with our intended use.

In response to this constraint, we made the decision to omit the solutions from our database. Nonetheless, we successfully developed three key Python script files that facilitate the integration of questions from LeetCode into our MongoDB database, enable the retrieval of questions from the database, and allow for the deletion of questions as needed. Following which, we created a serverless function by pushing

the Python scripts on Google Cloud Functions to put questions into our MongoDB on a HTTP request.

7.4. Question Service

Question Service primarily relies on MongoDB. As mentioned in Chapter 7.3, we utilised GraphQL, which allows us to query and acquire a rich repository of questions from LeetCode to store in MongoDB Atlas. Our current questions database stores a total of 369 high-quality questions from a variety of topics, categories and difficulty levels.

7.4.1. Storing Questions in MongoDB

We employed the use of MongoDB, a NoSQL database for the purpose of storing, retrieving and managing our questions data in a structured manner. MongoDB's schema-less nature allows a flexible structure for our questions data, while also allowing questions to be represented as a document in JSON-like format. In particular, we utilised MongoDB Atlas, MongoDB's cloud-based database service for this purpose. Some benefits of using MongoDB and MongoDB Atlas are:

1. **Fully Managed Service**: Handles administrative tasks such as provisioning, setup, scaling, updates, backups and monitoring, allowing us to focus on building our application without worrying about the underlying infrastructure.
2. **Scalability**: Atlas provides easy scalability, allowing us to scale our database up and down based on our application's needs.
3. **Ease of Use**: Atlas offers an intuitive and user-friendly interface, simplifying database setup, configuration and management tasks. MongoDB is also easy to work with because of its flexible schema and ability to represent objects in the application code directly into the database.
4. **Flexible Data Model**: MongoDB employs a flexible document data model that allows for the storage of data in JSON-like documents. This flexibility allows us to store various types of data and structures within a single database without requiring a predefined schema.

In MongoDB Atlas, we stored the fields as shown in Figure 7.2. The flag `isDeleted` is introduced to implement the soft delete function.

Mongoose

We opted for Mongoose while searching for an efficient Object-Relational Mapping (ORM) tool to seamlessly connect our Node.js application with MongoDB. This decision was guided by the extensive array of features and capabilities that Mongoose offers, aligning seamlessly with our objectives of structured data management and streamlined database interactions.

At the core of any ORM is the mission to simplify interactions with the underlying database, and Mongoose excels in this aspect. It skillfully abstracts away the intricacies of MongoDB queries and operations, providing a clean and expressive interface for common database tasks. This not only enhances code readability but also accelerates development by eliminating the boilerplate associated with database interactions.

One of Mongoose's standout strengths is its structured and intuitive approach to defining models for MongoDB collections. This methodology brings a sense of clarity to our application code by organising data models coherently and making them easily understandable.

Mongoose transcends the fundamental functions of an ORM by incorporating robust features such as built-in validation and middleware support. These features enable us to enforce data integrity by establishing rules for the structure and content of our data. Additionally, the middleware functions empower the execution of custom logic before or after specific database operations, providing a flexible and extensible architecture for our application.

In summary, our strategic integration of Mongoose as our ORM signifies a commitment to enhancing the efficiency, maintainability, and reliability of our application's interaction with MongoDB. By leveraging the robust features of Mongoose, our aim is to construct a resilient and scalable foundation for managing the diverse set of queries in our database.

7.4.2. Soft Deleting of Questions in MongoDB

The `isDeleted` flag is introduced within the schema of question documents in MongoDB Atlas. The variable acts as a marker to distinguish between active (not deleted) and soft deleted questions.

The `softDeleteQuestion` controller function is responsible for marking a question as soft deleted. Upon receiving a PATCH request by an admin to an API endpoint, the function sets the `isDeleted` flag to true for the specified question ID. Once a question has `isDeleted` set to true, it will not allow users to retrieve the questions in the Questions table anymore.

Certain controller functions, such as `getFirstPaginatedQuestions` which retrieves the first page of questions to display in the Questions table, incorporates the filter to retrieve only questions where the `isDeleted` flag is false. This ensures only active questions are returned in the query results, which is purposeful in allowing users to only view questions that have not been deleted by administrators.

The strategy of soft deletion, marked by the `isDeleted` flag, plays a critical role in preserving question data within the database. This method ensures that while the question is marked inactive, it remains part of the system's historical record. This approach is indispensable for our user history feature, as it allows users to access and revisit questions they have completed since creating their account.

Choosing soft deletion over physical deletion is paramount for maintaining comprehensive user histories. If physical deletion were employed and questions were entirely removed from the database at the discretion of administrators, users would lose access to question details when they navigate into individual questions from their user history table. The soft deletion technique provides the necessary balance, offering the capability to mark questions as inactive while retaining their data and ensuring that users can still access these details for their historical records.

7.4.3. User and Admin Privileges

In the context of Question Service, different user roles possess distinct privileges and access rights. The system distinguishes between regular users and administrators,

each with specific capabilities and permissions through middleware services. The middleware plays a critical role in confirming and validating admin status based on the user's token.

Regular users, upon authentication, have access to only the fundamental functionalities within the system. This includes the ability to view questions stored in the system. For example, users may switch to the Questions tab where they will have access to a complete table of questions and may view the individual question details as they click onto each row. However, unlike admins, they are not able to create new questions, update existing questions or delete any questions from the database.

Administrators possess elevated privileges compared to regular users. These privileges include unrestricted access to view and retrieve all questions in the system, permission to modify or delete any questions stored in the system, as well as the ability to create and add new questions into the database. These functions come in the form of additional buttons in the Questions page.

On the front end, the user interface dynamically adapts based on the user's role. When a user logs in, the system retrieves their role from the server. For regular users, the interface is streamlined to present only the features they are allowed to use, such as viewing questions. Conversely, administrators are presented with an interface that includes additional controls for creating, updating, and deleting questions. This conditional rendering of components not only enhances user experience by decluttering unnecessary elements but also serves as the first layer of defence against unauthorised actions.

On the backend, security is reinforced through a series of middleware services that intercept and verify the authenticity and authorization of the user before any action is processed. Each request to the backend API carries a JWT token in the header, which is decoded to verify the user's identity and role. Middleware services then ascertain whether the user has the appropriate permissions to perform the requested action. For example, when a request to update a question arrives at the question-service, the middleware checks if the sender is authenticated and authorised. If not, the request is rejected, and an error response is sent back to the user.

7.5. Matching Service

The Matching Service oversees both user matching and collaboration spaces. Merging and collaboration as a service is a deliberate choice for efficiency and will be further expanded on below. Socket.IO, CodeMirror, and OneCompiler are selected for user collaboration, coding syntax and highlighting, and code execution respectively.

7.5.1. Matching & Collaboration as a Service

As we navigated the development of our application, a pivotal decision arose concerning the architecture of the Matching Service. The crux of the matter was whether to unite the matching and collaboration capabilities within a single service or to segregate them into two distinct microservices.

Upon various considerations, our team opted for the integration of matching and collaboration capabilities into a unified service. This decision was underpinned by several pivotal factors, all converging toward the overarching goal of refining user experience and optimising system efficiency.

Firstly, a paramount consideration in our architectural choice was the reliance of both matching and collaboration capabilities on Socket.IO for real-time interactions. By amalgamating these features into a singular service, we achieve a centralised hub for handling Socket.IO connections. This not only obviates redundancy but also streamlines the flow of real-time communication, imparting a heightened level of efficiency to the system.

Secondly, the intrinsic nature of our application dictates that collaboration cannot transpire until a suitable match has been identified. Acknowledging the interdependence of these features, amalgamating them into a cohesive service simplifies the logic governing their interaction. This ensures a seamless transition for users, progressing seamlessly from matching to collaboration.

Lastly, the prospect of segregating matching and collaboration capabilities into distinct microservices presented the potential risk of a less user-friendly experience. In the event of downtime in the collaborative service, users might successfully find matches but face impediments progressing to the collaborative phase. Integration

prioritises a user-centric design, mitigating disruptions and enhancing overall user satisfaction.

In summation, the choice to integrate matching and collaboration capabilities into a unified service was steered by considerations of efficiency, simplicity, and, paramountly, the user experience. The seamless transition from matching to collaboration, coupled with the centralised handling of real-time interactions, positions our application to deliver a more reliable and user-friendly service.

7.5.2. Socket.IO Instead of Messaging Queue Technology

The decision to leverage Socket.IO instead of messaging queue technology for our Matching Service stems from strategic considerations that align closely with our application's unique requirements. Integrating matching and collaboration services into a unified entity necessitates a streamlined architecture, and Socket.IO serves as an all-encompassing solution for both functions, simplifying the overall system design. Opting for a messaging queue for matching and a distinct mechanism for collaboration could introduce unnecessary complexity, making the architecture more challenging to manage and maintain.

Socket.IO's use of the WebSocket protocol inherently reduces latency compared to traditional HTTP-based solutions commonly employed with messaging queues. This low-latency communication is especially advantageous during the matching phase, where timely updates are crucial. The lightweight nature of Socket.IO's protocol minimises communication overhead, contributing to a more responsive and efficient user experience throughout the matching process.

Notably, Socket.IO provides a straightforward and easily implementable solution for real-time communication. Its simplicity expedites development and reduces the learning curve for our development team. By choosing Socket.IO over a messaging queue for matching, we align with our goal of achieving efficiency in development without compromising the reliability and performance required for user matching.

In conclusion, our decision to utilise Socket.IO for user matching in the Matching Service is driven by real-time capabilities, suitability for a unified service architecture,

low latency, and ease of implementation. Collectively, these factors contribute to a more responsive, integrated, and user-friendly experience throughout the matching and collaboration phases of our application.

7.5.3. User Matching

Matching Peers with Socket.IO

Socket.IO proves to be an ideal choice for our project, offering a robust foundation for both real-time coding collaboration and chat messaging. Its suitability for real-time communication is evident in its ability to provide instant data exchange between the server and clients, eliminating the need for polling. This instant, bidirectional data flow is pivotal for ensuring an engaging and responsive experience, which is essential for both coding collaboration and chat interactions.

Moreover, Socket.IO's underlying technology, WebSockets, establishes and maintains persistent connections, significantly reducing network and server load compared to traditional HTTP requests. This level of efficiency is paramount in a chat application where users anticipate quick and seamless message delivery, ensuring an optimised performance. By choosing Socket.IO, we've laid the groundwork for real-time, efficient, and reliable communication that underpins both our coding collaboration and chat messaging features, offering a seamless and interactive experience for our users.

7.5.4. User Collaboration

This section described the chosen libraries and its functionalities used for user collaboration in PeerPrep.

7.5.4.1. Chat Feature

Socket.IO

Our chat feature seamlessly integrates with Socket.IO, fostering not just efficient but also interactive communication. To enhance the conversational experience, we've implemented real-time user activity updates and typing indicators, enriching the dynamics of our chat functionality.

Instant notifications are broadcasted to all users whenever someone connects or disconnects from the chat room. This real-time user activity update feature promotes

transparency, keeping everyone informed about who is currently present in the chat. This fosters an environment conducive to active engagement within the conversation.

To further elevate the interactive nature of the chat, we've incorporated typing indicators. When a user begins typing a message, a corresponding "typing" status is displayed, signalling to the other party that a response is imminent. This dynamic feature adds responsiveness to the chat, making interactions more engaging and natural.

By seamlessly integrating these features with Socket.io, our chat functionality not only ensures efficient communication but also actively encourages user engagement. This approach creates an environment where users can easily connect, stay informed about online presence, and enjoy dynamic and interactive conversations.

7.5.4.2. Coding Space Syntax and Suggestion

CodeMirror

CodeMirror In our evaluation of text editors for our coding environment, we considered Codemirror, Monaco (used in Visual Studio Code), and Ace (Ajax.org Cloud9 Editor). While each editor has its merits, we chose Codemirror for several compelling reasons.

Codemirror offers exceptional customizability, allowing us to tailor it to our specific project requirements and providing robust support for multiple programming languages. Monaco and Ace offer some level of customizability, but neither reaches the same level of flexibility as CodeMirror. CodeMirror's lightweight and modular design minimises overhead, enabling our applications to load swiftly and operate efficiently. In contrast, Monaco, as the foundation for Visual Studio Code, carries a heftier resource footprint, making it less suitable for resource-constrained environments. Ace falls somewhere in between in terms of resource usage.

CodeMirror offers a comprehensive feature set that extends beyond the basics, including syntax highlighting, error detection, and code folding. These features streamline the coding process and enhance productivity. Monaco, backed by Visual Studio Code, boasts an impressive feature set as well, suitable for professional development environments. Ace, while feature-rich, may not match the extensive feature set of Monaco or CodeMirror.

These technical choices of Socket.IO and CodeMirror create an efficient, customizable, secure, and adaptable environment for collaborative coding.

7.5.5. Code Execution

OneCompiler

In our quest to select the ideal code execution platform for our project, we thoroughly explored various options and narrowed down our choices to two finalists: Judge0 and OneCompiler. While Judge0 offered an enticing proposition as a free and open-source online code editor with support for a rich set of programming languages, a critical limitation emerged during our evaluation. It became apparent that Judge0 had primarily been tested on Linux and MacOS, potentially leading to compatibility issues on Windows. Given our commitment to ensuring accessibility across various operating systems, including Windows, we made the strategic decision to exclude Judge0 from our consideration.

Subsequently, we opted for OneCompiler as our preferred code execution platform. Despite certain limitations, such as a more restricted number of free runs per day on its free plan, OneCompiler demonstrated superior cross-platform compatibility, aligning seamlessly with our goal of ensuring our application's versatility. Moreover, we recognized the scalability potential of our project with OneCompiler's premium plans, providing room for expansion if the need arises. This decision was strategic, aimed at enhancing the accessibility and flexibility of our project while offering a robust code execution environment for our users.

OneCompiler, our chosen platform, serves as a versatile API provider with comprehensive code execution capabilities. Notably, it supports over 60 programming languages, providing users with a wide spectrum of options for their coding endeavours. The integration of OneCompiler's capabilities into our application is facilitated through its API, which is conveniently hosted on RapidAPI, the world's largest API hub. RapidAPI plays a pivotal role in simplifying the integration and utilisation of APIs from different providers for developers, ensuring that our application can effortlessly harness the power of OneCompiler's code execution capabilities.

This strategic choice, combining OneCompiler's robust code execution capabilities, extensive language support, and integration via RapidAPI, underscores our commitment to providing users with a seamless and efficient coding experience. By making this decision, we aim to offer an inclusive and versatile platform that caters to the diverse needs of our user base.

RapidAPI

The OneCompiler's code execution API is hosted on RapidAPI. To invoke this API from the PeerPrep application, specific API endpoint details, including the 'X-RapidAPI-Key' and 'X-RapidAPI-Host' associated with the current subscription, are requisite.

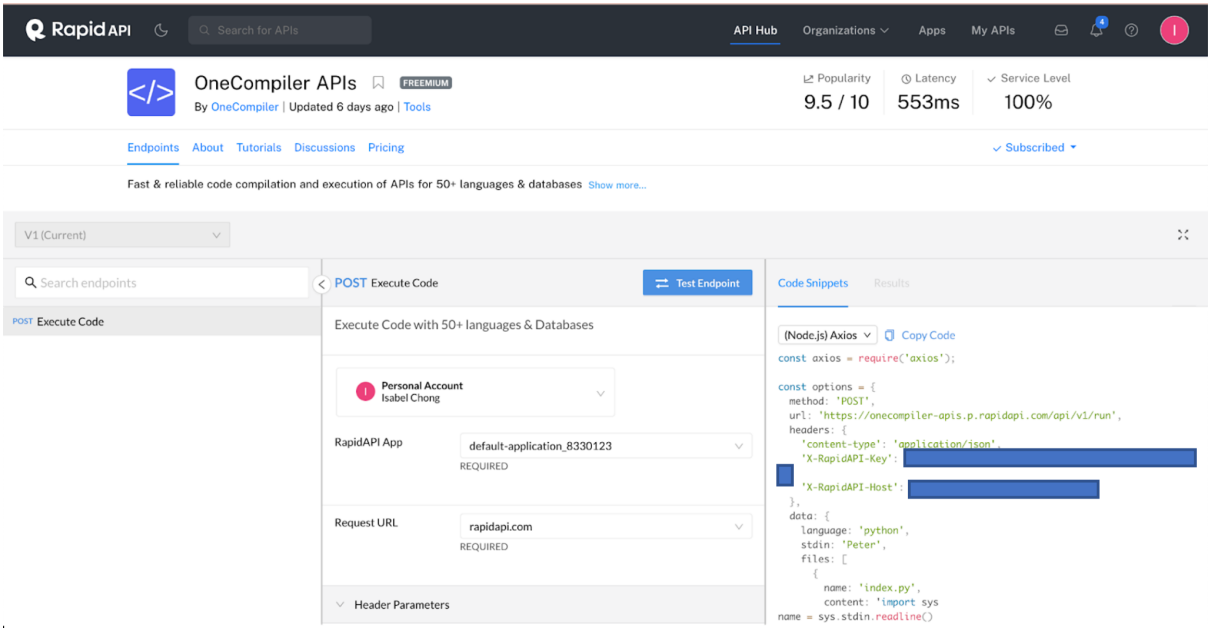

Figure 7.3. RapidAPI Interface showing OneCompiler's Code Execution API Details

We follow the given details given by RapidAPI to execute the code as shown in the figure below.

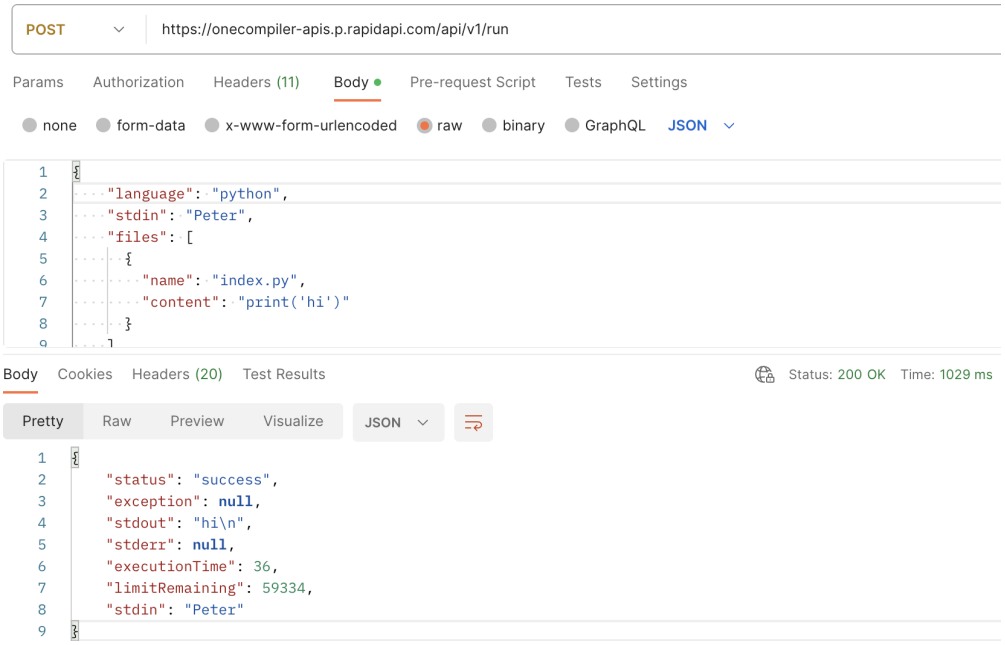

Figure 7.4. Postman Interface showing OneCompiler's Code Execution Output

For the sake of maintaining uniformity in the naming of our API routes, the Matching Service initiates this call for the 'code/run' path. We remove some inputs which are not necessary in this iteration of PeerPrep such as `stdin`, and the resulting output is displayed in the code space page under the 'Console,' exemplified in Figure 6.7.

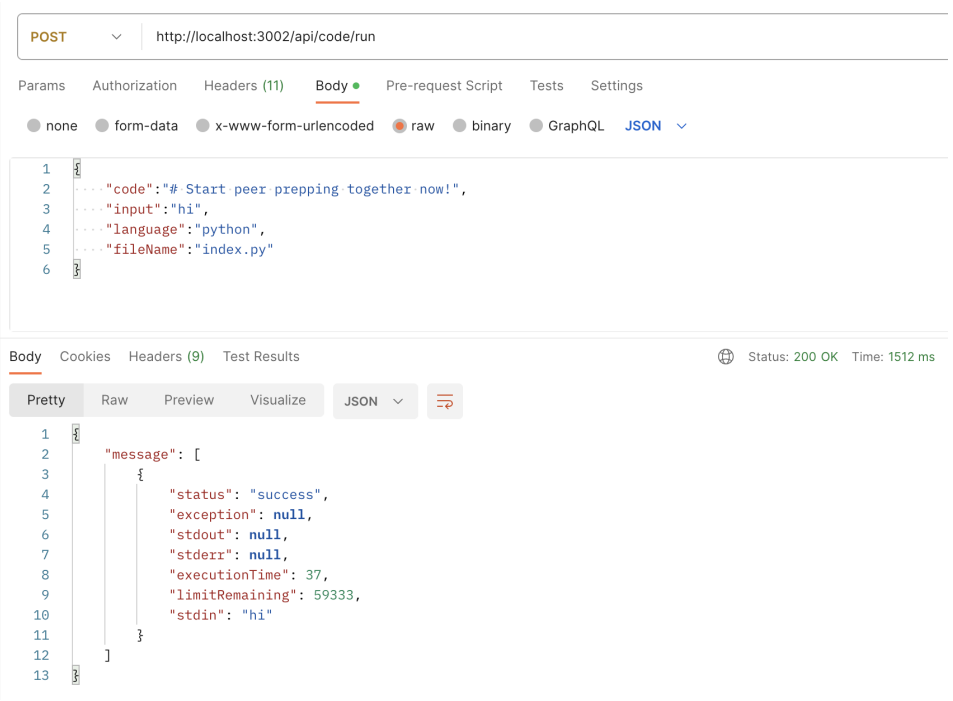

Figure 7.5. Postman Interface showing PeerPrep's Code Execution Output

RapidAPI provides the capability to modify the subscription directly. Under the current free plan, we can execute up to 50 requests daily, totaling approximately 1000 requests monthly. In consideration of scaling the application, opting for a paid subscription becomes viable, allowing for the execution of up to 100,000 and beyond requests per month.

Objects	Basic $0.00 / mo Change Plan	Pro $5.00 / mo Change Plan	Ultra $10.00 / mo Change Plan	Mega $25.00 / mo Change Plan	Deprecated Plan Deprecated $0.00 / mo Currently Subscribed Manage And View Usage
Requests	**1,000 / month** Hard Limit	**10,000 / month** Hard Limit	**30,000 / month** + US$0.001 each other	**100,000 / month** + US$0.001 each other	**50 / day** Hard Limit
Features					
Code Execution 50+ languages	✓	✓	✓	✓	✓
Rate Limit	1000 requests per hour	50 requests per second	100 requests per second	500 requests per second	10 requests per second

Figure 7.6. OneCompiler's Subscription Pricing

Chapter 8. API Documentation

This chapter serves as a detailed reference for all API routes implemented in the application. Each API route plays a crucial role in facilitating communication between different components, enabling seamless data exchange and interaction within the system. Understanding the structure, functionality, and usage of these routes is essential for developers, third-party integrators, and any stakeholders engaging with the application programmatically.

8.1. User Service

8.1.1. User Authentication

User authentication in User-Service is implemented using JWT Tokens. In our use case, we present these tokens to the server in the HTTP header field under “x-access-token”. The following are the API routes for user-authentication.

HTTP Method	API Route	Purpose	Headers	Parameters (JSON)	Require Admin?
POST	/api/auth/ signup	User sign up	-	username, email, password, roles (optional)	No
POST	/api/auth/ signin	User sign in	-	username, password	No
DELETE	/api/auth/ removeuser	Remove a user	x-access-token	None	No
PATCH	/api/auth/ updateprofile	Update user profile	x-access-token	username, email	No
PATCH	/api/auth/ updatepassword	Update user password	x-access-token	currentPassword, newPassword	No
GET	/api/auth/ getuser	Get user profile	x-access-token	None	No
GET	/api/auth/ verifytoken	Verify JWT token	x-access-token	None	No
GET	/api/auth/ verifyadmin	Verify if user is an admin	x-access-token	None	Yes

Figure 8.1. API Endpoints for User Authentication

8.1.2. User History

The following are the API routes for user service.

HTTP Method	API Route	Purpose	Headers	Parameters (JSON)	Require Admin?
POST	/api/hist/save	Save user history	x-access-token	questionId, difficulty, attempt	No
POST	/api/hist/ customsave	Save custom user history	x-access-token	questionId, difficulty, attempt, date	No
GET	/api/hist/ attempts	Get all attempted dates from history	x-access-token	None	No
GET	/api/hist/ get	Get all questions from user history	x-access-token	None	No

Figure 8.2. API Endpoints for User History

8.2. Question Service

The following are the API routes for question service.

HTTP Method	API Route	Purpose	Headers	Parameters (JSON)	Require Admin?
GET	/api/questions/	Retrieve all questions	x-access-token	None	No
GET	/api/questions/pagination/ first	Retrieve first page of paginated questions	x-access-token	None	No
GET	/api/questions/pagination/ remaining	Retrieve remaining paginated questions	x-access-token	None	No
GET	/api /questions/:id	Retrieve a question by ID	x-access-token	None	No
POST	/api/questions	Create a new question	x-access-token	title, frontendQuestionId, difficulty, content, category, topics	No

PUT	/api /questions/:id	Update a question	x-access-token	title, frontendQuestionId, difficulty, content, category, topics	No
PATCH	/api /questions/:id	Soft delete a question	x-access-token	None	No
DELETE	/api/questions/:id	Delete a question	x-access-token	None	No
GET	/api/questions/matched	Get a random question via filters	x-access-token	difficulty, topics	No
POST	/api/questions/questionbyid	Retrieve questions by IDs	x-access-token	ids (array)	No
GET	/api/questions/total	Get total number of questions per difficulty	None	None	No

Figure 8.3. API Endpoints for Question Service

8.3. Matching Service

The following are the API routes for matching service.

HTTP Method	API Route	Purpose	Headers	Parameters (JSON)	Require Admin?
GET	/api/room /:roomId	Retrieve room information by ID	-	-	No
POST	/api/code/run	Run user submitted code	x-access-token	code, input, language, fileName	No

Figure 8.4. API Endpoints for Matching Service

Chapter 9. Conclusion

9.1. Reflections

Developing the PeerPrep platform using MERN and other technologies has been a journey replete with learning experiences and collaborative efforts. Each phase of development, from ideation to the final product, revealed numerous obstacles and successes. Exploring the different technologies not only allowed us to leverage on their different flexibilities and scalabilities, but also presented a learning curve that enhanced our understanding of full-stack development.

The iterative nature of our development process enabled us to adapt and evolve, accommodating user needs and industry best practices. Through collaboration within our team and brainstorming solutions, we were able to overcome hurdles, implementing efficient solutions and refine our codebase. The development journey was not just about coding but also understanding the intricacies of each technology used and integrating them harmoniously to create a seamless user experience.

Reflecting on our collaborative journey with PeerPrep, we have distilled several insightful points that encapsulate our collective experiences and growth as collated below.

1. **Collaborative Workflow:** Working in a team, especially in subgroups, has taught us the significance of clear and effective communication. This becomes increasingly vital as certain tasks overlap, necessitating meticulous coordination to ensure seamless integration and a coherent end product. Through weekly SCRUM meetings and constant communication in our group chat, we ensured a unified understanding of each subgroup's progress, aiding in the resolution of any interdependencies and enhancing our overall project efficiency.
2. **Problem-Solving Skills:** Our journey with PeerPrep was marked by encounters with various challenges. One instance we faced is reverse engineering our assignment submissions due to the course submission requirements. This was because our submission tagging method did not meet the course requirements. These instances served as practical lessons, highlighting our adaptability and resilience in the face of adversity. We were able to sit down as a team to think about how to solve this problem promptly and effectively. As we document

these challenges and outline the strategies employed for resolution, we showcased our capacity to learn from mistakes and continuously refine our approaches. In doing so, we also train our problem-solving skills, an essential skill that we have to develop in the field of software engineering.

3. **Time Management and Adaptability:** Within our team, each subgroup shouldered varied responsibilities and contributed to different features, which resulted in a multitude of tasks essential for PeerPrep's development. Balancing these tasks necessitated adept time management and a keen sense of adaptability. Given the scope and diversity of features to implement, unforeseen changes and evolving requirements were an inherent part of our project landscape. In response, our team embraced these challenges by flexibly adjusting our strategies, redistributing tasks and reorganising priorities, showcasing our agility in handling complex and ever-evolving project demands. This adaptability not only allowed us to navigate unexpected changes but also underscored our capability to manage time efficiently amidst dynamic project needs.

9.2. Challenges

In the course of our project, the initial intention was to divide the group into two sub-groups, each responsible for specific features of an application. However, as we delved deeper into the project, we quickly realised that complete segregation of work was not feasible, and the mingling and sharing of tasks became inevitable. Some challenges we faced in splitting work into different subgroups include the following.

1. **Interconnected Dependencies:** The features of the application turned out to be more interconnected than initially anticipated. Attempting to separate tasks into isolated subgroups led to challenges where one subgroup's work often depended on the progress of the other. This interdependence posed hurdles and we quickly realised that managing these dependencies required a closer collaboration, transparent communication and an agile approach to adapt and adjust to the changes.
2. **Collaborative Synergy:** We discovered that some features required collaborative effort and sharing of knowledge to ensure that the final product met our quality and functionality standards. There also exist certain differences

in the level of exposure to various technologies and tools among individuals, with some having more familiarity and experience than the others.

3. **Deployment:** Most of us were not familiar with containerised deployment and either relied on Heroku or AWS EC2 where the entire codebase is manually deployed on a single virtual machine. As such, it was interesting to learn how to add listeners in google cloud build to assist not only in building from our main branch, but also in testing features after deployment in other branches.

9.3. Takeaways

Amidst these challenges, we gleaned valuable lessons that have shaped our approach to working on full-stack development projects in a team. The collaborative nature of full-stack development highlighted the necessity of cohesive teamwork and cross-functional communication. Understanding the interconnectedness of different layers and technologies also emphasised the need for a holistic view of the project. Some valuable insights we took away from PeerPrep are as follows:

1. **Enhanced Communication:** By allowing team members to work closely and share ideas, our project benefited from improved communication, leading to more effective problem-solving. This open dialogue allowed for transparency and encouraged iterative feedback loops, resulting in more efficient and innovative solutions. Our team's agile and adaptive approach, facilitated by robust communication, significantly boosted our project's efficiency and productivity.
2. **Flexibility:** By embracing mingling and information sharing, we cultivated a more adaptable approach to resource allocation. This flexibility empowered us to adjust resources in response to the ever-evolving needs of the project. As we openly exchanged information and ideas, we were also able to effectively reallocate resources, ensuring a more agile and responsive project environment.
3. **Cross-Functional Learning:** By allowing everyone to work on diverse tasks, each team member gained a broader understanding of the project, enriching skill sets and knowledge base. The exposure to various facets of the project allowed for a more holistic comprehension, empowering us with versatile skills and a deeper understanding of the project's intricacies.

4. **Continuous Collaboration:** Mingling and sharing of work promoted continuous collaboration and brainstorming sessions, which contributed to the overall quality of PeerPrep. This consistent exchange of ideas and collective effort created a dynamic environment where collaborative input significantly enriched our project's outcome.

9.4. Future Recommendations

This subchapter includes some of the recommendations that can be done to improve our iteration of PeerPrep.

1. **Checking Correctness of Code Submissions:** We encountered limitations while attempting to retrieve solutions from LeetCode to display due to file types that are incompatible or invalid for display outside of the platform. These specific file types can only be viewed or accessed on the original LeetCode website, hindering our ability to showcase or view the solutions on our frontend. In a future iteration, given we had more time to work with, we could create our own repository of questions and solutions to check users' solutions during code execution and submission.
2. **Incorporating Generative AI:** To further augment the preparation journey, integrating generative AI (Artificial Intelligence) technologies such as ChatGPT or Copilot as a future extension would be highly beneficial, offering user interactive aid. They would provide users with instant coding assistance, detailed explanations and help users develop strategies to tackle the problems.
3. **Incorporating Scalability in deployment:** The current infrastructure of PeerPrep which is built on Docker, Google Cloud Builds and running on Google Cloud Run sets a solid foundation for scalability. To capitalise on this, Kubernetes for container orchestration and Nginx for API Gateway management and monitoring would bolster traffic management and enhance PeerPrep's ability to scale horizontally.
4. **Testability:** While we had postman test cases, we lacked unit test cases on our CI and code repository, we should have implemented test cases with each feature created instead of solely focusing on feature development.
5. **Optimising Docker Compose Functionality:** To enhance our development environment, we should consider deploying MongoDB within a container for

local development, rather than relying on MongoDB Atlas. This approach ensures individual developers have their own isolated local databases, eliminating the need to share a development cloud database. Furthermore, we can leverage Docker Compose features by mounting volumes onto containers. This practice preserves data generated within each container, facilitating the persistence of changes in packages and source code during development. With this setup, we can achieve hot reloading of containers, enabling us to promptly observe and validate changes without the necessity of restarting the container after each modification. This strategy enhances the efficiency and autonomy of our local development processes.

6. **Security Enhancement:** In the spirit of continuous development, enhancing the security framework of the application, implementing advanced encryption for data at rest and transit would provide an additional layer of protection against unwanted attacks. User Service making use of Multi-Factor Authentication (MFA) would also significantly enhance account security, ensuring that just user credentials alone would not be sufficient to gain access to the service.
7. **User Experience (UX) Enhancements for Questions Page:** For an elevated user experience, particularly within the Question Service domain, the implementation of a refined search feature such as a filter would greatly enhance how users interact with the questions in the system. An advanced search algorithm would facilitate fast and accurate retrieval of the questions.

Given PeerPrep's academic context and time constraints associated with such undertakings, the recommendations are envisaged as a roadmap for future development enhancements. They aim to continuously evolve and enhance PeerPrep to exceed user expectations and industry standards.

Glossary

Term	Explanation
API	An **Application Programming Interface (API)** defines a set of rules and tools for building software applications, allowing them to communicate with each other.
AWS	**Amazon Web Services (AWS)** is a widely used cloud computing platform that offers a variety of services, including computing power, storage, and databases, among others.
CI/CD	**Continuous Integration/Continuous Deployment (CI/CD)** is a set of practices and tools aimed at automating the integration and deployment of code changes, ensuring frequent and reliable releases.
DevOps	**DevOps** is a set of practices that combines software development (Dev) and IT operations (Ops), aiming to shorten the development lifecycle and deliver high-quality software continuously.
GCP	**Google Cloud Platform (GCP)** is a public cloud vendor providing a suite of cloud computing services, including computing, storage, databases, machine learning, etc.
Heroku	**Heroku** is a platform as a service (PaaS) that enables developers to build, run, and operate applications entirely in the cloud.
HTTP	**Hypertext Transfer Protocol (HTTP)** is a fundamental protocol used for communication between web clients (such as browsers) and servers.
HTTPS	**Hypertext Transfer Protocol Secure (HTTPS)** is a secure version of HTTP, encrypting data exchanged between web clients and servers to ensure the confidentiality and integrity of the communication.
JWT	**JSON Web Token (JWT)** is a compact, URL-safe means of representing claims to be transferred between two parties.
MERN	**MERN** is an acronym for a full-stack development stack, including MongoDB (database), Express.js (backend framework), React (frontend library), and Node.js (JavaScript runtime).
OS	**Operating System (OS)** is software that manages computer hardware and provides services for computer programs, facilitating communication between software and hardware components.
RESTful	**RESTful** refers to Representational State Transfer, a set of architectural principles for designing networked applications. It emphasises stateless communication and resource-based interactions.
SQL	**Structured Query Language (SQL)** is a standardised programming language used for managing and manipulating relational databases.

UI	**User Interface (UI)** refers to the graphical and interactive elements through which users interact with software applications. It includes components like buttons, menus, and visual elements.
YAML	**YAML** is a human-readable data serialisation format commonly used for configuration files. It stands for "YAML Ain't Markup Language".

Appendix A: PeerPrep Features

Below are the must-have and nice-to-have features snipped from the CS3219 project instructions document.

The following are the must-have features (labelled Mx) of PeerPrep.

- M1: User Service – responsible for user profile management.
- M2: Matching Service – responsible for matching users based on some criteria (e.g., difficulty level of questions, topics, proficiency level of the users, etc.) This service can potentially be developed by offering multiple matching criteria.
- M3: Question service – responsible for maintaining a question repository indexed by difficulty level (and any other indexing criteria – e.g., specific topics).
- M4: Collaboration service – provides the mechanism for real-time collaboration (e.g., concurrent code editing) between the authenticated and matched users in the collaborative space.
- M5: Basic UI for user interaction – to access the app that you develop.
- M6: 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).

The following are nice-to-have features (labelled Nx) of PeerPrep.
Note that the list is not exhaustive. You are welcome to conceptualise and implement other nice-to-have features/functionalities. Discuss your ideas with your mentor and get his approval before implementing them.

- N1: Communication: Implement a mechanism to facilitate communication among the participants in the collaborative space (other than the shared workspace) e.g., text-based chat service and/or video (+voice) calling service.
- N2: History: Maintain a record of the questions attempted by the user e.g., maintain a list of questions attempted along with the date-time of attempt, the attempt itself and/or suggested solutions.
- N3: Code execution: Implement a mechanism to execute attempted solution/code in a sandboxed environment, and retrieve+present the results in the collaborative workspace.

- N4: Enhance question service to enable managing questions, for example, tagging (by topic, popularity, etc.,), retrieving questions on the fly during a session initiation.
- N5: Enhance collaboration service by providing an improved code editor with code formatting, syntax highlighting for one language, syntax highlighting for multiple languages.
- N6: Enhance collaboration service by providing code translation from one language to another (e.g., JS to Python) depending on the preference of the users. (I.e., user 1 prefers python, user 2 prefers JS, provide a mechanism to translate code to the preferred language of each of the users)
- N7: Incorporate generative AI to assist during the preparation. For example, users can seek help from ChatGPT, Copilot (or such other services) to seek explanation of code written by the other participant. Users can seek help from gen AI to solve the puzzle by providing prompts from within the interface.
- N8: Extensive (and automated) unit, integration, and system testing using CI. Teams can also use various test frameworks or demonstrate effective usage of CI/CD in the project.
- N9: Deployment of the app on the production system (AWS/GCP cloud platform).
- 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.
- N11: 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 (https://kubernetes.GitHub.io/ingress-nginx/).
- N12: 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 (https://kubernetes.io/docs/concepts/services-networking/dns-podservice/).

Appendix B: Non-Functional Requirements (NFR) Testing

B.1: Usability Testing

NF1	Usability	Priority
	NF1.1 The UI should have a consistent colour scheme.	M
	NF1.2 The application should be accessible via a public URL.	H
	NF1.3 The application should have a responsive UI.	H
	NF1.4 The application should support a minimum resolution of 1024 x 768 px.	H

NF1.1 (Medium): The UI should have a consistent colour scheme

The screenshot below shows the Figma design which follows a consistent colour scheme of purple on grey.

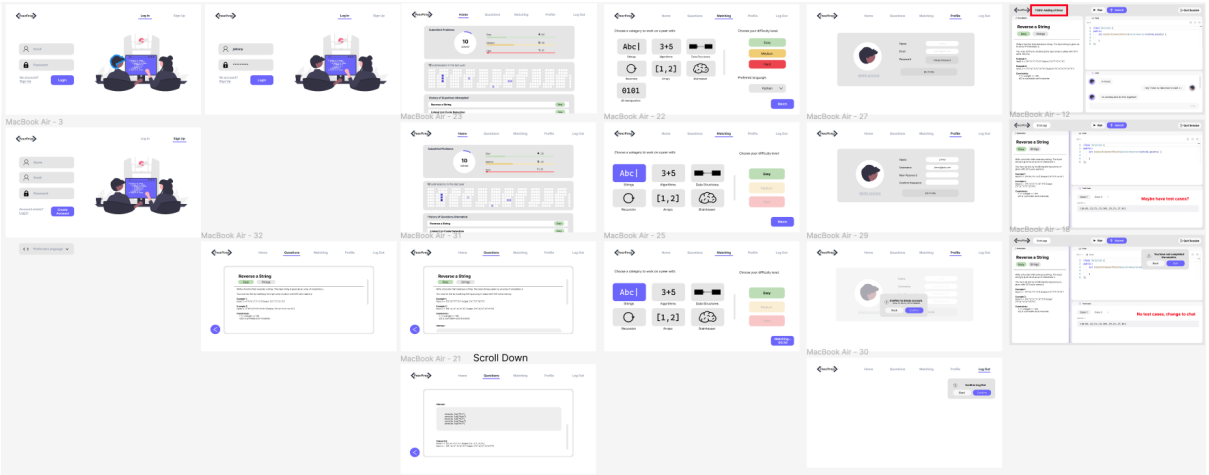

Figure B.1: Colour Scheme of PeerPrep Extracted From Figma

NF1.2 (High): The application should be accessible via a public URL

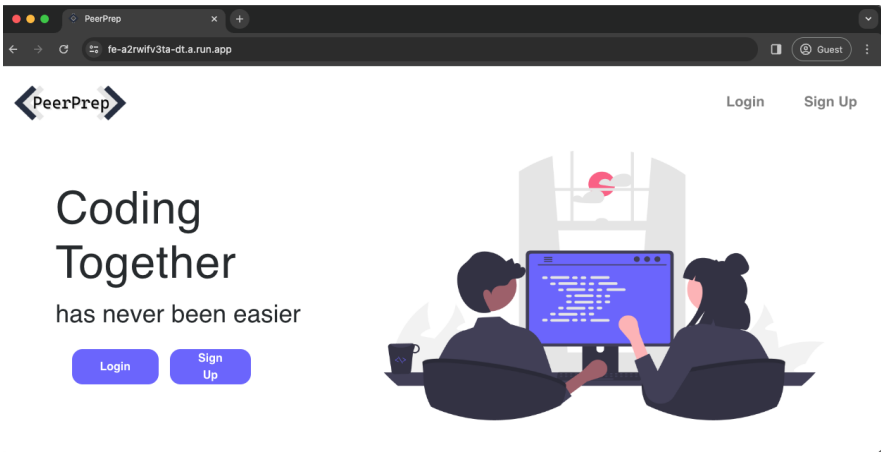

Figure B.2: PeerPrep Accessed From a Publicly Accessible Website

NF1.3 (High): The application should have responsive UI

The screenshot below shows how the application's UI is responsive to the different screen sizes the user is using.

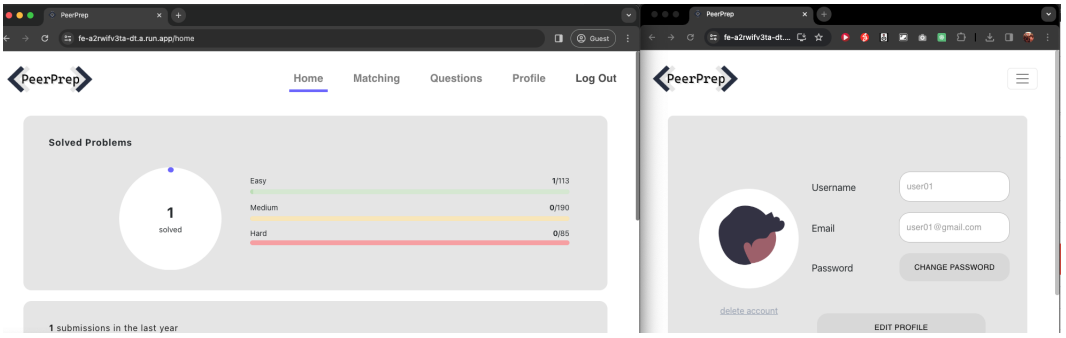

Figure B.3: PeerPrep Accessed By Browser of Different Sizes

NF1.4 (High): The application should support a minimum resolution of 1024 x 768 px

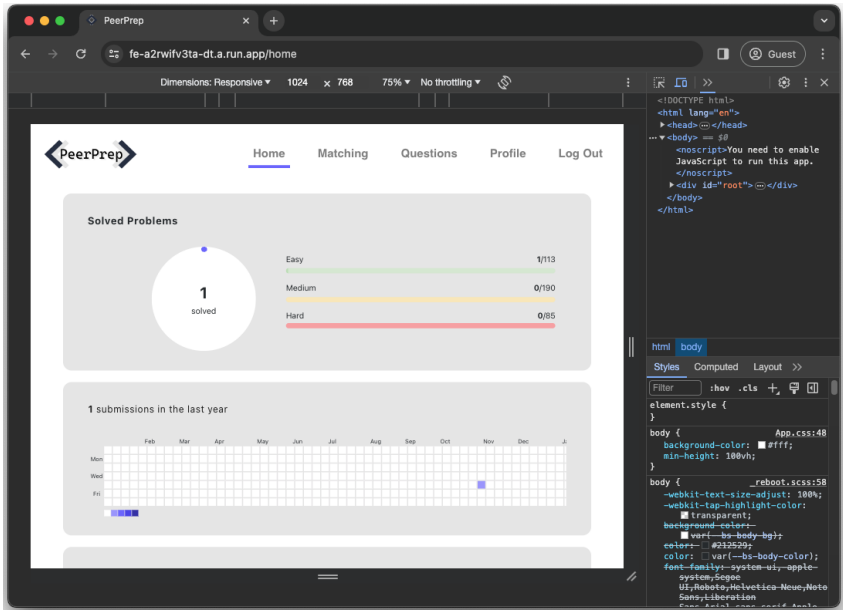

Figure B.4: PeerPrep Accessed by Browser Simulated With 1024x768 Screen

B.2: Performance Testing

NF2	Performance	Priority
	NF2.1 The application should be able to handle at least 2 people using it at once.	H
	NF2.2 The application should take less than 5 seconds to load in the questions for the match.	H
	NF2.3 The application should take no longer than 5 seconds to log in	H
	NF2.4 The system should respond to each request within 5 seconds.	M

NF2.1 (High): The application should be able to handle at least 2 people using it at once.

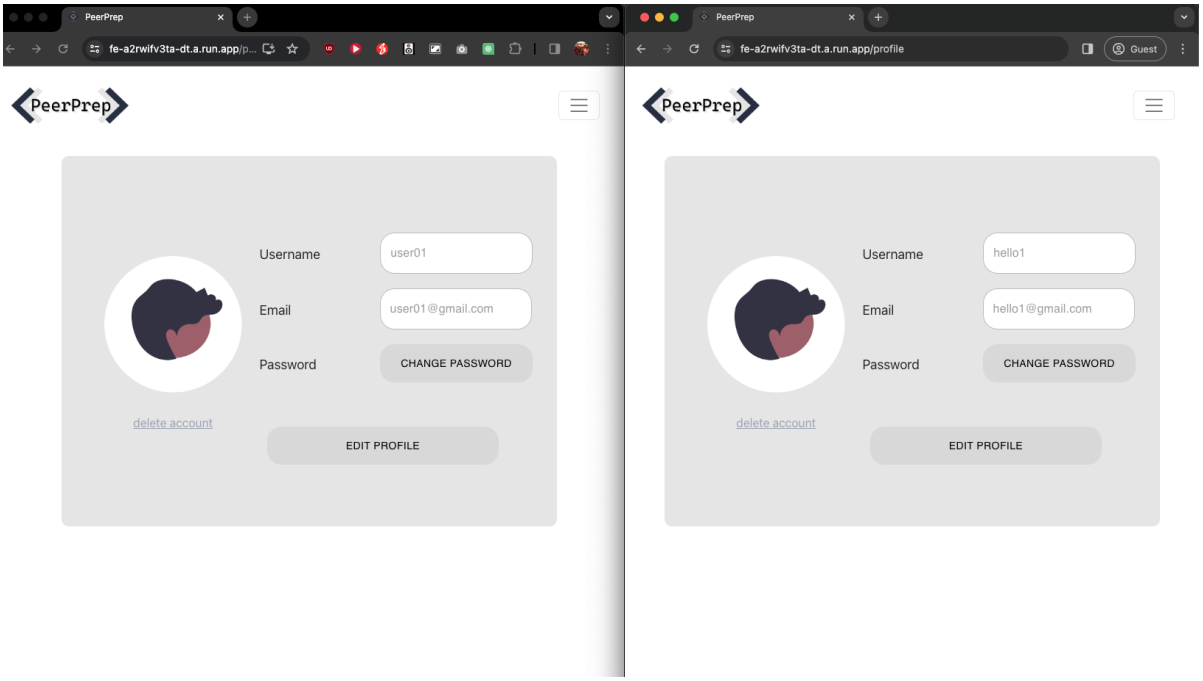

Figure B.5: PeerPrep Showing Two Different Users Logged In At Once

NF2.2 (High): The application should take less than 5 seconds to load in the questions for the match.

The screenshot below shows that the question loading for the two users took 602ms and 470ms respectively.

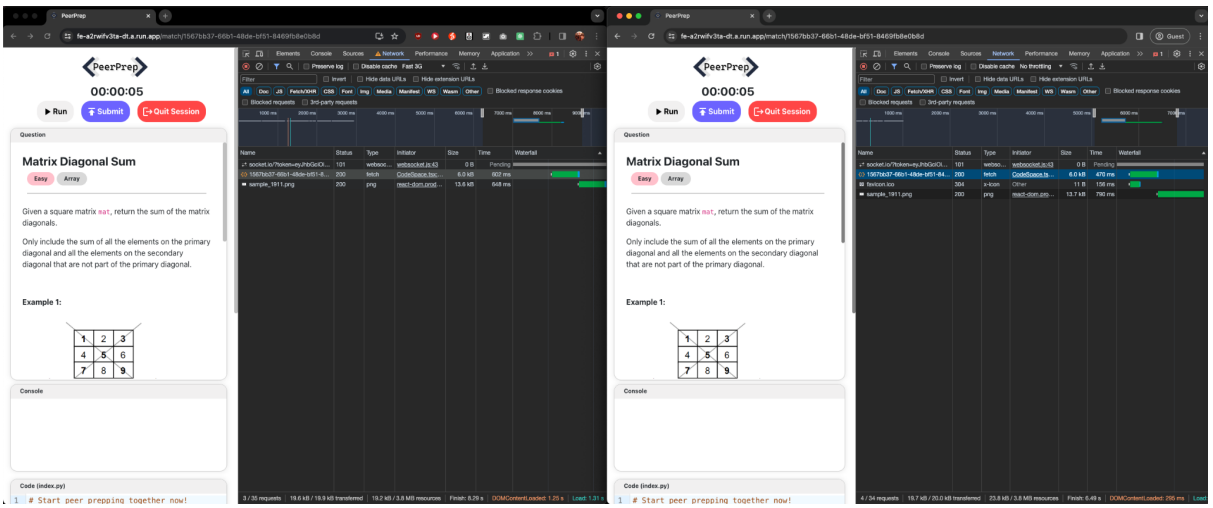

Figure B.6: Web Details Showing Question Loading to Collaboration Space

NF2.3 (High): The application should take no longer than 5 seconds to log in

The screenshot below shows that logging in took 1.05 seconds.

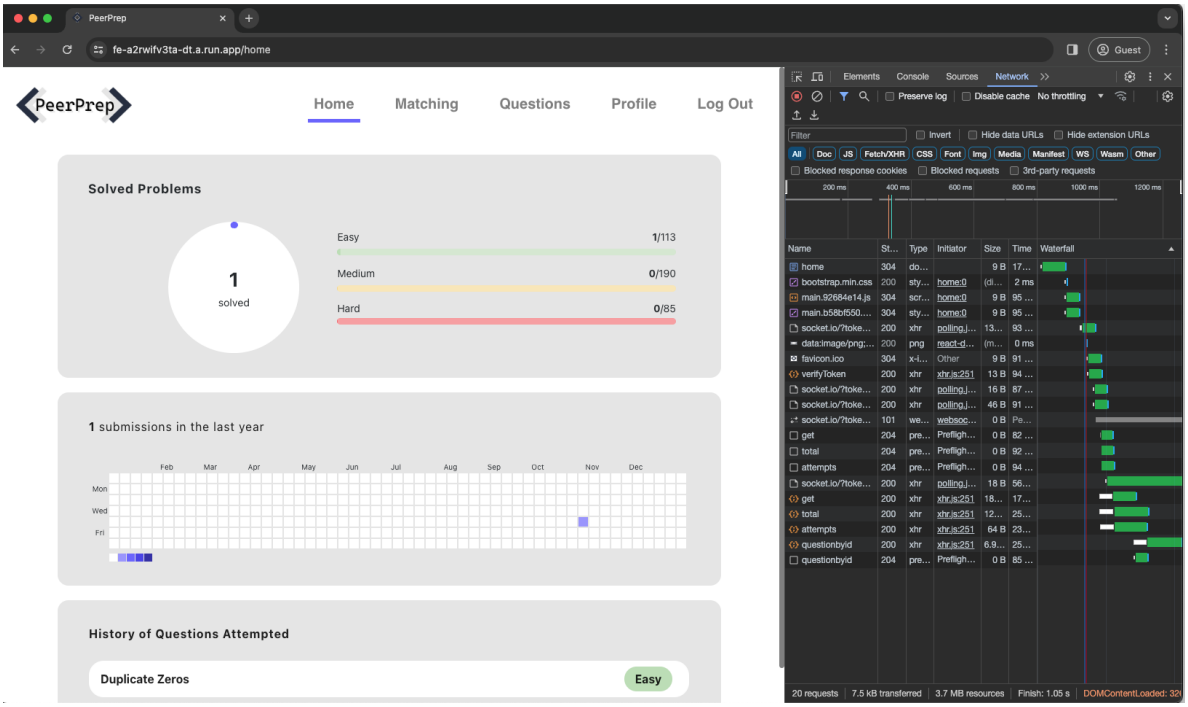

Figure B.7: Web Details Showing User Logging In

NF2.4 (Medium): The system should respond to each request within 5 seconds.

The screenshot below shows that question loading for the “Questions” page took 2.47 seconds.

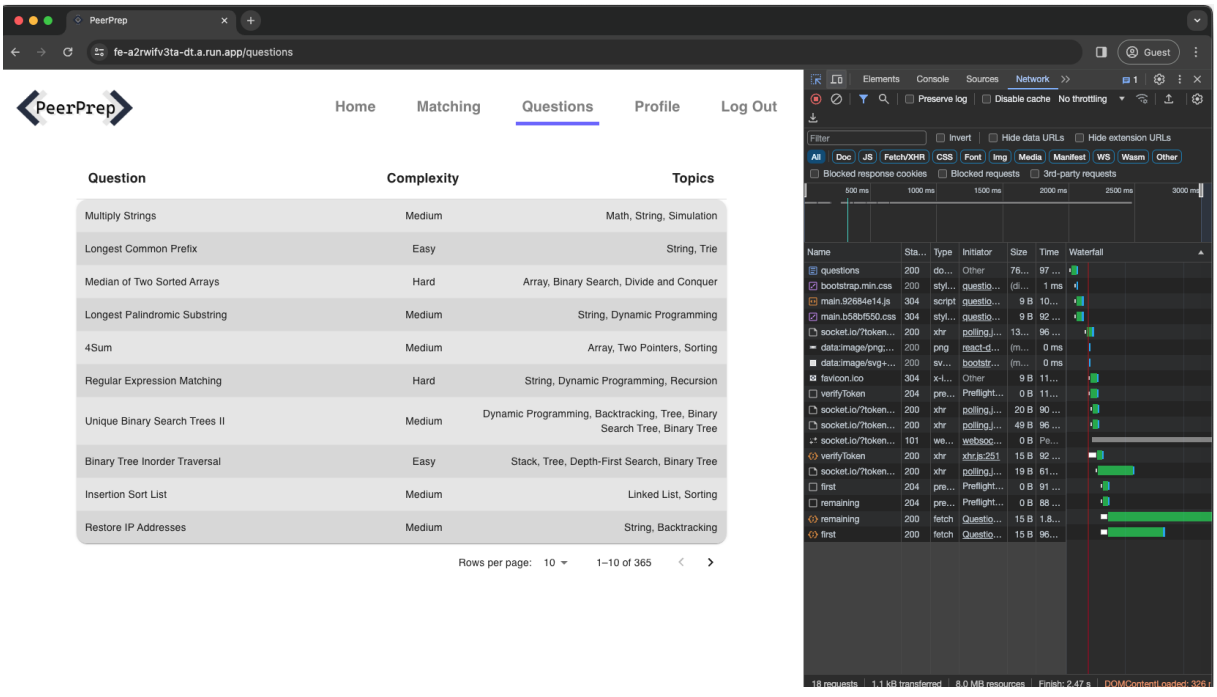

Figure B.8: Web Details Showing Question Loading to Question Page

B.3: Deployability Testing

NF3	Deployability	Priority
	NF3.1 The application should be able to be rolled back to previous versions	H
	NF3.2 The application should be deployed on the cloud	H
	NF3.3 The application should be automatically deployed based on the latest version	M
	NF3.4 The application should be deployed within 15 minutes after updating to the latest version	L
	NF3.5 The application be deployed with health checks and monitoring capabilities	L

NF3.1 (High): The application should be able to be rolled back to previous versions

The screenshot below shows the traffic management on Google Cloud Run where it can revert traffic to previous revisions of the application.

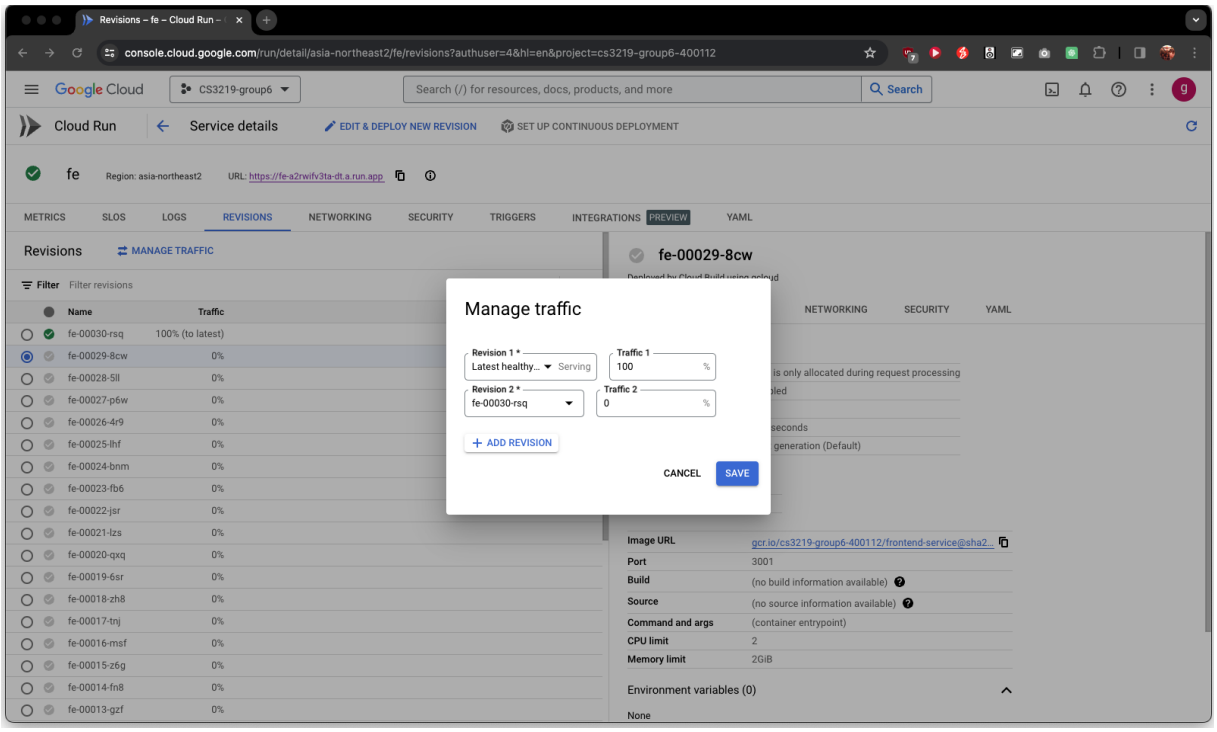

Figure B.9: Google Cloud Run Interface Showing Management of Traffic

NF3.2 (High): The application should be deployed on the cloud

The screenshot below shows all PeerPrep services deployed on Google Cloud Run.

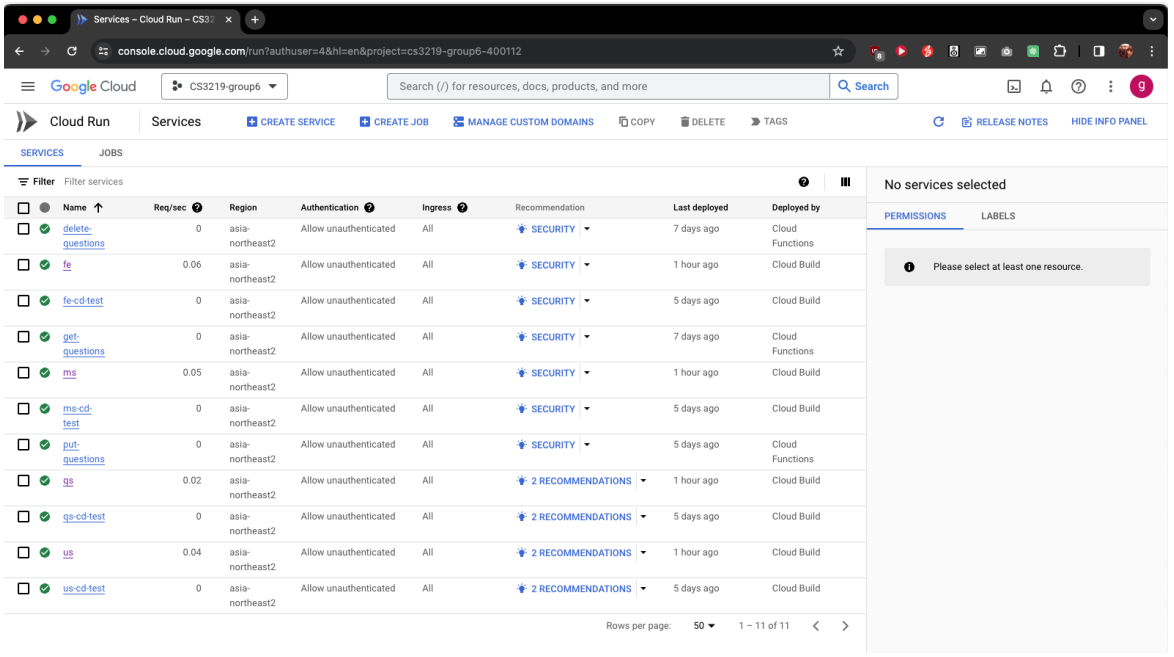

Figure B.10: Google Cloud Run Interface Showing All Services

NF3.3 (Medium): The application should be automatically deployed based on the latest version

The screenshot below shows the settings of Google Cloud Build for automatic deployment, where changes to the master branch of our GitHub repository triggers the deployment.

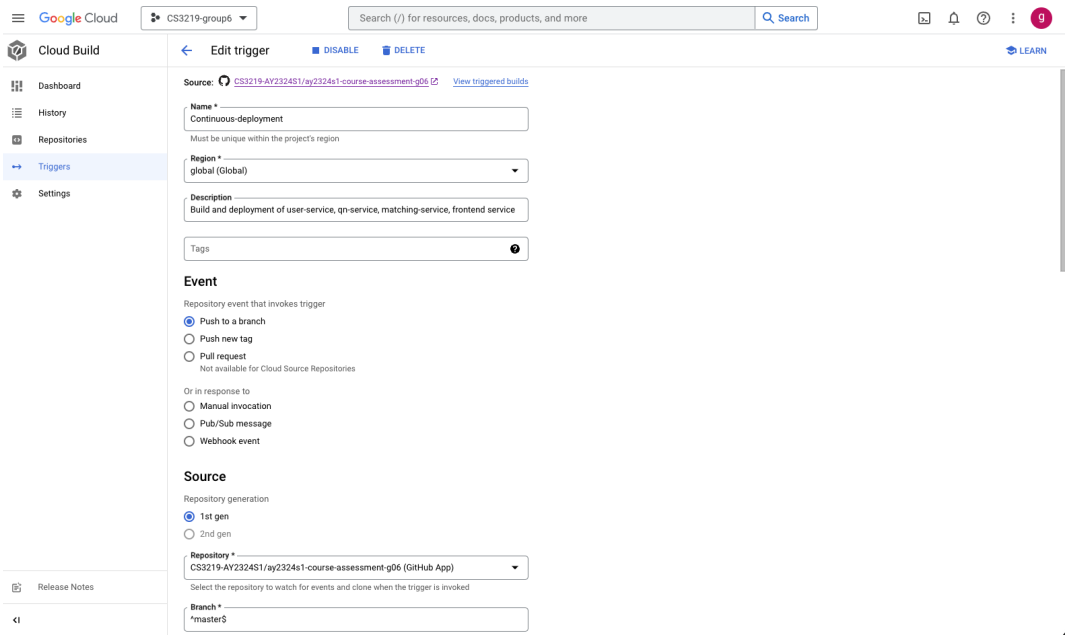

Figure B.11: Google Cloud Build Tracking Master Branch of Github Repository

NF3.4 (Low): The application should be deployed with 15 minutes after updating to the latest version

The screenshot below shows that it takes an average of 8 minutes for the deployment to be completed.

Figure B.12: Google Cloud Build Showing an Average of 8 Minutes for Automated Build and Deployment

NF3.5 (Low): The application be deployed with health checks and monitoring capabilities

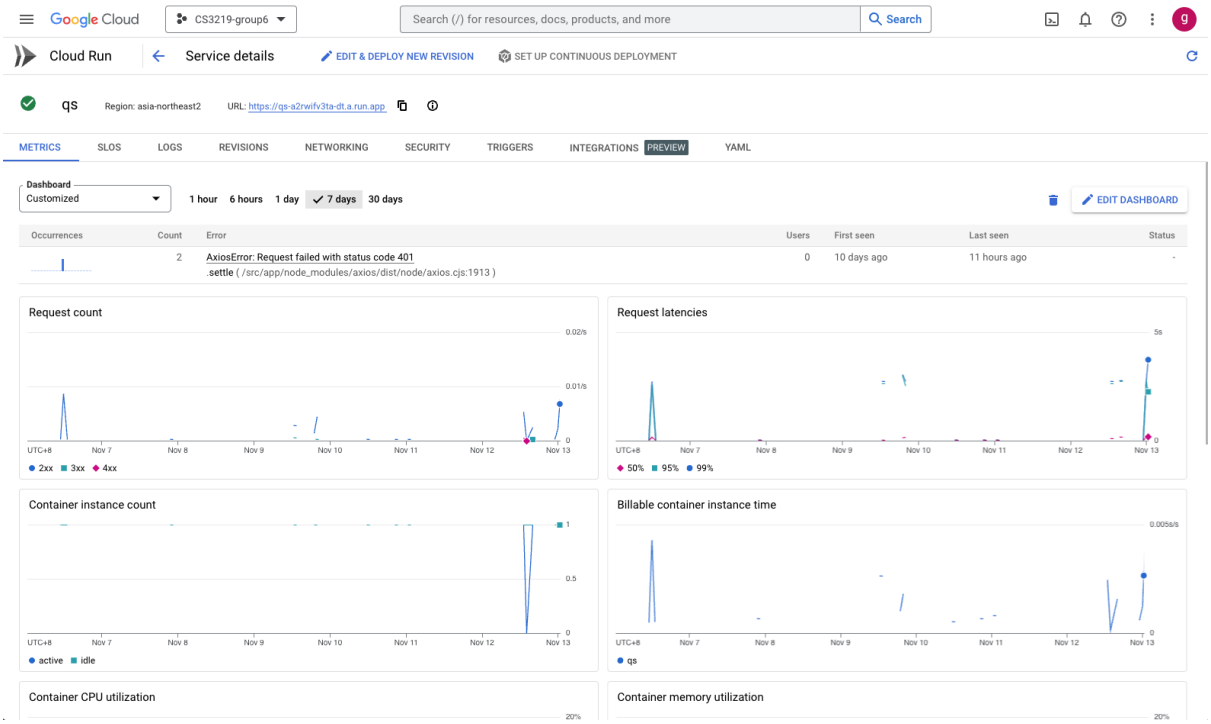

Figure B.13: Google Cloud Run Monitoring of Deployed Services