



---

School of Computing

**CS3219 Software Engineering Principles and Patterns  
AY23/24 Semester 1**

**Final Project Report**

**Group 10**

<b>Team Members</b>	<b>Matriculation Number</b>
Wong Chee Hong	A0217558W
Isaac Li Haoyang	A0233542M
Gavin Cho Weibin	A0229821A
Li Zhaoqi	A0233180R

**Tutor:** N Vijay Narayanan

<b>1</b>	<b><i>Individual Contributions</i></b>	<b>5</b>
<b>2</b>	<b><i>Introduction</i></b>	<b>8</b>
2.1	Background	8
2.2	Product Vision	8
2.3	Project Scope	8
2.4	Envisioned Usage	9
<b>3</b>	<b><i>Requirements and Specifications</i></b>	<b>10</b>
3.1	Requirements Organisation	10
3.2	Functional Requirements Prioritization	10
3.3	Functional Requirements Categories	10
3.4	Requirements Specification	11
3.4.1	User accounts and authentication	11
3.4.2	Matching Service	11
3.4.3	Question Service	12
3.4.4	Question History	13
3.4.5	Code Editor and Collaboration	13
3.4.6	Code Execution	14
3.4.7	Text Chat	14
3.5	Non-Functional Requirements	15
3.5.1	Non-Functional Requirements table	15
3.5.2	Reasoning for NFRs	16
3.6	Quality Attributes Prioritisation	17
3.6.1	Usability	18
3.6.2	Modifiability	18
3.6.3	Security	18
3.6.4	Maintainability	18
<b>4</b>	<b><i>Software development Process</i></b>	<b>19</b>
4.1	Weekly Sprints	19
4.2	Development Process	19
4.2.1	Workflow	19
4.2.2	GitHub Issue Tracker	20
4.3	DevOps	20
4.3.1	Local Development	20
4.3.2	Production Deployment	20
4.3.3	Continuous Integration	21
4.3.4	Continuous Delivery and Deployment	22
<b>5</b>	<b><i>Application Design</i></b>	<b>23</b>
5.1	Overall Architecture Design	23
5.2	Frontend	24
5.2.1	Frontend Tech-stack	24
5.2.2	Frontend Render Hierarchy	25
5.2.3	Choice of code editor	25
5.2.4	Consistency of visual elements	26
5.2.5	Choice of clear language	27

<b>5.3</b>	<b>Design Patterns .....</b>	<b>27</b>
5.3.1	Model View Controller Pattern (MVC) .....	27
<b>5.4</b>	<b>Microservices Design .....</b>	<b>29</b>
<b>5.5</b>	<b>Backend Microservices .....</b>	<b>29</b>
5.5.1	User Service.....	29
5.5.2	Question Service .....	32
5.5.3	Matching Service .....	34
5.5.4	Execution Service .....	41
5.5.5	API Gateway .....	44
<b>6</b>	<b>Future Enhancements .....</b>	<b>45</b>
<b>6.1</b>	<b>Features .....</b>	<b>45</b>
6.1.1	Video/Voice Chat.....	45
6.1.2	Login using Github.....	45
6.1.3	Gamification and Achievements .....	45
<b>6.2</b>	<b>Deployment.....</b>	<b>45</b>
<b>6.3</b>	<b>Cache .....</b>	<b>46</b>
<b>7</b>	<b>Learning Points .....</b>	<b>46</b>
<b>7.1</b>	<b>Deployments .....</b>	<b>46</b>
<b>7.2</b>	<b>Team Processes.....</b>	<b>46</b>
<b>8</b>	<b>Appendix A: Product Screen Shots.....</b>	<b>47</b>
<b>8.1</b>	<b>Account pages.....</b>	<b>47</b>
8.1.1	Login Page .....	47
8.1.2	Sign Up Page .....	47
8.1.3	Account profile (User) .....	48
8.1.4	Account Profile (Admin) .....	48
8.1.5	Reset Password.....	49
<b>8.2</b>	<b>Portal Pages (User).....</b>	<b>50</b>
8.2.1	Logged-in Landing Page.....	50
8.2.2	Question Bank Page .....	50
<b>8.3</b>	<b>Portal Pages (Admin).....</b>	<b>51</b>
8.3.1	Logged-in Landing Page.....	51
8.3.2	Question Bank Page .....	51
8.3.3	Create Question Page .....	52
8.3.4	Edit Question Page .....	52
<b>8.4</b>	<b>Code Editor Pages .....</b>	<b>53</b>
8.4.1	Code Editor Page .....	53
8.4.2	Code Editor Page with Question Tab.....	53
8.4.3	Choosing template code for common algorithms .....	54
8.4.4	Choose the language to code in .....	54
<b>8.5</b>	<b>Matching Process .....</b>	<b>55</b>
8.5.1	Looking for a match.....	55
8.5.2	Match found .....	55
<b>8.6</b>	<b>Collaborative Coding Session Page .....</b>	<b>56</b>
8.6.1	Session Page .....	56
8.6.2	Session Page with Question Tab.....	56
8.6.3	Session Page with Chat.....	57

<b>9</b>	<b><i>Appendix B: Use Cases.....</i></b>	<b>58</b>
----------	--	-----------

# 1 Individual Contributions

Chee Hong	<p><b>Code</b></p> <p>Frontend:</p> <ul style="list-style-type: none"><li>- All development for user-profile viewing, management, and authentication components and integration with backend API</li><li>- Integration of Question History components with profile page</li><li>- Integrated frontend-backend networking for question-service</li></ul> <p>Backend:</p> <ul style="list-style-type: none"><li>- Implement user-service and database to store users and files</li><li>- Implement authentication with jwt tokens</li><li>- Implement question service and database to store questions and testcases</li><li>- Integration of script with question service to populate question repository</li><li>- Implement API gateway</li><li>- Self-hosting of Judge0</li><li>- Develop and integrate code execution service</li></ul> <p>Devops:</p> <ul style="list-style-type: none"><li>- Docker-compose deployment on local machine</li><li>- Production deployment to GCP cloud platform</li><li>- Setting up all CI/CD for building and pushing of docker images to private registry and deployment on GCP cloud platform</li></ul> <p>Miscellaneous:</p> <ul style="list-style-type: none"><li>- Integration and code review of pull requests from other members</li></ul>
Zhaoqi	<p><b>Code</b></p> <p>Frontend:</p> <ul style="list-style-type: none"><li>- Implemented various reusable components like navigation bar, Closeable Tag, DeleteQnBtn, Paginator, Question Table, markdown editor, collab editor ...</li><li>- Designed and implemented most pages except for login, logout, home page and profile page.</li></ul>

	<ul style="list-style-type: none"> <li>- Designed and implemented View Question page and all its components except for the chatbox component.</li> <li>- Designed and implemented the Edit Question page and all the components in it.</li> <li>- Designed and implemented the Question Bank page and all its components except the filter bar.</li> <li>- Designed and implemented the client-side matching logic to work with matching service.</li> <li>- Designed and implemented peer-to-peer syncing and initializing logic to work with CRDS style collaborate sync libraries.</li> <li>- Integrate collaborative service with frontend</li> <li>- Integrate Matching service with frontend</li> <li>- Integrate some of execution service with frontend</li> <li>- Syntax highlighting for markdown as well as code editor</li> <li>- Multi language support for code editor as well as template</li> </ul> <p>Backend:</p> <ul style="list-style-type: none"> <li>- Implement matching service that allows users to be matched in multiple categories or match specific questions.</li> <li>- Implement collaboration service and all related logics</li> <li>- Implements synchronous code execution to check for validity in code execution service</li> <li>- Callback logic in execution service</li> </ul> <p>Miscellaneous:</p> <ul style="list-style-type: none"> <li>- Integration and code review of pull requests from other members</li> <li>- Refactor overused code / bad logic found in other components.</li> <li>- Script to scrape dummy code data from cpbook repo</li> </ul> <p>Report</p> <ul style="list-style-type: none"> <li>- [2] Introduction</li> <li>- [3] Requirements and Specifications</li> <li>- [4] Software Development Process</li> <li>- [5] Application Design</li> </ul>
Gavin	<p><b>Code</b></p> <p>Frontend:</p> <ul style="list-style-type: none"> <li>- Implemented and designed Chatbox component between 2 users</li> <li>- QnHistory: Implemented and designed the Solved Table component</li> <li>- Implemented Add Question and Delete Question pages</li> <li>- Organised and designed frontend folder structure</li> </ul> <p>Backend:</p>

	<ul style="list-style-type: none"> <li>- Implemented a solved question model to differentiate solved and unsolved questions</li> <li>- Built on the user service to retrieve solved questions</li> <li>- Added new user API's to retrieve solved questions</li> <li>- Built new schema in Prisma database to handle solved questions</li> </ul> <p>Miscellaneous:</p> <ul style="list-style-type: none"> <li>- Set up code base for react project</li> <li>- Set up github classroom and repo for the team</li> <li>- Integration and code review of pull requests from other members</li> </ul> <p>Report</p> <p>Contributed to the following section:</p> <ul style="list-style-type: none"> <li>- Overall report structure</li> <li>- [2] Introduction</li> <li>- [3] Requirements and Specifications</li> <li>- [4] Software Development Process</li> <li>- [5] Application Design</li> <li>- [6] Future Enhancements</li> <li>- [7] Learning Points</li> <li>- [8] Appendix A: Product Screen Shots</li> </ul>
Isaac	<p>Code</p> <p>Frontend:</p> <ul style="list-style-type: none"> <li>- Implemented and designed Progress Bar component to track solved questions</li> <li>- Implemented functionality to filter questions by difficulty</li> <li>- Implemented Homepage, Questions table, View Questions page and local storage for Assignment 1</li> </ul> <p>Backend:</p> <ul style="list-style-type: none"> <li>- Implemented code execution service</li> <li>- Implemented server to send requests to Judge0 RapidAPI</li> <li>- Added new API's to retrieve execution results</li> <li>- Implemented script to populate the questions repository</li> </ul> <p>Report</p> <p>Contributed to the following section:</p> <ul style="list-style-type: none"> <li>- [2] Introduction</li> <li>- [3] Requirements and Specifications</li> <li>- [4] Software Development Process</li> <li>- [5] Application Design</li> <li>- [8] Appendix A: Product Screen Shots</li> </ul>

## 2 Introduction

### 2.1 Background

In the realm of job applications, students often encounter formidable technical interviews, grappling with various challenges ranging from expressing their thought process clearly to struggling with problem-solving. Additionally, the repetitive nature of practicing questions can become tiresome. To tackle these hurdles, our team developed PeerPrep, an online platform tailored for interview preparation. PeerPrep facilitates peer matching, enabling students to pair up and tackle whiteboard-style interview questions collaboratively. The primary goal of this application is to foster a collaborative learning environment where students can revise core algorithmic concepts together, engaging in mutual learning and alleviating the monotony of solo preparation.

### 2.2 Product Vision

The product vision, encompassing the team's overarching aspirations for the application, was formalized, and honed before initiating any planning or development efforts. This vision served as a guiding principle for subsequent product design and requirement definition. The team's vision statement is as follows:

*“A web application for collaboration on coding interview questions. With an inviting, minimalistic and intuitive user interface, users can practice on coding questions and collaborate with other users in real time. With real time collaboration, users can work together and solve questions while bouncing their computational thought processes with their peers.”*

### 2.3 Project Scope

In order to better define what our application should achieve for our end users, we have settled upon the following definitions for our end-users, as well as the general functionality that our application should aim to provide. These delineations culminated in the scope document, which provided a comprehensive description of the parameters guiding the product's design. The scope document is presented below.

<b>Target User(s)</b>	<ul style="list-style-type: none"><li>• Individuals with prior coding experience.</li><li>• Students and professionals who are actively preparing for coding interviews.</li><li>• Professionals seeking to refresh their technical skills.</li><li>• Individuals are interested in learning modern coding patterns.</li><li>• Beginners who are in the process of acquiring knowledge in algorithms and problem-solving.</li><li>• People searching for coding partners or collaborators.</li></ul>
-----------------------	--



<b>High-Level Functionality</b>	<p>The application should</p> <ul style="list-style-type: none"> <li>• Provide a collaborative workspace facilitating real-time collaboration between two users.</li> <li>• Enable both users to simultaneously view the coding question, promoting collaborative problem-solving.</li> <li>• Offer a selection of coding questions categorized by varying difficulty levels.</li> <li>• Display test cases and their expected outcomes to aid users in evaluating the correctness of their solutions.</li> </ul>
-------------------------------------	---

## 2.4 Envisioned Usage

This section aims to provide a detailed breakdown of the high-level functionality outlined in the scope section. The process flow described below represents the essential functions expected from the product. It primarily focuses on the standard workflow and does not encompass potential failure scenarios or unforeseen events.

A typical user would interact with the product in the following sequence:

1. The user opens a web browser and accesses the application.
2. If the user does not have an existing account, they can register for one. Otherwise, they log in to their account.
3. The user can either enter a matching queue to find a partner for coding collaboration or solve the question by himself.
4. Upon finding a match, the user enters the collaborative workspace environment.
5. A coding question is presented to all users in the same workspace, and they work together to solve it.
6. Once the question has been solved a success screen will appear and all users will be kicked from the collaborative room and returned to their homepage.

## 3 Requirements and Specifications

This section succinctly encapsulates the fundamental concepts and pivotal decisions that steer the requirements development process.

### 3.1 Requirements Organisation

To enhance requirements management, we've systematically categorized the requirements into two primary groups: functional requirements (FR) and non-functional requirements (NFR). A hierarchical requirement coding is used to help identify the requirement category, which differs for functional and non-functional requirements. The coding used is as follows.

<code>&lt;F(unctional) / N(on-Functional)&gt;&lt;Category&gt; . &lt;Running Number&gt;</code>
---

### 3.2 Functional Requirements Prioritization

A priority label—**high**, **medium**, **low**—is assigned to each requirement, reflecting its significance in achieving the project's objectives:

**High Priority** - Must be completed for the Minimum Viable Product.

**Medium Priority** - Important for ensuring a smoother user experience.

**Low Priority** - Implementation is contingent on available time and resources.

### 3.3 Functional Requirements Categories

Each functional requirement (FR) is tagged to a category. The following table summarizes the high-level intent of each category and is derived from the envisioned usage defined earlier.

FR Category		Intent
FR1	User accounts and authentication	The system should be able to distinguish various users. Users should also be able to identify each other
FR2	Matching and Queuing	The system should be able to pair two users up to put into a room
FR3	Question Repository	Provide a database of questions for users to work on
FR4	Question History	Allow users to keep track of their attempts on the platform
FR5	Code Editor & Collaboration	Provide a workspace for users to code with capability for real-time collaboration among multiple users
FR6	Code Execution	Compile and execute code written by the users in the platform, allowing for immediate testing and debugging
FR7	Text Chat	Facilitate communication and effective collaboration in real-time between users

### 3.4 Requirements Specification

The following section details each functional requirement and the non-functional requirements for each category.

#### 3.4.1 User accounts and authentication

FRs	Description	Priority
FR1.1	The system should allow users to create an account with a username and password	High
FR1.2	The system should ensure that every account created has a unique username	High
FR1.3	The system should allow users to log into their accounts by entering their username and password	High
FR1.4	The system should allow users to log out of their account	High
FR1.6	The system should allow users to change their password	Medium
FR1.7	The system should enable users to establish distinctive visual identities for easy recognition by other users	Medium
FR1.8	The system should also allow users to modify their profile to tailor how they are perceived by other users	Medium

#### 3.4.2 Matching Service

FRs	Description	Priority
FR2.1	The system should allow users to select the difficulty levels (easy, medium, hard) of the questions they wish to attempt	High
FR2.2	The system should be able to match two waiting users with same difficulty levels	High
FR2.3	The system should direct two users that have been matched to the same room	High
FR2.4	If there is a valid match, the system should match the users within 30s	High
FR2.5	The system should inform the users that no match is available if a match cannot be found within 30 seconds	High

FR2.6	The system should disallow the same users from joining multiple queues	High
FR2.7	The system should provide a means for the users to leave a room once matched	Medium
FR2.8	The system should allow users to exit the matching process before the timer ends.	Medium
FR2.9	The system should allow users to change the difficulty level chosen if a match is not yet found.	Medium
FR2.10	The system should allow user to reconnect the queue/match should he/she be accidentally disconnected	Medium
FR2.11	The system should allow the user to queue multiple ranges of difficulty level (i.e. all difficulties)	low
FR2.12	The system should allow user to queue specific questions instead of difficulties	low
FR2.13	The system should allow user to rejoin the queue should the user be disconnected accidentally	low
FR2.14	The system should allow the same user to join the same session from multiple instances	low

### 3.4.3 Question Service

FRs	Description	Priority
FR3.1	The system contains questions with difficulty values that can be categorized in to basic, easy, medium and hard	High
FR3.2	The system should allocate questions that match with the user's selected difficulty level	High
FR3.3	Within the same difficulty level, the system should generate selected question randomly to the users	High
FR3.4	The system should allow users to generate a new question while in the same room	Medium
FR3.5	The system should display an image for applicable questions to help the user understand the question	Low
FR3.6	The system allows users to select questions of a particular topic	Low

FR3.7	For each question, the system should provide template code for users to fill in	Low
-------	---	-----

#### 3.4.4 Question History

FRs	Description	Priority
FR4.1	The system should display the questions that the user has successfully completed	High
FR4.2	The system should display the past attempts the questions the user is working on	High
FR4.3	The system should display a progress bar to indicate the users progress for every question	Medium
FR4.4	The system should be connected to a persistent Postgres datastore for storin history information	High
FR4.5	The system should allow the user to view detail of past attempt	Medium
FR4.6	The system should allow the user to sort the attempts by difficulty and date	Low

#### 3.4.5 Code Editor and Collaboration

FRs	Description	Priority
FR5.1	The system should allow users to edit the content in a workspace	High
FR5.2	The system should automatically update the workspace changes by the other user	High
FR5.3	The system should alert the user once the other user has joined	High
FR5.4	The system should alert the user when the partner leaves the room	Medium
FR5.5	The system should allow the user to submit his code	High

FR5.6	The system should allow the user to select his preferred coding language	Medium
FR5.7	The system should allow the user to leave the room	High
FR5.8	The system should allow users to select coding templates	Low
FR5.10	The system should allow the partner to see the location of each other's cursor	Medium

#### 3.4.6 Code Execution

FRs	Description	Priority
FR6.1	The system should connect to a queue to submit code execution jobs	High
FR6.2	The system should execute the jobs submitted to the queue	High
FR6.3	The system should return the execution results of the jobs	High

#### 3.4.7 Text Chat

FRs	Description	Priority
FR7.1	The system should allow display a chatbox for users to interact with	High
FR7.2	The system should allow users to communicate via messaging in the chatbox	High
FR7.3	The system should display both the users and other user's messages on the chatbox	High

## 3.5 Non-Functional Requirements

### 3.5.1 Non-Functional Requirements table

NFRs	Description	Priority
<b>Security (NFR1)</b>		
NFR1.1	User passwords should be hashed and salted before storing.	High
NFR1.2	Users should be automatically logged out after 30 mins of inactivity	Low
NFR1.3	The password should be alphanumeric and be at least of length 8	Low
NFR1.4	The system should only allow users to access APIs that they are authorized to	High
<b>Availability (NFR2)</b>		
NFR2.1	The service should perform without failure in 95% percent of use cases in a month.	High
NFR2.2	The services should have an uptime of 99%.	High
<b>Performance (NFR3)</b>		
NFR3.1	The system should respond to frontend requests within 5 seconds.	High
NFR3.2	Any diagrams or animations attached to the question should load within 3 seconds.	High
NFR3.3	The room should be created within 7 seconds after being matched.	High
NFR3.4	The code editor should be loaded within 5 seconds of being matched.	Medium
NFR3.5	The question should be loaded within 5 seconds of being matched.	Medium
NFR3.6	The collaborative editor should not have a delay of more than 1 seconds from when a user makes a modification to when the other user sees it when under normal internet conditions.	High

NFR3.7	The chat service should have an end-to-end delay of at most 1 second when under normal internet conditions.	High
<b>Usability (NFR4)</b>		
NFR4.1	The system should support concurrent matchmaking without any error.	High
NFR4.2	The stored history data must be consistent.	High
NFR4.4	The UI should have a consistent theme.	Medium
NFR4.5	The application should be accessible via a public URL.	Medium
<b>Scalability (NFR5)</b>		
NFR5.1	The system should support up to 100 concurrent matchmaking requests.	Medium
NFR5.2	All requests to the application should be facilitated by an API Gateway.	High

### 3.5.2 Reasoning for NFRs

NFRs	Description
<b>Security (NFR 1)</b>	
NFR1.1	User passwords should be stored securely to prevent malicious attackers from stealing sensitive information.
NFR1.2	Inactive users should be logged out to reduce load on server and user's own devices.
NFR1.3	The password should be hard to guess for malicious attackers.
NFR1.4	Allowing users to edit the questions will result in chaos for other users.
<b>Availability (NFR 2)</b>	
NFR2.1 & NFR2.2	The service should perform without failure in 95% percent of use cases in a month.



Performance (NFR 3)	
NFR3.1 To NFR3.8	These metrics encapsulate the essential functionalities of our application. Ensuring their smooth operation, devoid of significant delays or latency, is vital for enabling users to concentrate on programming problems and their interview preparation process as well as to maintain a responsive page.
Usability (NFR 4)	
NFR4.1	The system will be used by the student population thus it has to support multiple matches.
NFR4.2	The history allows users can see their past attempts and keep track of their progress.
NFR4.4	The UI is crucial to usability.
NFR4.5	A public URL helps users to access our page easily, thus improving usability.
Scalability (NFR 5)	
NFR5.1	The system be designed to support a large student population.
NFR5.2	API Gateway supports the scalability of other features.

### 3.6 Quality Attributes Prioritisation

The team established quality attributes to influence requirement development and ranked them based on their significance in the product design. These attributes were categorized into four groups, with three priority levels, and a final category labelled as "irrelevant." The table below presents the ranking of these quality attributes:

High Priority	Medium Priority	Low Priority
Usability	Installability	Availability
Modifiability	Performance	
Maintainability	Robustness	
Security		

Table 1 – Prioritisation of Quality Attributes

The team's primary focus in the development of PeerPrep is on usability, modifiability, security and maintainability while the other quality attributes may influence the requirements specifications either directly or indirectly. The selection of these quality attributes is justified as follows:

### **3.6.1 Usability**

Ensuring a seamless user experience is a paramount focus for PeerPrep. Our vision statement underscores the significance of a user-friendly and intuitive user interface (UI), where actions are immediately comprehensible without the need for elaborate explanations. This design philosophy caters to users across all proficiency levels, from beginners to experienced coders. Usability encompasses elements like effortless navigation, transparent instructions, and an intuitively organized layout, all contributing to an outstanding user experience.

### **3.6.2 Modifiability**

Modifiability stands as a pivotal quality attribute for the PeerPrep system, given the rapid pace of our project's development. The system is crafted to readily embrace changes accompanying the introduction of new features. This approach aligns with fundamental software engineering principles, such as the Open-Closed Principle (OCP) and the Single Responsibility Principle (SRP).

### **3.6.3 Security**

Incorporating security as a quality attribute underscores our project's commitment to adhering to best security practices and establishing a baseline security level for the application. While the system may not handle critical data, addressing security concerns from the outset mitigates the risk of potential security breaches, reflecting our proactive approach to safeguarding user information and ensuring system integrity.

### **3.6.4 Maintainability**

Maintainability refers to how easy it is to maintain the software, which includes fixing bugs, adding new features, or meeting new requirements. Maintainability is a crucial quality attribute for long-term efficiency and speed for PeerPrep, as it reduces the complexity and cost of the ongoing enhancement and correction process while ensuring we meet our deliverables on time.

## 4 Software development Process

### 4.1 Weekly Sprints

Our team adhered to the Scrum framework, conducting weekly sprints every Sunday. At the onset of each sprint, we engage in a comprehensive discussion on the week's progress. This includes sharing completed tasks, outlining pending work, and addressing any challenges encountered. Subsequently, we collaboratively formulate the backlog for the upcoming week, aligning tasks with the functional and non-functional requirements (FRs and NFRs). Tasks are then assigned based on each team member's capabilities. If there are any group tasks scheduled for the week, we address them during this meeting.

### 4.2 Development Process

To streamline the management of functional requirements, we organized them into distinct phases. Typically, all requirements within one phase must be finalized before progressing to the next. However, exceptions can be made for the lowest-priority task in a phase, treating it as a nice-to-have without significantly impacting the transition to the subsequent phase. The prioritization of requirements is established globally and generally decreases with each phase. This prioritization strategy acknowledges that requirements in later product phases naturally carry lower priority in the overall project context.

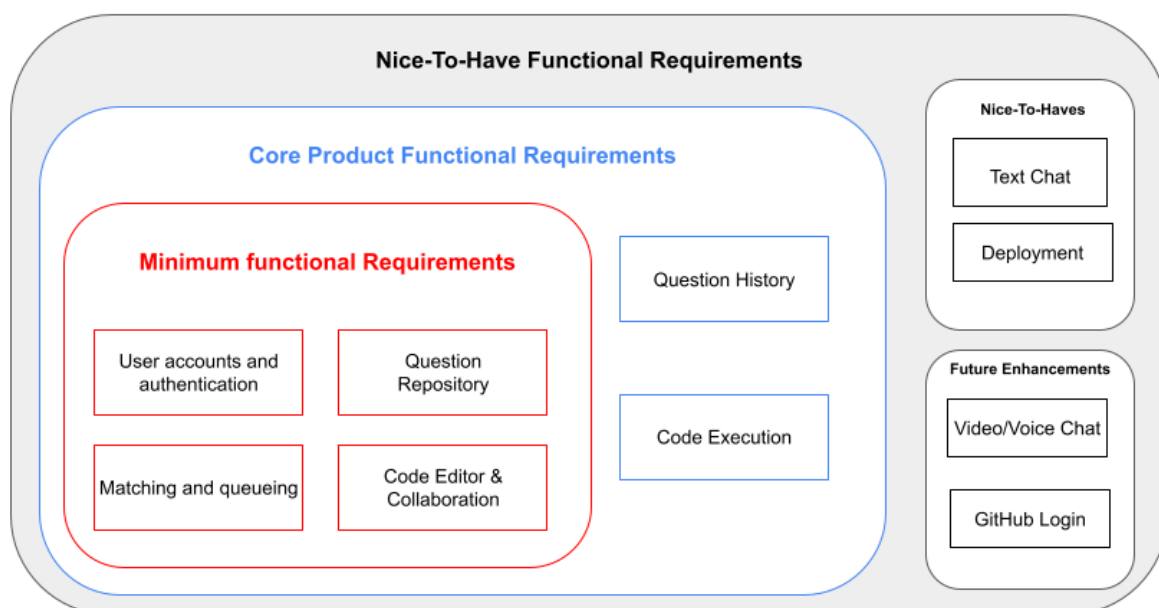


Fig 1. Global View of Requirements

#### 4.2.1 Workflow

Integrating phases alongside our weekly sprints within the SCRUM framework represents a strategic shift in our development strategy. This enhancement is driven by a concentrated emphasis on prioritizing the most critical work items for each phase, fostering a more coordinated advancement aligned with pivotal project milestones. This refined approach

guarantees a systematic progression even within our weekly sprints, furnishing a structured framework for our development rhythm. The choice to implement these adjustments underscores our team's adaptability and agility, addressing project obstacles with a tailored method that aligns with our distinct requirements and operational dynamics.

#### **4.2.2 GitHub Issue Tracker**

Our team made use of the Issue tracker in GitHub to track our progress and give others a rough understanding of what we were working on.

The following tables show the tags used to tag PRs/Issues for issue tracking purposes:

Tag	Description
bug	Something is not working as expected
Nice to have	An enhancement feature to the app
In Progress	The PR/Issue is still in progress and should not be merged

Team members would create an issue whenever they were working on a certain part of project with a brief description of the issue. Tags would be labelled appropriately, and issues would be removed once the pull request related to that issue has been merged

### **4.3 DevOps**

#### **4.3.1 Local Development**

To facilitate local development, a docker-compose project was created to orchestrate the startup of each individual microservice. Each service is dockerized and configured to be in the same docker network and exposes a port so that the services could communicate with each other via a service's IP address and port, while the frontend is run locally on the node environment to allow for hot-reloading and development.

#### **4.3.2 Production Deployment**

For production deployment, optimized production builds were created for each of the microservices, as well as the frontend. For the frontend, this involved invoking the create-react-app build script. For the microservices in the backend, this involved compiling all the TypeScript files into JavaScript.

To automate the production builds, we use a separate production version Docker file that allows for multistage build. In the build stage, microservices will be compiled into JavaScript, and an optimized build will be made for frontend. When the docker image is being executed, the docker image will serve the compiled/build version of the microservice/frontend

accordingly. All the production version Docker files can be built simultaneously with a docker compose project.

```
1  # Build stage
2  FROM node:18-alpine as build
3  WORKDIR /app
4  COPY package.json ./
5  COPY package-lock.json ./
6  RUN npm ci
7  COPY . ./
8  RUN npm run build
9
10
11 # Serve stage
12 FROM nginx:alpine
13 COPY --from=build /app/build /usr/share/nginx/html
14 EXPOSE 80
15 CMD ["nginx", "-g", "daemon off;"]
```

Fig 2. Example of a multi-stage docker build for frontend

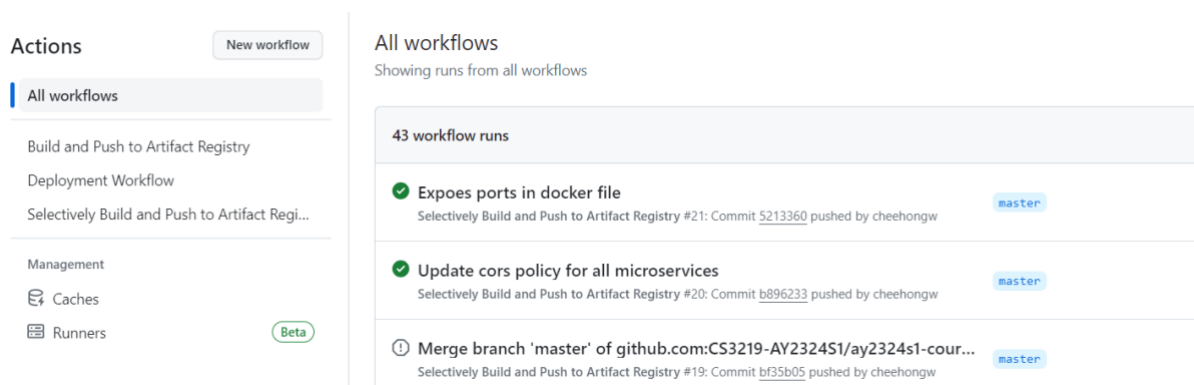
A Google Compute VM instance with the correct ports exposed will execute the docker compose project, effectively serving our production build.

### 4.3.3 Continuous Integration

To automate the process of building the production docker images, a GitHub workflow was created. The workflow is named “Selectively Build and Push to Artifact Registry” and does exactly what it says.

Upon push to the master branch, the workflow scans the repository for changes in any of the microservices or frontend directories. If a change was detected, the workflow runs `docker-compose build` to rebuild the image for that service. It then pushes the build to the Google Artifact Registry. Such a selective building strategy was used to minimize the lag time between pushing and deploying our updated builds.

A workflow called “Build and Push to Artifact Registry” was also created, but it is manually dispatched and builds and push docker images for all services.



The screenshot shows the GitHub Actions interface. On the left, under 'Actions', there is a list of workflows: 'All workflows', 'Build and Push to Artifact Registry', 'Deployment Workflow', and 'Selectively Build and Push to Artifact Registry...'. The 'All workflows' tab is selected. On the right, under 'All workflows', it says 'Showing runs from all workflows'. Below this, there is a section titled '43 workflow runs'. The first two runs are successful (green checkmarks) and are for the 'Selectively Build and Push to Artifact Registry' workflow. The first run is 'Exposes ports in docker file' (Commit 5213360) and the second is 'Update cors policy for all microservices' (Commit b896233). Both are pushed by 'cheehongw' on the 'master' branch. The third run is a failed run (yellow warning icon) for the 'Merge branch 'master' of github.com:CS3219-AY2324S1/ay2324s1-cour...' workflow (Commit bf35b05), also pushed by 'cheehongw' on the 'master' branch.

Fig 3. Workflows

#### 4.3.4 Continuous Delivery and Deployment

Finally, to achieve continuous deployment, the “Selectively Build and Push to Artifact Registry” workflow triggers the deployment workflow if a new build was pushed to the registry. The deployment workflow tells the VM to run docker-compose up using the docker-compose.yml project specified in the project repository, which pulls the updated images from the registry and serves them.

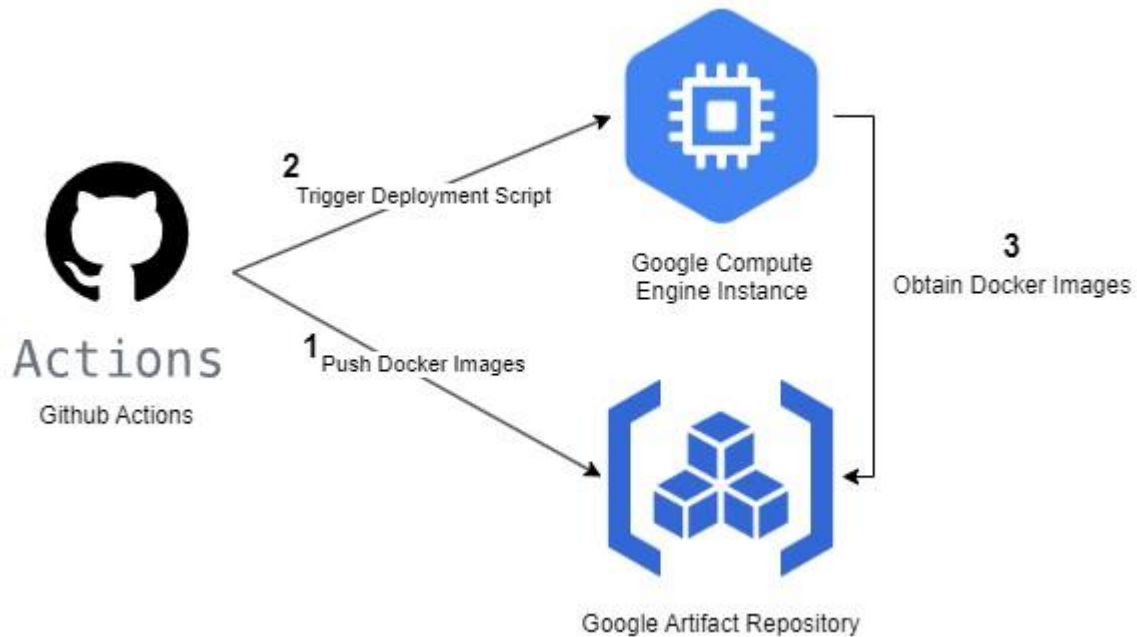


Fig 4. Continuous Delivery and Deployment

## 5 Application Design

This section details the design decisions and implementation details for our application.

### 5.1 Overall Architecture Design

Our application was designed around a microservice-based architecture for its modularity and strong adherence to the Single Responsibility Principle (SRP). In comparison to monolithic systems, such modularity allows for the development of independent components that can be deployed, updated, and scaled separately, facilitating flexibility and agility in both development and maintenance. Its strong adherence to SRP also simplifies understanding, development, and testing of the system. From a business perspective, adopting a microservices architecture allows for faster time-to-market, as we can develop and deploy services independently and in parallel.

The system architecture is shown below.

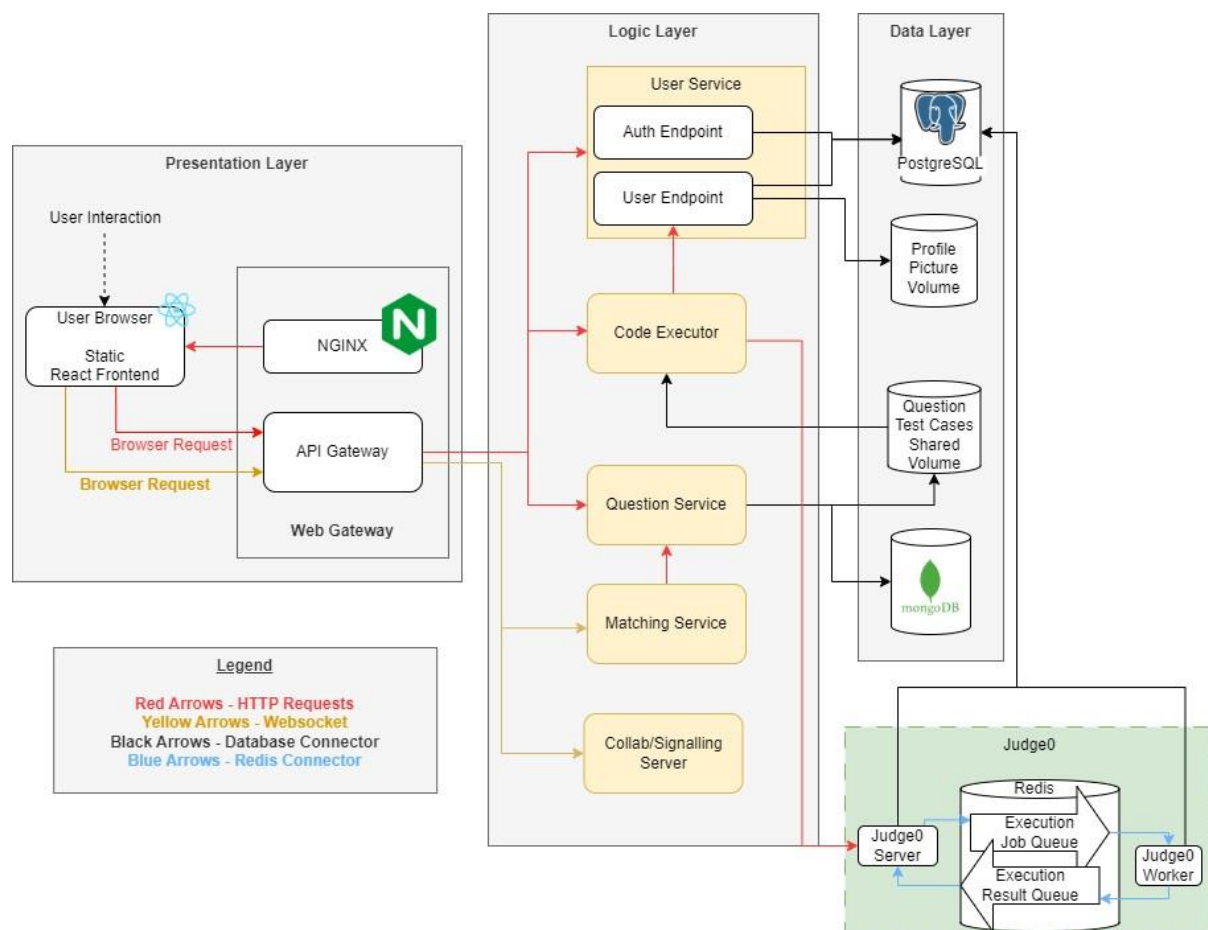


Fig 5. System Architecture

## 5.2 Frontend

The frontend consists of the presentation layer. The frontend is hierarchically structured, made up of multiple layers. Subsequent sections will detail the frontend architecture and how each component interfaces with one another. The frontend architecture is illustrated below.

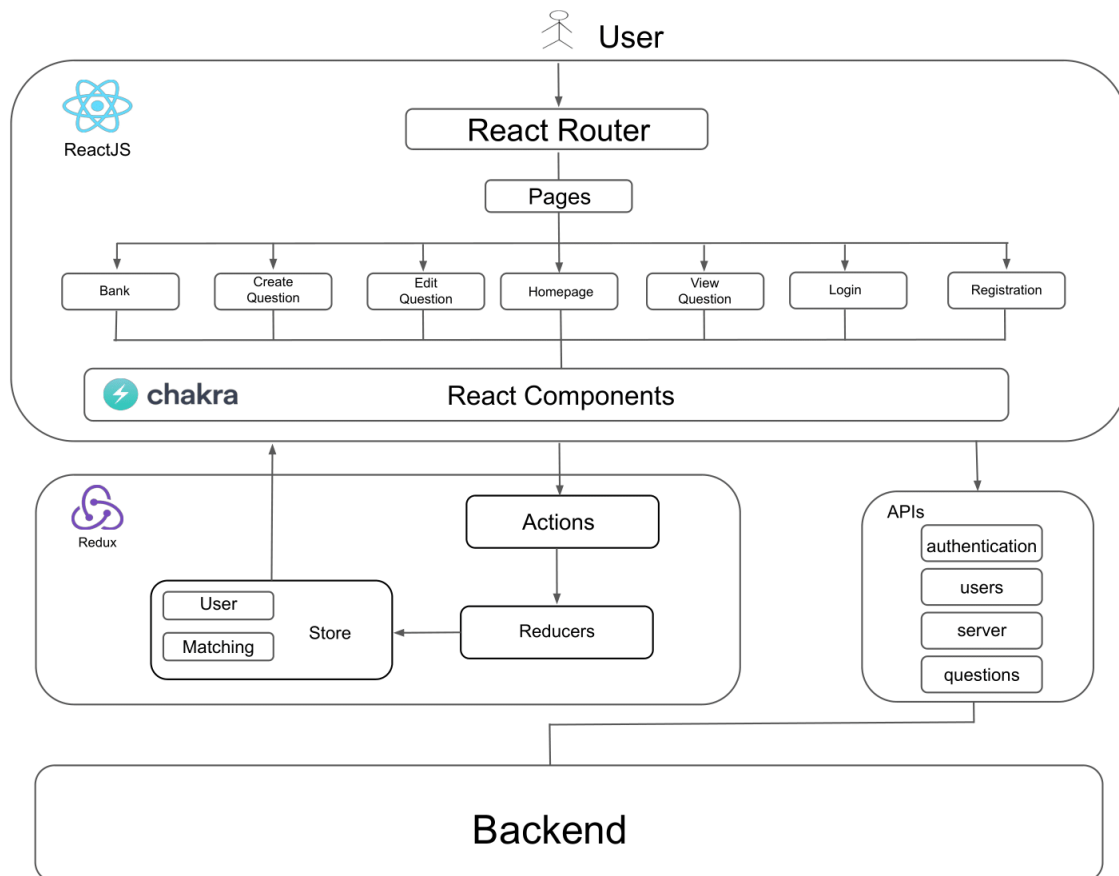


Fig 6. Frontend Component Architecture

### 5.2.1 Frontend Tech-stack

The following table lists the main technologies used and their rationale in the development of the application's frontend.

	Technologies	Rationale
Frontend	React	<ul style="list-style-type: none"><li>• The component-based architecture approach allows for the development of micro-frontends</li><li>• Each frontend component represents a specific feature/sub-component of the entire application. This allows the split of application into decoupled component codebase</li></ul>



		<ul style="list-style-type: none"> <li>• Allow for sharing of reusable components across the application</li> <li>• A lot of community support</li> <li>• Allows for Chakra UI library to be used to decrease frontend component build time</li> </ul>
	Judge0	<ul style="list-style-type: none"> <li>• A lot of community support</li> <li>• Provides detailed error handling information</li> <li>• Easy to execute when testing code with various inputs</li> </ul>
	CodeMirror	<ul style="list-style-type: none"> <li>• Rich programming API with a focus on extensibility and allows for easy integration of code editor with functionalities like syntax highlighting and auto-indentation into existing UI</li> </ul>
	Yjs	<ul style="list-style-type: none"> <li>• The Allow for binding of a third-party editor to a Yjs document (e.g CodeMirror)</li> <li>• Support awareness and presence which allow communication of more information which aid collaboration such as cursor position</li> <li>• Allow reliable syncing of data structures like maps and arrays across networks</li> <li>• Reliable implementation of Conflict-Free Replicated Data Type (CRDT).</li> </ul>

### 5.2.2 Frontend Render Hierarchy

When a user loads the frontend, it first passes through the React Router, which acts as a demultiplexer and determines the page to render. These pages then contain React components, which are rendered based on the state information provided by the Redux store.

### 5.2.3 Choice of code editor

We opted for Code Mirror as our application's code editor due to its aptness for collaborative editing. Unlike other editors like Monaco Editor, Code Mirror offers much easier integration with CRDT libraries like YJS through officially supported plugins. This feature enables the frontend to transmit changes to the collaboration room solely when prompted by user events converted as individual actions, streamlining the process. Moreover, Code Mirror boasts various beneficial extensions, including syntax highlighting, editor theming, and code autocompletion based on the chosen programming language. We anticipate that these functionalities will empower users to create a more conducive coding environment, ultimately enhancing their performance.

### 5.2.4 Consistency of visual elements

The PeerPrep application maintains visual consistency across all its pages, incorporating the same font style, icon design, and color scheme throughout the interface. Additionally, our design layout features a fixed navigation bar at the top of every page, ensuring a consistent user experience.

This uniformity in design contributes to a more coherent and integrated application. It simplifies user navigation by eliminating the need to learn new interfaces or tools when moving between different pages, enhancing overall usability.

Navbar always present at the top, white-blue-grey colour scheme

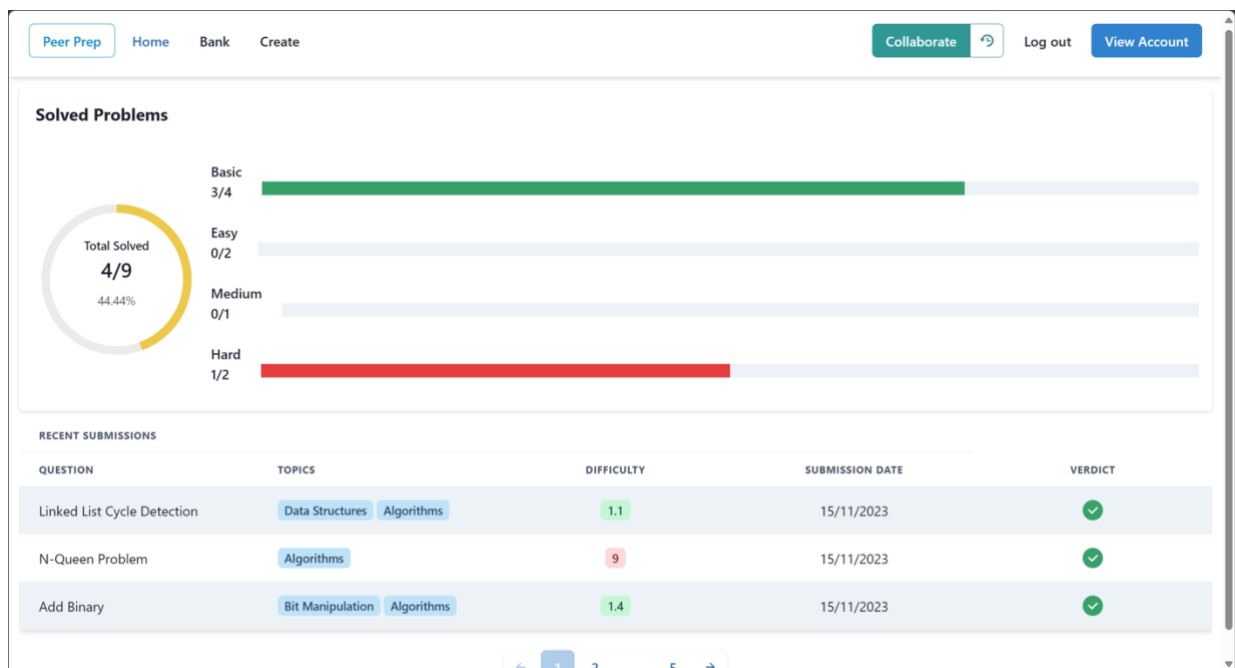


Fig 7. Theme consistent at Home Page

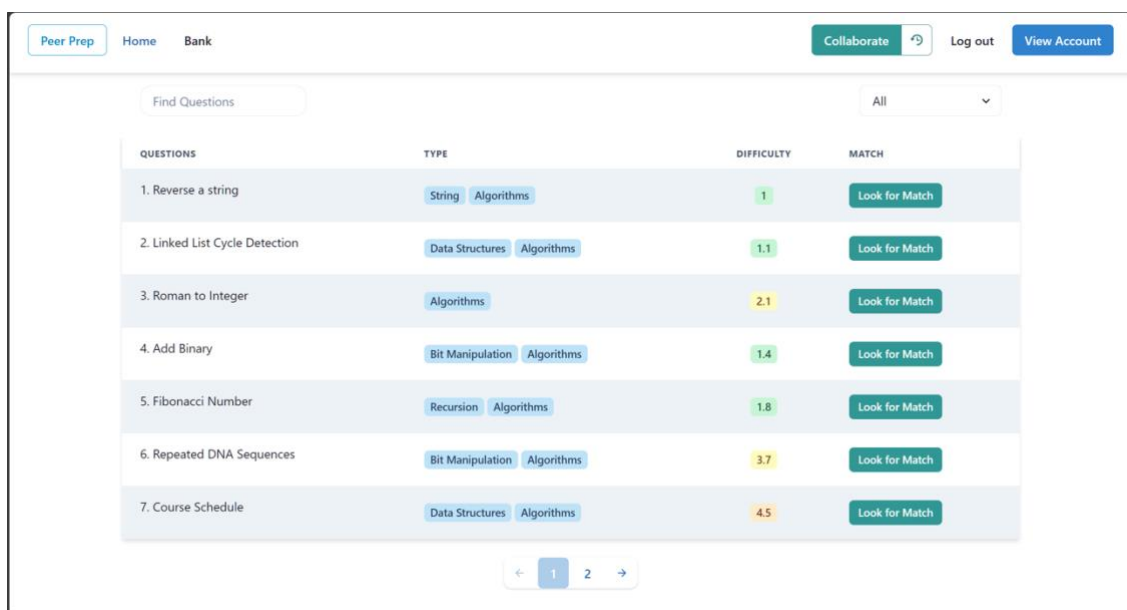


Fig 8. Theme consistent at Bank Page

### 5.2.5 Choice of clear language

Throughout PeerPrep, we've maintained adherence to established design standards and widely recognized conventions. We've utilized universally understood language across the application to represent various functions. For instance, we use the 'View Account' button in the navigation bar to symbolize user account profile.

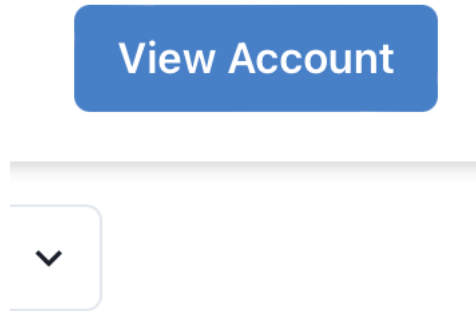


Fig 9. View Account button

A chat icon for messaging, and a disconnect icon to denote session termination

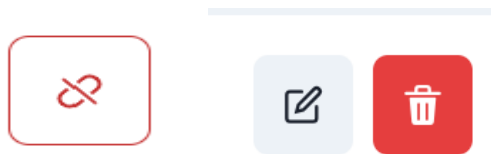


Fig.10 Disconnect icon

These icons serve to capture user attention and clarify the functionalities available. By leveraging iconography and clear language, we've aimed to reduce ambiguity in function representation, facilitating users in easily locating and utilizing various features. This approach minimizes confusion during application usage, enabling users to seamlessly perform tasks like managing settings or leaving a room.

## 5.3 Design Patterns

### 5.3.1 Model View Controller Pattern (MVC)

In certain services like user service and matching service, the main functionalities involve data storage, retrieval of data, displaying of data to the user, responding to users' actions. We adopt the MVC pattern for these services.

Our decisions in adopting the MVC pattern are as follows:

Reason	Justification
High modularity and reusability	MVC follows the separation of concern principle.
Accelerate development as model, view, and controller could be developed in parallel	<p>As there is low coupling and high cohesion between model, view, and controller, each parts have little dependency to each other.</p> <p>This allows for parallel development as there is little coupling between components, allowing for a faster development process.</p>
Enhance UI modification capabilities	View is not highly coupled with model and controller, therefore changes in the view components are less likely to require changes in the model and controller.

The following is an example of how the MVC pattern is being applied in our application.

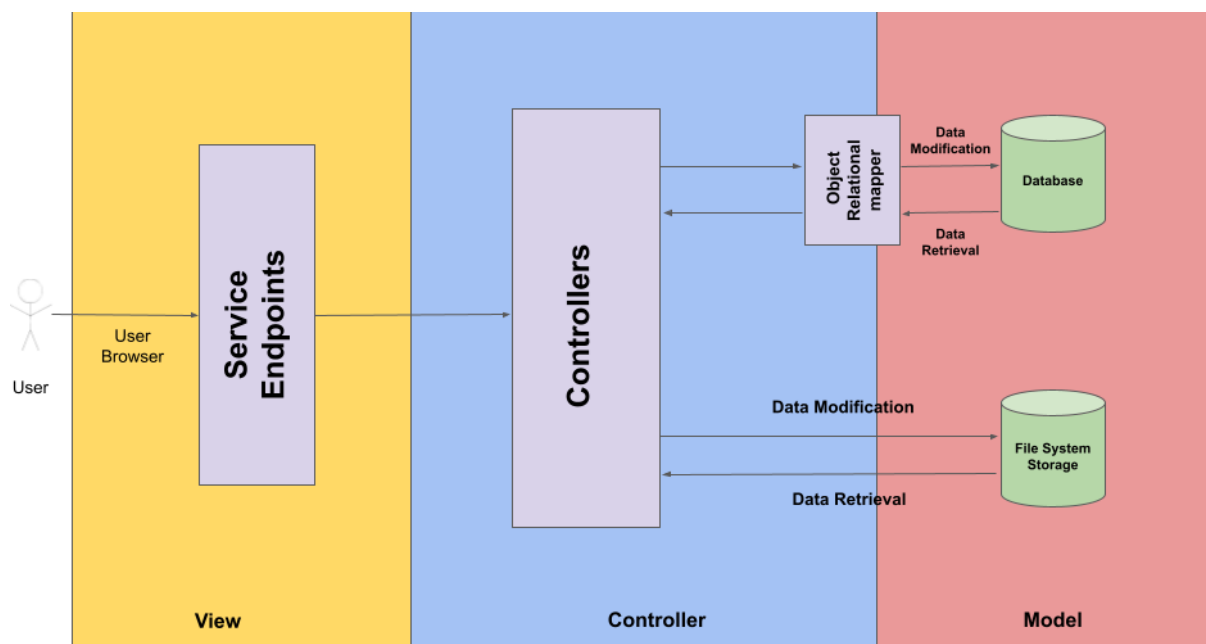


Fig 11. High level view of MVC in User Service

## 5.4 Microservices Design

Our application comprises different microservices to provide different functionalities. Some of these microservices are hosted in-house, while others have some implementations that rely on external third-party APIs. We outline the current arrangement in the table below:

Microservice	In-House/Third-Party	External Service Relied On
Matching service	In-House	NA
Question service	In-House	NA
User service	In-House	NA
Execution service	In-House	NA
Collaboration service (Real-time peer programming)	Third-Party	Public Stun/Turn Servers

## 5.5 Backend Microservices

Technologies	Rationale
Node.js	<ul style="list-style-type: none"><li>• High speed and scalability due to single threaded as well as fast concurrency support with their asynchronous implementation</li></ul>
Express.js	<ul style="list-style-type: none"><li>• The event-driven architecture and non-blocking runtime environment allow for process of multiple requests at the same time</li><li>• Fast data sync is possible</li><li>• Using node to create lightweight backend model for asynchronous data-loading via callback functions with react for Single Page Application (SPA)</li><li>• A lot of community support</li></ul>
Socket.IO	<ul style="list-style-type: none"><li>• Duplex communication between users</li><li>• Allow users to receive event driven messages from the Server</li><li>• A lot of powerful functions abstracted by the libraries</li></ul>

### 5.5.1 User Service

The user service allows the system to identify between users. It allows for the creation, retrieval, and updating of user profiles, which contain information such about users such as their username, their role and their profile pictures and bio and their attempted questions.

Besides enabling distinct user profiles, the user service handles authentication.

#### 5.5.1.1 Service Boundary Considerations Between Authentication and User Service

An architecture where the User service and Authentication service are each separated was considered. The user service would be focused on managing user information and behaviours not directly related to the security aspects, such as managing one's profile page, while the authentication service would exclusively be concerned with verifying identity, managing credentials as well as enforcing role-based access control for other API endpoints.

This design would adhere more strongly to SRP and would allow for the authentication service to become a centralized and secure gateway for user verification. However, given that user and authentication data are closely related, these two services would likely need to be developed and deployed simultaneously. Combined with the need to manage inter-service communication, the coordination of these services would slow down development process and deemed a suboptimal design. Therefore, the final design merges these two components into a single service, but still employs a clear separation between the functionalities of these two services.

#### 5.5.1.2 Core features

The following table defines the core features that fall under each service boundary.

User information and behaviour related	Authentication Related
<ul style="list-style-type: none"><li>- Retrieval of user profile (includes name, role, bio, profile picture)</li><li>- Searching of other users</li><li>- Uploading and retrieval of profile pictures</li><li>- Update of bio</li><li>- Insertion and Retrieval of questions attempted by user</li></ul>	<ul style="list-style-type: none"><li>- Registration</li><li>- Login</li><li>- Logout</li><li>- Change password</li><li>- Retrieval of session user</li><li>- Deletion of account</li><li>- Updating of role</li></ul>

#### 5.5.1.3 Tech stack

It utilizes the following technologies to enable its services

	Technologies	Rationale
Database/ Object Relational Mapper	PostgreSQL, Prisma	<ul style="list-style-type: none"><li>• Prisma provides robust schema and query language for PostgreSQL, allowing to define schema in type-safe way</li><li>• Supports database migration making it easier to evolve database in the future</li></ul>
Authentication	JWT	<ul style="list-style-type: none"><li>• Provides token-based authentication mechanism</li><li>• Always validated incoming JWTs on server-side to ensure authenticity, integrity and ensure the token is not expired</li></ul>

#### 5.5.1.4 Design

The user service is mainly a create-read-update-delete (CRUD) service and adopts the MVC high-level architecture for most of its features.

#### 5.5.1.5 User Authentication

User authentication is done through issuing the user with a JWT token encapsulated as a http-only cookie. The JWT token contains payload information about a user's unique ID, as well as their user role.

When the user logs out, the server invalidates the JWT token by removing the http-only cookie. The following sequence diagram depicts the authentication process involved in login and log out of the user.

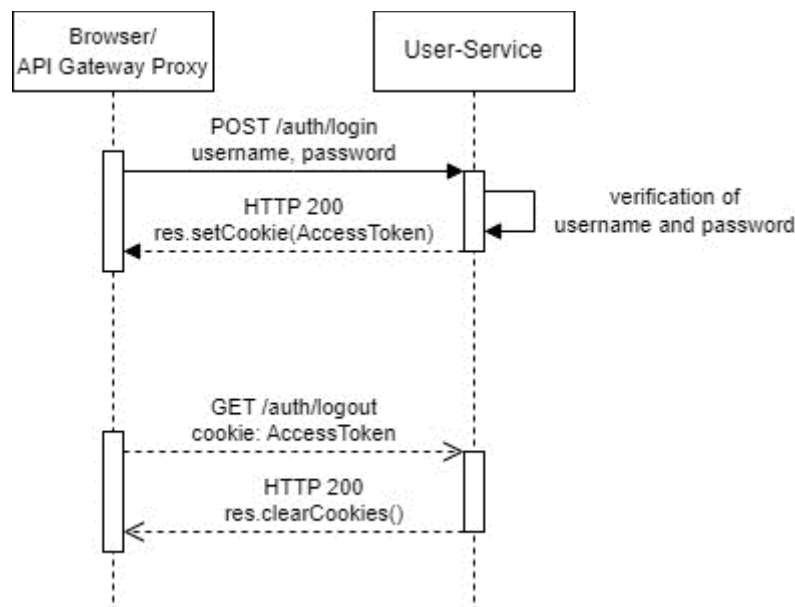


Fig 12. User Authentication

#### 5.5.1.6 Persisting user session

The user can also persist their session and remain logged in if they refresh their browser on the frontend. As http-only cookies can only be cleared by the server, the cookie is persisted upon refresh. Upon refresh of the frontend application, the application makes an initial call to the user service. If the cookie still exists, then the user responds with the necessary user details to set the user in a logged-in state on the frontend.

#### 5.5.1.7 User Profile Pictures

To enable users to have a distinct visual identity on the platform when they collaborate with others, the user service also allows users to store their profile pictures. The images are stored in a persistent volume on the user service instead of in the database as assets such as pictures may not be well suited to be stored in PostgreSQL.

When users upload their profile pictures, the service replaces their current profile picture with the new one and updates the user object in the database to point to filename of the new profile picture. These pictures are served through a static link whenever the frontend requests for it.

### 5.5.2 Question Service

The question service deals with the creation, retrieval, update, and deletion of questions. It allows the system to have a repository of questions for the user to work on. Beyond that, it is also responsible for uploading a question's input files and expected output meant for the execution service.

The question service utilizes the same MVC architecture as the user service, for its clean separation of concerns between model, controller, and view. MVC architecture also promotes **modifiability** and **maintainability**, key quality attributes that we chose to prioritize for PeerPrep. We see this in action in question service, as the use of the MVC architecture allows us to easily use a different storage technology to store our test cases, as elaborated in [5.5.2.2](#) below.

#### 5.5.2.1 Tech Stack

The following technologies were used when designing the question service.

- MongoDB – NoSQL Database to store data about each question
- Mongoose – Object Relational Mapper
- Express – Server provider

The question service was initially designed after the SE-Toolbox provided. We settled on MongoDB initially as we thought that question data was an unstructured piece of data and would fit well into a NoSQL database such as MongoDB. Mongoose was used as the ORM. However, when developing the user service, Prisma proved to be a much more effective ORM than mongoose due to the type hints and stronger type safety features it provides during development.



#### 5.5.2.2 Storing of testcases

In addition to storing the question data, there was a subsequent enhancement to the question service which involved storing testcases for each individual question. These testcases can be uploaded by admin users via a zip file containing `.in` and `.out` files and will be unzipped safely to a shared volume, under a directory named after the question's UUID. The execution service can then access this shared volume to read a question's input and output files for execution.

The developer acknowledges that storing the testcases in a shared volume rather than MongoDB might be a suboptimal design, considering that MongoDB was already being used to store question data. Furthermore, the benefits when it comes to using a database to manage our stored files might be more apparent when the application is further developed and scaled up in the future.

However, the choice of storing the testcases on a shared volume was a strategic decision driven by the need for efficient file management and straightforward integration with existing services. By using shared volumes, we can make the testcases immediately available to our execution service without any additional development overhead to setup MongoDB on the execution service. Shared volumes also integrate seamlessly with container orchestration systems such as Docker-Compose, enhancing deployment flexibility. In essence, the choice of using shared volumes was simply using the right tool at the right time, optimizing for operational efficiency and simplicity.

### 5.5.3 Matching Service

#### 5.5.3.1 Tech Stack

The matching service

	Technologies	Rationale
Communication	Socket.io	<ul style="list-style-type: none"><li>• Enables real-time bidirectional communication between client and servers</li><li>• Uses WebSockets as primary protocol for low-latency communication</li><li>• Facilitates event-based communication. Clients and servers can emit and listen for events, allowing for more organised communication</li><li>• A lot of powerful functions abstracted by the libraries</li></ul>

In our matching service, we have adopted the MVC (Model-View-Controller) design pattern to provide a well-structured framework for user interaction through the shared frontend user interface. The model component serves as a temporary data storage, specifically for users awaiting matches, with unique identification based on either their assigned Socket ID or registered username. This model implements queue logic, where users are stored temporarily based on their chosen range of difficulty levels. When a match is found, the respective user is promptly removed from the queue. In cases where a user is not matched within 30 seconds, the client notifies the server, triggering actions to remove the user from the queue. Additionally, if a user cancels the matching process before the 30-second mark, their data is swiftly deleted from the database. This implementation ensures an organized and efficient matchmaking process with clear separation of concerns, enhancing the overall user experience.

## Server structure

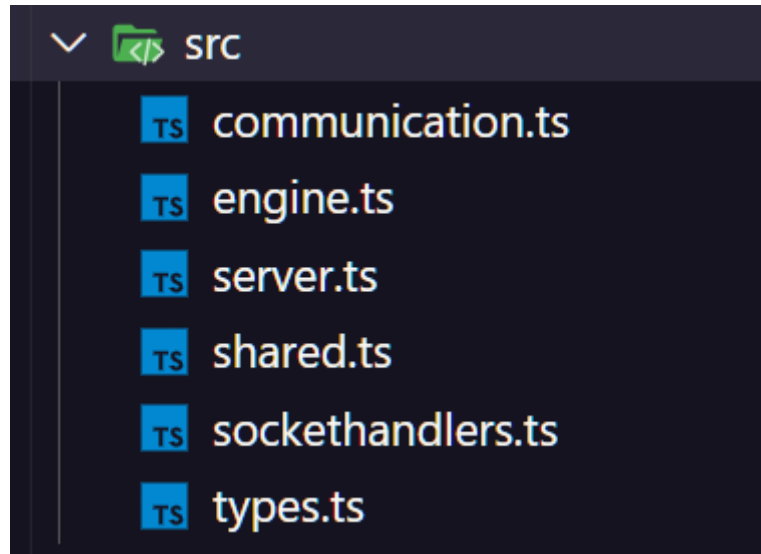


Fig 13. Server Structure

The duty of each file:

### 5.5.3.1.1 server.ts:

This Express.js server script, using Socket.IO for real-time communication, manages user interactions within collaborative coding sessions. The script also manages user disconnections, removing their socket entry. The server runs internally on port 8082, facilitating seamless collaboration between connected users.

### 5.5.3.1.2 shared.ts:

This TypeScript module exports utility functions and data structures for managing WebSocket connections. This module plays a crucial role in maintaining the mapping between users, their sockets, and enabling seamless communication in the collaborative coding environment.

### 5.5.3.1.3 types.ts:

These interfaces provide a clear structure for the data exchanged within the collaborative coding system.

### 5.5.3.1.4 communication.ts:

Allows service to communicate with questions service in the backend to fetch desired questions that likes in the range chosen in a match

### 5.5.3.1.5 engine.ts:

Exposed functions that abstracts the underlying matching data structure which is an interval queue. Interval queue allows us to match different ranges of result in  $O(\log n)$  time.

The following is an activity diagram that captures the logical flow in which the match events are executed.

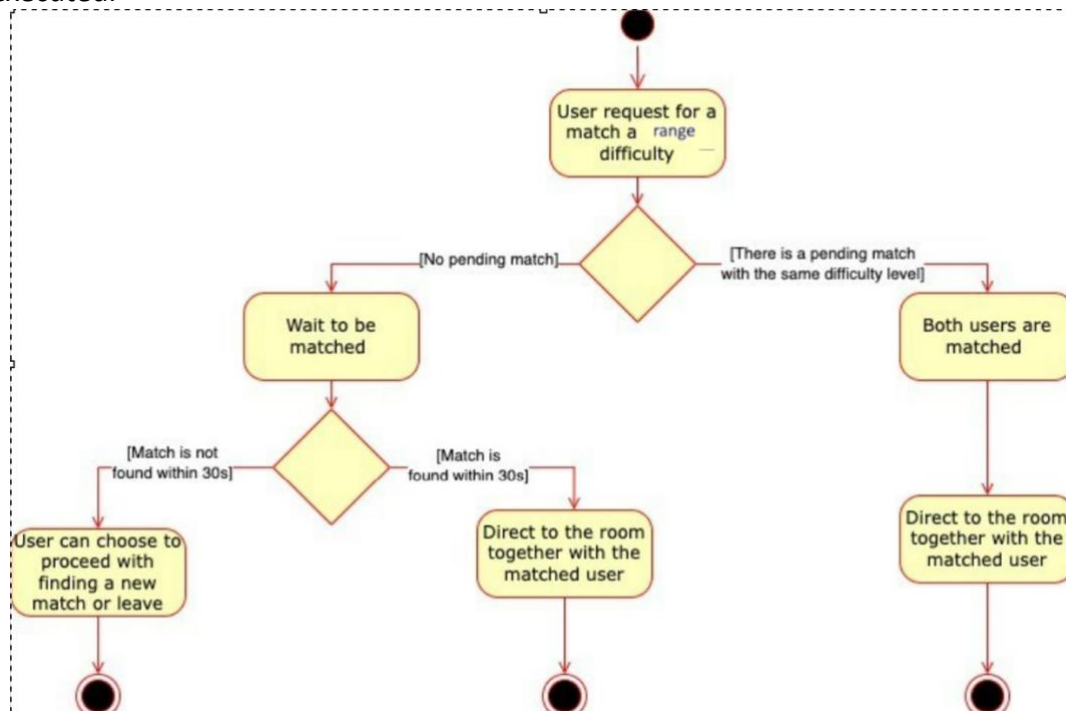


Fig 14. Matching service

### 5.5.3.2 Leaving a room

When a user leaves the room in the collaborative coding system by clicking on the disconnect icon, the following actions are taken:

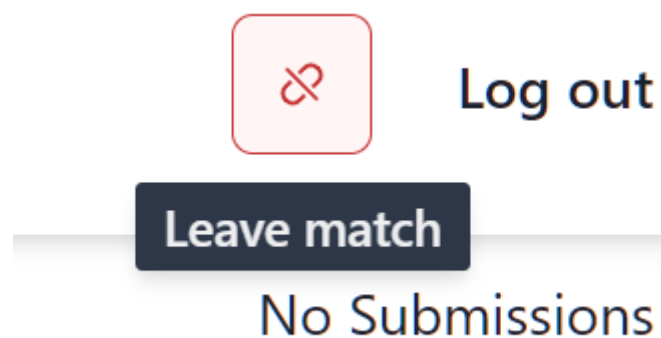


Fig 15. Logout

### 1.Event Emission / immediate effect:

- The user initiates the leave action voluntarily.
- The server receives the "quitRoom" event from the user.
- The user gets redirected to home page

### 2. Notify Matched User:

- If the user is currently matched with another user in a collaborative session, the system identifies the matched user.
- "matchEnded" event is emitted to notify the matched user that their partner has left the room. This causes the matched user to transition to single player mode to allow him/her to continue working on the question.

### 3. Remove from Match Service:

- ❑ The user is removed from the match queue, ensuring that they are no longer considered for new matches as well as clearing the matching service to allow them to be queued in the future to match again.

### 4. Cleanup:

- The user's data is removed from relevant data structures, such as the mapping of user IDs to sockets.
- The user's socket is disconnected.

By following this process, the system ensures that the departure of a user from a collaborative session is handled gracefully. The matched user is notified, and necessary data structures are updated to reflect the user's departure. This approach helps maintain the integrity of the collaborative coding experience for both users involved in the session.

#### 5.5.3.3 Disconnection from a room

When a user leaves the room in the collaborative coding system involuntarily (by navigating to another page, refresh, log out, internet disconnection)

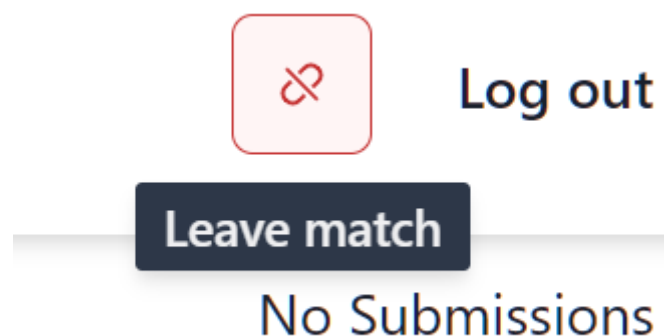


Fig 16. Leave room

### 1.Event Emission:

- The user initiates the leave action by accident
- The server receives the "disconnect" event from the user. Since this is an automatic event upon disconnect, the system will identify this as accidental disconnect since "quitRoom" event was not received by the server.

### 2. Notify Matched User:

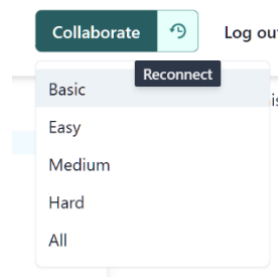
- If the user is currently matched with another user in a collaborative session, the system identifies the matched user.
- A "matchEnded" event is emitted to notify the matched user that their partner has left the room accidentally with a payload indicating that the partner can rejoin the session. At the same time, the remaining user is now promoted to master of the pair to allow them to submit/continue work on his/her work.

### 3. Begin timeout from **Match Service**:

- The user that is accidentally left is now placed in a timeout where should the user not join back during this period, both users will be removed from the queue and clean up step from 5.5.3.2 will occur for both users.
- Should the user that remains also leave during this period (whether voluntarily or disconnect), both users will be removed from the queue and the time out will be cancelled and the queue is cleaned.

The above step will end similarly the clean-up step and onwards as described in 5.5.3.2 for both users

### 4. Disconnected user rejoins via reconnect button within the timeout



The user will retrieve his/her match information from the datastructure and restore his/her connection to the room and clears the timeout responsible for kicking the user.

By following this process, the system ensures that accidental departure of a user from a session can be restored reliability.

#### 5.5.3.4 Cancelling a match

Users can cancel a match by clicking the red cancel button at the top right corner. If not, the user will automatically be removed from matchmaking after 30 seconds of unsuccessful matching. The timer below shows how long the user has left in the queue before being automatically removed. When a user cancels a match in the collaborative coding system, the following actions take place:

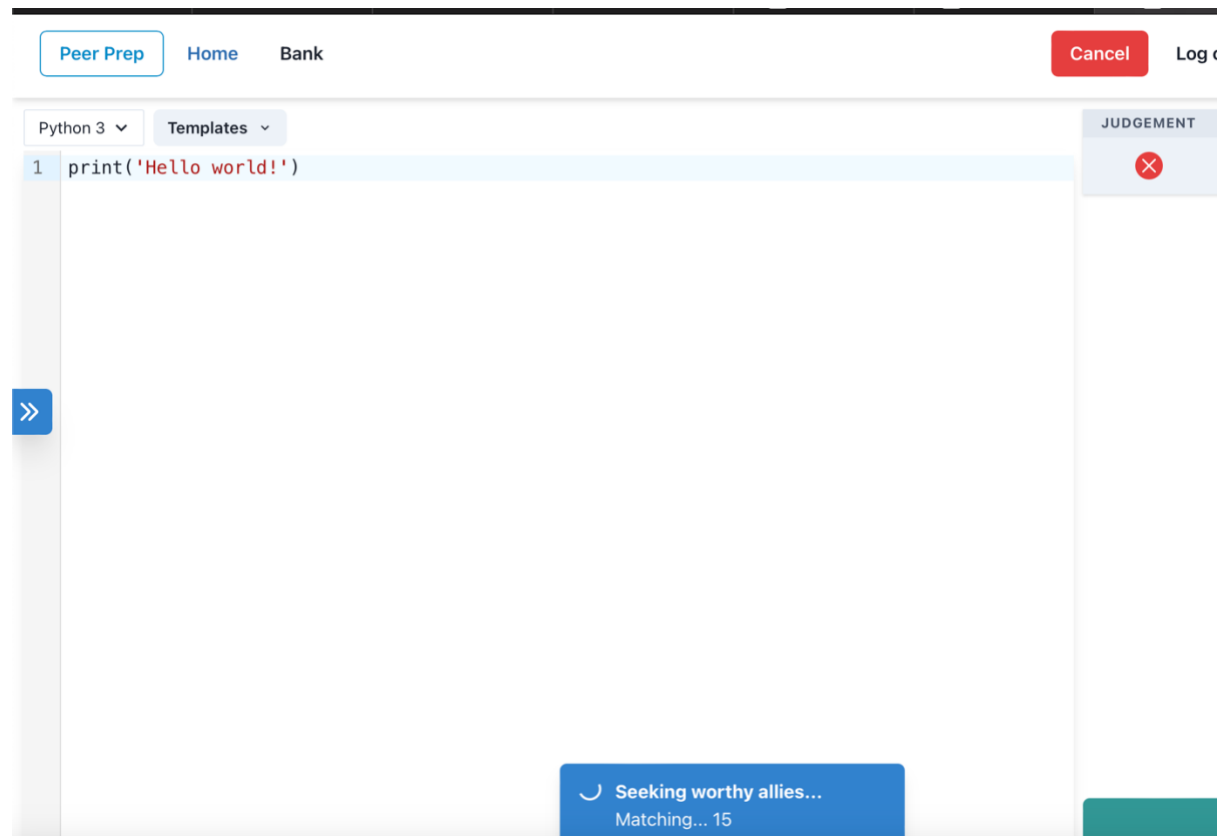


Fig 17. Collab room

##### 1. Event Emission:

- The user initiates the cancel action, expressing the desire to cancel the matchmaking process.
- The server receives the "cancelMatch" event from the user.

##### 2. Remove from Match Queue:

- The user is removed from the matchmaking queue, ensuring that they are no longer considered for matching.

##### 3. Possible Scenarios

- This user was paired with some other user in this process (but yet to find a suitable question)
  - The other user will then requeue into the current matching engine.
- This user was not paired (go direct to cleanup)

### 3. Cleanup:

- The user's data is removed from relevant data structures, such as the mapping of user IDs to matching information.

### 4. Disconnect Socket:

- The user's socket is disconnected.

By following this process, the system ensures that the user's request to cancel the matchmaking is handled effectively. The user is removed from the matchmaking queue, and any ongoing matchmaking processes are halted while still allowing other users to continue their queueing process. This approach helps maintain the flexibility for users to control their participation in the collaborative coding system.

#### 5.5.3.5 Room allocation

The allocation of rooms in the collaborative coding system typically follows a process outlined by the application's logic. Here's a general overview of how room allocation might occur:

#### 1. Match Found:

- When the matchmaking service identifies two users who are suitable matches for collaboration, it triggers a "match found" event.

#### 2. Room Creation:

- A unique identifier for a collaborative coding session (room) is generated. This identifier is a combination of user IDs and a unique identifier for the coding question.

#### 3. Inform Users via event "matchFound":

- Each user involved in the match is informed about the match, providing details such as the partner's ID, the question ID, and the room identifier as well as who to initiate the initial code and who is the master of the 2 users.
- The information is sent to the users via their respective sockets that represents them.  
(A user can have multiple connections thus each socket (once identified to be user XXX) will be connected to a room by their username on server (using socket io). This allow us to easily broadcast to all sockets from this user.)
- These users will then be redirected to the view question page.

#### 4. Room Initialization:

- The collaborative coding room is initialized with a random language with their default templates by the socket that is assigned to initialize the code.
- Reinit event are propagated to users who were unable to connect to the session reliably from the frontend to ensure do disconnects when using the service.

#### 5. Socket Room Assignment:

- Both users' sockets are assigned to the same room, allowing them to communicate and collaborate in real-time in a peer-to-peer manner.



## 7. Collaboration Begins:

- The users can now collaborate in the allocated room, with real-time code editing and communication features enabled.

By following this process, the system ensures that users who have been successfully matched can seamlessly enter a shared collaborative coding environment. The unique room identifier allows the system to manage and facilitate the collaboration between the matched users effectively.

### 5.5.4 Execution Service

The code execution service orchestrates the interaction between the user's code and the Judge0 API. Users write their code within the platform's editor, and upon submission, the master user will transmit the code to the service. The service will then fetch a series of test cases, and expected outcomes for that question send to Judge0 for execution.

Once processed, the service retrieves and delivers the execution results, determining the pass or fail status based on the code's performance against the provided test cases. This crucial functionality ensures seamless code evaluation and verification within the platform, aiding users in assessing the accuracy and efficiency of their solutions.

#### 5.5.4.1 Tech Stack

	Technologies	Rationale
Backend	Judge0 Server	<ul style="list-style-type: none"><li>- Provides detailed error handling information</li><li>- Easy to execute when testing code with various inputs</li></ul>
Backend	Express.js	<ul style="list-style-type: none"><li>- Express's middleware facilitates detailed error handling, providing clear error messages for debugging.</li><li>- Express simplifies routing and request handling, making it easy to process diverse input types efficiently.</li><li>- Its modular design allows breaking down the application into reusable components, aiding scalability and maintainability as the codebase grows.</li><li>- Leveraging its popularity, Express benefits from a large community and a rich ecosystem of libraries and plugins for support and extensions.</li><li>- Express seamlessly integrates with various third-party modules and APIs, enabling compatibility with the Judge0 Server for code execution and result retrieval.</li></ul>
Backend	Http callbacks	<ul style="list-style-type: none"><li>- As there are not a lot of communication between front end and the code execution, http callback is used to communicate between the front and backend. To red</li></ul>

#### 5.5.4.2 Design

The code execution service is built as an Express-based server, handling code submissions, result queries, and submission deletions. It relies on the Judge0 API for code execution and result retrieval.

#### Service Components and Responsibilities:

1. **Server-Side Logic (Express App):**
  - **API Endpoints:** Exposes RESTful endpoints for code submissions, result queries, and deletion requests.
  - **Error Handling:** Implements error handling to gracefully handle errors and provide informative responses to client requests.
2. **Submission Processing:**
  - **Submission Execution:** Orchestrates code execution by interfacing with the Judge0 API. Handles both individual and batch submissions for code execution.
  - **Polling Mechanism:** Utilizes polling to monitor submission statuses continuously until completion.
3. **Judge0 API Integration:**
  - **Submission and Result Handling:** Interacts with the Judge0 API for submitting code, retrieving status updates, and deleting submissions once completed.
  - **Verdict Determination:** Evaluates the results obtained from the Judge0 API to determine the verdict for each submission.
4. **File Handling (Test Cases):**
  - **Test Case Retrieval:** Fetches test case files (input and output) from a shared volume based on the provided question ID (`qn\_\_id`).
  - **Batch Submission Creation:** Generates batch submissions based on retrieved test case files and the code to be executed.

#### Execution Flow:

1. **Submission Initiation:**
  - Clients submit code, language, question ID (`qn\_\_id`), and user ID (`uid`) to `/api/code/submit` endpoint.
  - The server calls the Judge0 API with the provided code and necessary parameters to initiate the execution process (`runSubmission` function).
2. **Submission Execution:**
  - Generates individual or batch submissions based on the code and associated test cases (`getQnStdInOut`).
  - Executes code submissions asynchronously using the Judge0 API (`submitIndividual` and `submitCode`).
  - Monitors submission statuses through polling mechanisms (`pollIndividualTillComplete` and `pollStatusTillComplete`).

3. Result Retrieval:
  - Clients request submission results through `/api/code/result/:token`` endpoint by providing the associated token.
  - The service retrieves and returns results stored in the ``callbacks`` object based on the provided token.
4. Submission Deletion:
  - Provides an endpoint `/api/code/result/:token`` to delete submissions based on their tokens.
  - Deletes submissions from the Judge0 API using associated tokens (``delSubmissions``).

### **Key Design Considerations:**

1. Asynchronous Handling:
  - Utilizes asynchronous operations and polling for concurrent processing of multiple submissions.
2. Robust Error Handling:
  - Implements error handling mechanisms to manage errors during submission, execution, and result retrieval.
3. Modular Structure:
  - Maintains a modular codebase with separated responsibilities for enhanced readability and maintainability.
4. Judge0 API Integration:
  - Extensively interacts with the Judge0 API for submission, monitoring, and result retrieval.

### **Scalability and Future Enhancements:**

1. Scaling Capabilities:
  - Designed for handling high loads by concurrently processing multiple submissions.
2. Language Expansion:
  - Allows easy inclusion of additional programming languages by mapping them to corresponding Judge0 language IDs.
3. Result Persistence:
  - Potential integration with a database to store submission results for historical tracking and analysis.

The code execution service demonstrates a structured architecture that manages code submissions, execution, and result retrieval efficiently. Its design prioritizes scalability, extensibility, and robustness, making it adaptable for handling diverse workloads and accommodating future improvements or enhancements.

### 5.5.5 API Gateway

The API Gateway serves as a centralized gateway for all requests to pass through by proxying all our requests to the correct microservice. To do so, it utilizes http-proxy-middleware. It also enables authentication on the correct routes by implementing jwt checks before proxying.

#### 5.5.5.1 Choice of Tech Stack

The choice of using Express and http-proxy-middleware as opposed to common gateway technologies such as NGINX was driven by the need to deploy as quickly as possible.

Owing to the learning curve of NGINX, we settled on using a familiar technology to build our API gateway. Furthermore, using Express offered more flexibility in terms of introducing custom logic at the gateway level. We utilized this flexibility to introduce JWT token verification at the gateway, to block all unauthorized requests from accessing our internal microservices.

## 6 Future Enhancements

This section details possible enhancements to improve the experience of using PeerPrep.

### 6.1 Features

#### 6.1.1 Video/Voice Chat

To further enhance the utility of PeerPrep to help individuals to prepare for in-person technical interviews, a video and voice function would be a good enhancement to implement. In an in-person interview setting, non-verbal cues like eye contact, body posture, and facial expressions play a crucial role in conveying one's confidence in their abilities and expressing genuine interest in the company. Therefore, incorporating a video chat feature allows individuals to engage in mock interviews with peers, in addition to facilitating whiteboard-style technical interviews. This feature proves highly beneficial for honing interview skills and preparing for technical assessments.

#### 6.1.2 Login using Github

To minimise the trouble of having to remember another username and password, we can integrate with external providers, such as GitHub authentication.

#### 6.1.3 Gamification and Achievements

Enhancing user engagement and boosting retention on PeerPrep can be achieved by incorporating gamification elements. For instance, we can introduce achievements for completing a specific number of questions or engaging in a set number of matches, surpassing a defined time limit. The latter ensures that users genuinely collaborate with their peers on questions and discourages attempts to exploit the system for achievements. Earning achievements could grant users experience points, allowing them to level up. Additionally, badges corresponding to the attained achievements could be awarded.

### 6.2 Deployment

Our strategy revolves around deploying the microservices and frontend application on a reliable and scalable cloud service provider. By opting for such a platform, we're poised to efficiently scale up the microservices and frontend application to adeptly accommodate a growing user base. This strategic move aligns with our vision to proactively manage and handle increased traffic and usage of the application in the foreseeable future. Furthermore, leveraging a robust cloud infrastructure not only ensures the current performance and reliability of our services but also sets the foundation for future enhancements and expansion to meet evolving user demands and market needs.

## 6.3 Cache

We've identified that authentication is among the most frequent operations in the application, crucial for maintaining continuous protection and blocking unauthorized access. However, when numerous users are active simultaneously, it can lead to slow response times, extended loading periods, and a subpar user experience. To enhance performance, the middleware necessary for authentication can be cached. For instance, we can store the blacklist of Access Tokens and whitelist of Refresh Tokens in a Redis Cache, ensuring the security of the user's account data in the database. Caching these lists in-memory also prevents users from being forcibly logged out in the event of a user-service server crash or restart.

## 7 Learning Points

Before delving into this module, our team had limited hands-on experience with microservices, CI/CD pipelines, Dockerization, and a myriad of other concepts encapsulated within this domain. Engaging with this project became an eye-opening journey, enriching us with a plethora of insights spanning technical intricacies and non-technical facets. The collaborative effort fostered an environment conducive to learning, allowing us to grasp the intricacies of modern development practices, thereby broadening our understanding of these technologies and methodologies.

### 7.1 Deployments

This project marked the team's initial foray into application deployments, providing insights into various deployment techniques and the associated challenges. We delved into different cloud vendors, experimenting with deployments on diverse solutions and cloud architectures. This exploration allowed us to grasp how distinct toolchains from cloud providers handle scaling and networking, even when deploying the exact same codebase. Although the team didn't delve deeply into specifics, troubleshooting problematic deployments enhanced our understanding of scalable applications, particularly within the microservices architecture. We anticipate that this knowledge will prove valuable in our future endeavours.

### 7.2 Team Processes

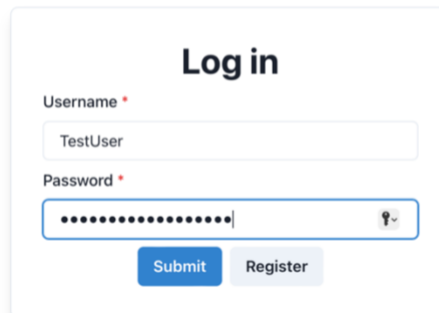
As each team member assumes responsibility for specific components of the application, mirroring the structure of a typical software engineering team, the team gained a more profound understanding of the development processes and challenges. Due to the intricacies of the product, the team encountered hurdles when members overseeing different parts of the system were not in sync on certain issues. This necessitated additional effort to rectify the misaligned components, emphasizing the significance of sprint meetings and open communication—integral pillars of the SCRUM framework.

## 8 Appendix A: Product Screen Shots

This section includes screenshots of the final product, deployed on the deployment environment at <http://peerprep-g10.com/>.

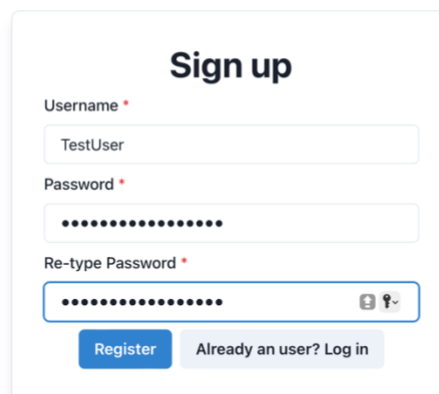
### 8.1 Account pages

#### 8.1.1 Login Page



The screenshot shows a 'Log in' form. It has a title 'Log in' in bold. Below the title are two input fields: 'Username' with a red asterisk and 'Password' with a red asterisk. The 'Username' field contains the text 'TestUser'. The 'Password' field is filled with dots and has a small eye icon to its right. Below the input fields are two buttons: a blue 'Submit' button and a light blue 'Register' button.

#### 8.1.2 Sign Up Page




The screenshot shows a 'Sign up' form. It has a title 'Sign up' in bold. Below the title are three input fields: 'Username' with a red asterisk, 'Password' with a red asterisk, and 'Re-type Password' with a red asterisk. The 'Username' field contains the text 'TestUser'. The 'Password' field is filled with dots. The 'Re-type Password' field is also filled with dots and has a small eye icon to its right. Below the input fields are two buttons: a blue 'Register' button and a light blue 'Already an user? Log in' button.

### 8.1.3 Account profile (User)

[Peer Prep](#) [Home](#) [Bank](#) [Collaborate](#) [Log out](#) [View Account](#)

**Hello TestUser**  
USER  
23 year old programmer from Singapore



Edit Profile

Find users

Find user

Change Password

Change Password

**Solved Problems**

Total Solved  
**5/9**  
55.56%

Basic  
2/4

Easy  
1/2

Medium  
1/1

Hard  
1/2


RECENT SUBMISSIONS

QUESTION	TOPICS	DIFFICULTY	SUBMISSION DATE	VERDI
N-Queen Problem	Algorithms	9	15/11/2023	✓
Course Schedule	Data Structures Algorithms	4.5	15/11/2023	✓

### 8.1.4 Account Profile (Admin)

[Peer Prep](#) [Home](#) [Bank](#) [Create](#) [Collaborate](#) [Log out](#) [View Account](#)

**Hello admin**  
ADMIN



Edit Profile

Find users

Find user

Change Password

Change Password

**Solved Problems**

Total Solved  
**0/9**  
0%

Basic  
0/4

Easy  
0/2

Medium  
0/1

Hard  
0/2

You have no past submissions!

Demote to user role



### 8.1.5 Reset Password

## Change Password

New Password

Re-type Password

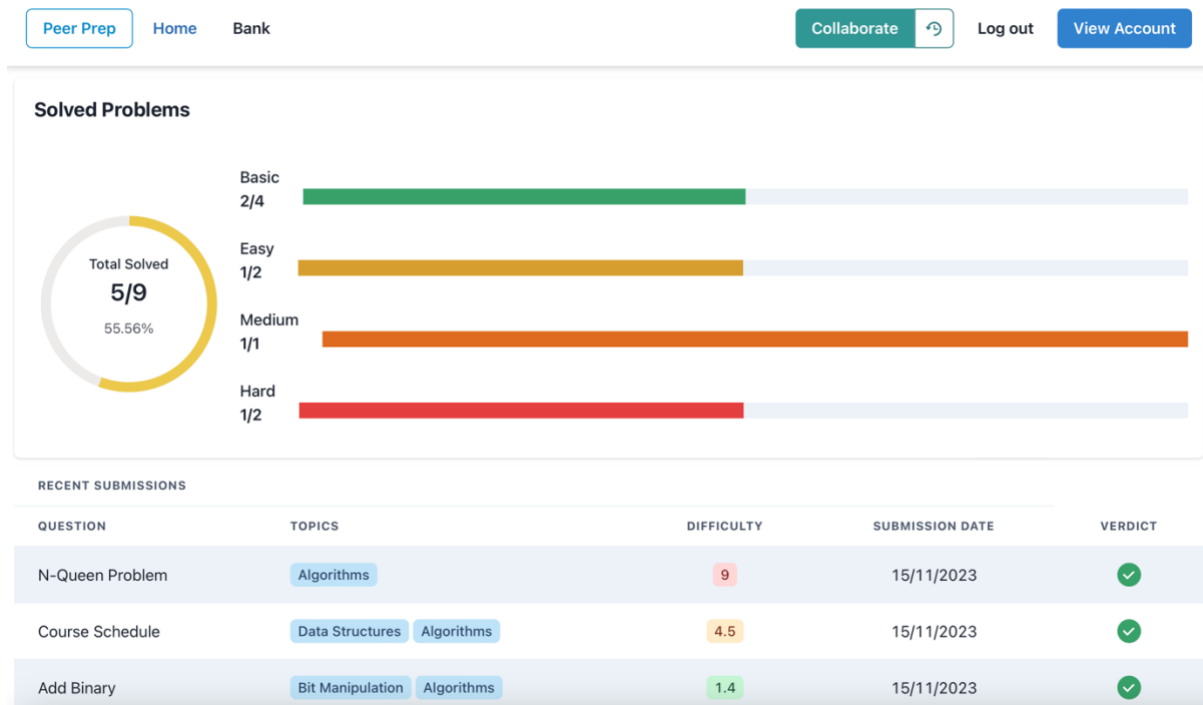
Current Password

Save

Cancel

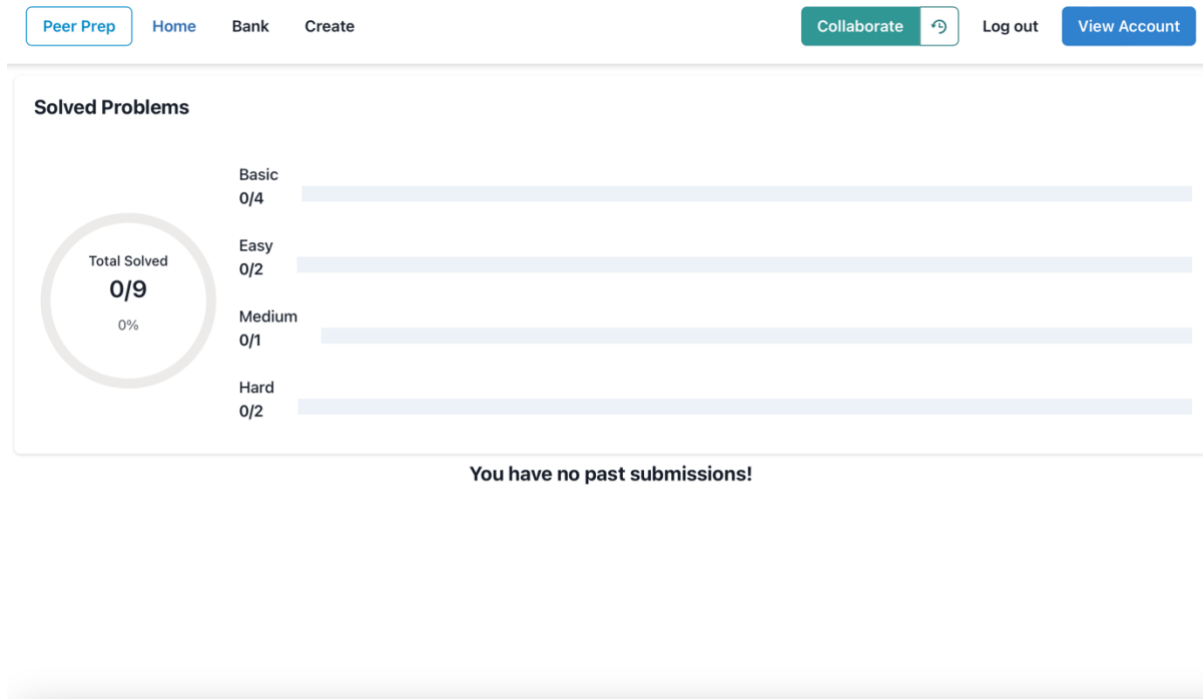
## 8.2 Portal Pages (User)

### 8.2.1 Logged-in Landing Page



## 8.3 Portal Pages (Admin)

### 8.3.1 Logged-in Landing Page



### 8.3.2 Question Bank Page

The screenshot displays the 'Question Bank' page. At the top, there is a navigation bar with links: 'Peer Prep' (highlighted), 'Home', 'Bank', and 'Create'. On the right, there are buttons for 'Collaborate', 'Log out', and 'View Account'. Below the navigation bar, there is a search bar labeled 'Find Questions' and a dropdown menu set to 'All'. The main content is a table with the following columns: 'QUESTIONS', 'TYPE', 'DIFFICULTY', 'MATCH', and 'MODIFY/DELETE'. The table lists seven questions with their respective details.

QUESTIONS	TYPE	DIFFICULTY	MATCH	MODIFY/DELETE
1. Reverse a string	String Algorithms	1	Look for Match	
2. Linked List Cycle Detection	Data Structures Algorithms	1.1	Look for Match	
3. Roman to Integer	Algorithms	2.1	Look for Match	
4. Add Binary	Bit Manipulation Algorithms	1.4	Look for Match	
5. Fibonacci Number	Recursion Algorithms	1.8	Look for Match	
6. Repeated DNA Sequences	Bit Manipulation Algorithms	3.7	Look for Match	
7. Course Schedule	Data Structures Algorithms	4.5	Look for Match	

### 8.3.3 Create Question Page

[Peer Prep](#) [Home](#) [Bank](#) [Create](#)

[Collaborate](#) [Log out](#) [View Account](#)

#### Create Question

Question Name

Difficulty: 0.1

Question Type

Testcases

Please submit a zip file containing all your testcases for this question. The input for the testcases should be suffixed with a `.in`, and the expected outputs with a `.out` For every `xxx.in` file, please include a `xxx.out` file.

Choose File

no file selected

Description

No detail have being provided

### 8.3.4 Edit Question Page

[Peer Prep](#) [Home](#) [Bank](#) [Create](#)

[Collaborate](#) [Log out](#) [View Account](#)

#### Edit Question

Question Name

Reverse a string

Difficulty: 1

Question Type

String

Algorithms

Testcases

Please submit a zip file containing all your testcases for this question. The input for the testcases should be suffixed with a `.in`, and the expected outputs with a `.out` For every `xxx.in` file, please include a `xxx.out` file.

Choose File

no file selected

Description

Write a function that reverses a string. The input string is given as an array of characters `s`.

You must do this by modifying the input array in-place with  $O(1)$  extra memory.

Example 1:

Input: `s = ["h","e","l","l","o"]`

Output: `["o","l","l","e","h"]`

Example 2:

Input: `s = ["H","a","n","n","a","h"]`

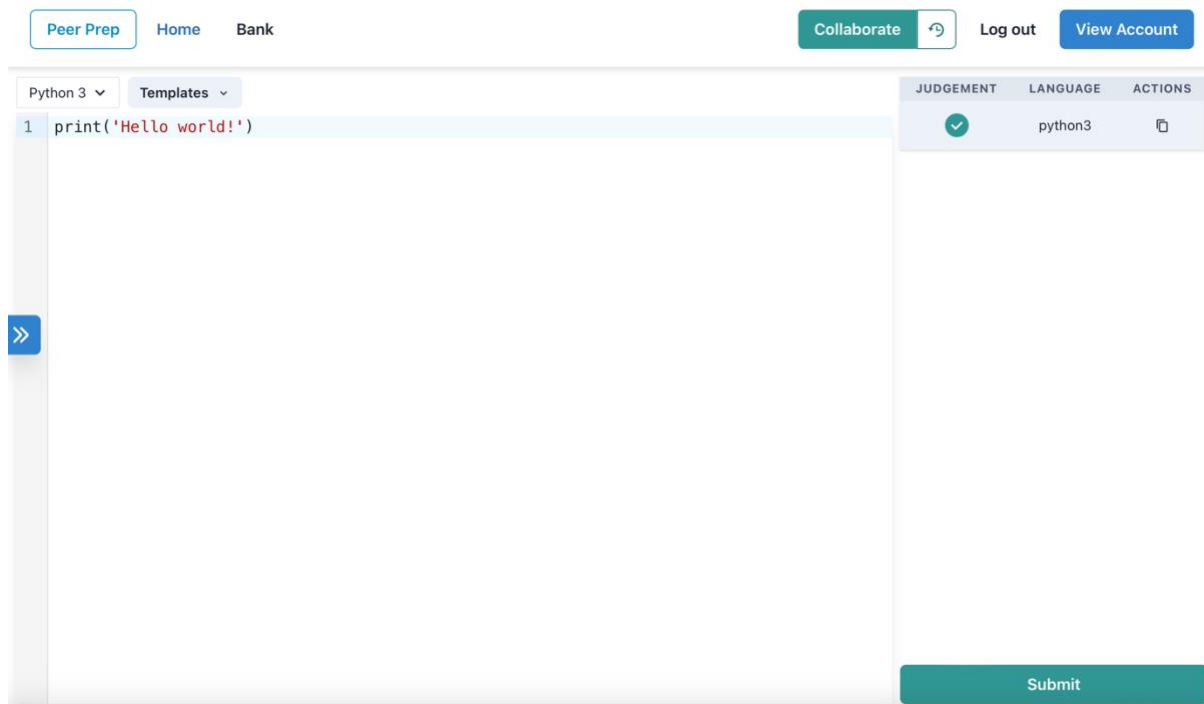
Output: `["h","a","n","n","a","H"]`

Constraints:

$-1 \leq s.length \leq 105$

## 8.4 Code Editor Pages

### 8.4.1 Code Editor Page



### 8.4.2 Code Editor Page with Question Tab

#### 1. Reverse a string

Difficulty: 1 String Algorithms

Write a function that reverses a string. The input string is given as an array of characters `s`.

You must do this by modifying the input array in-place with  $O(1)$  extra memory.

Example 1:

Input: `s = ["h","e","l","l","o"]`

Output: `["o","l","l","e","h"]`

Example 2:

Input: `s = ["H","a","n","n","a","h"]`

Output: `["h","a","n","n","a","H"]`

Constraints:

-  $1 \leq s.length \leq 105$

- `s[i]` is a printable ascii character.



### 8.4.3 Choosing template code for common algorithms

Peer Prep

Home

Bank

Collaborate

Log out

View Account

Python 3

Templates

1 print(''

Quick Select

Fenwick Tree

Segment Tree

Union Find

Floyd Warshall (APAP)

MCBM

Hierholzer

Kruskal (MST)

Prim (MST)

Bellman Ford (SSSP)

Dijkstra (SSSP)

BFS (SSSP)

Toposort

Trie

Max flow / Min Cut

Min Cost Max Flow

JUDGEMENT

LANGUAGE

ACTIONS

✓

python3

Submit

### 8.4.4 Choose the language to code in

Peer Prep

Home

Bank

Collaborate

Log out

View Account

✓ C++ 17

Python 3

Java

Templates

RCE: <https://raw.githubusercontent.com/stevenhalim/cpbook-master/master/ch4/sssp/bfs.cpp>

2 #include <bits/stdc++.h>

3 using namespace std;

4

5 typedef pair<int, int> ii; // In this chapter, we will frequently use these

6 typedef vector<ii> vii; // three data type shortcuts. They may look cryptic

7 typedef vector<int> vi; // but shortcuts are useful in competitive programming

8

9 const int INF = 1e9; // INF = 1B, not 2^31-1 to avoid overflow

10

11 vi p; // addition:parent vector

12

13 void printPath(int u) { // extract info from vi p

14 if (p[u] == -1) { printf("%d", u); return; }

15 printPath(p[u]); // output format: s -> ... -> t

16 printf(" %d", u);

17 }

18

19 int main() {

20 /\*

21 // Graph in Figure 4.3, format: list of unweighted edges

22 // This example shows another form of reading graph input

23 13 16

24 0 1 1 2 2 3 0 4 1 5 2 6 3 7 5 6

25 4 8 8 9 5 10 6 11 7 12 9 10 10 11 11 12

26 \*/

JUDGEMENT

LANGUAGE

ACTIONS

✓

python3

Submit

## 8.5 Matching Process

### 8.5.1 Looking for a match

[Peer Prep](#) [Home](#) [Bank](#) [Cancel](#) [Log out](#) [View Account](#)

All

QUESTIONS	TYPE	DIFFICULTY	MATCH
1. Reverse a string	<a href="#">String</a> <a href="#">Algorithms</a>	1	
2. Linked List Cycle Detection	<a href="#">Data Structures</a> <a href="#">Algorithms</a>	1.1	
3. Roman to Integer	<a href="#">Algorithms</a>	2.1	
4. Add Binary	<a href="#">Bit Manipulation</a> <a href="#">Algorithms</a>	1.4	
5. Fibonacci Number	<a href="#">Recursion</a> <a href="#">Algorithms</a>	1.8	
6. Repeated DNA Sequences	<a href="#">Bit Manipulation</a> <a href="#">Algorithms</a>	3.7	
7. Course Schedule	<a href="#">Data Structures</a> <a href="#">Algorithms</a>	4.5	

Seeking worthy allies...  
Matching... 25

### 8.5.2 Match found

#### 7. Course Schedule

Difficulty: 4.5 [Data Structures](#) [Algorithms](#)

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where prerequisites[i] = [ai, bi] indicates that you must take course bi first if you want to take course ai.

For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1.

Return true if you can finish all courses. Otherwise, return false.

Example 1:

Input: numCourses = 2, prerequisites = [[1,0]]

Output: true

Explanation: There are a total of 2 courses to take. To take course 1 you should have finished course 0. So it is possible.

Example 2:

Input: numCourses = 2, prerequisites = [[1,0],[0,1]]

Output: false

Explanation: There are a total of 2 courses to take. To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.

Constraints:

- 1 <= numCourses <= 2000
- 0 <= prerequisites.length <= 5000
- prerequisites[i].length == 2
- 0 <= ai, bi < numCourses
- All the pairs prerequisites[i] are unique.

Match found  
Redirecting...

[Log out](#) [View Account](#)

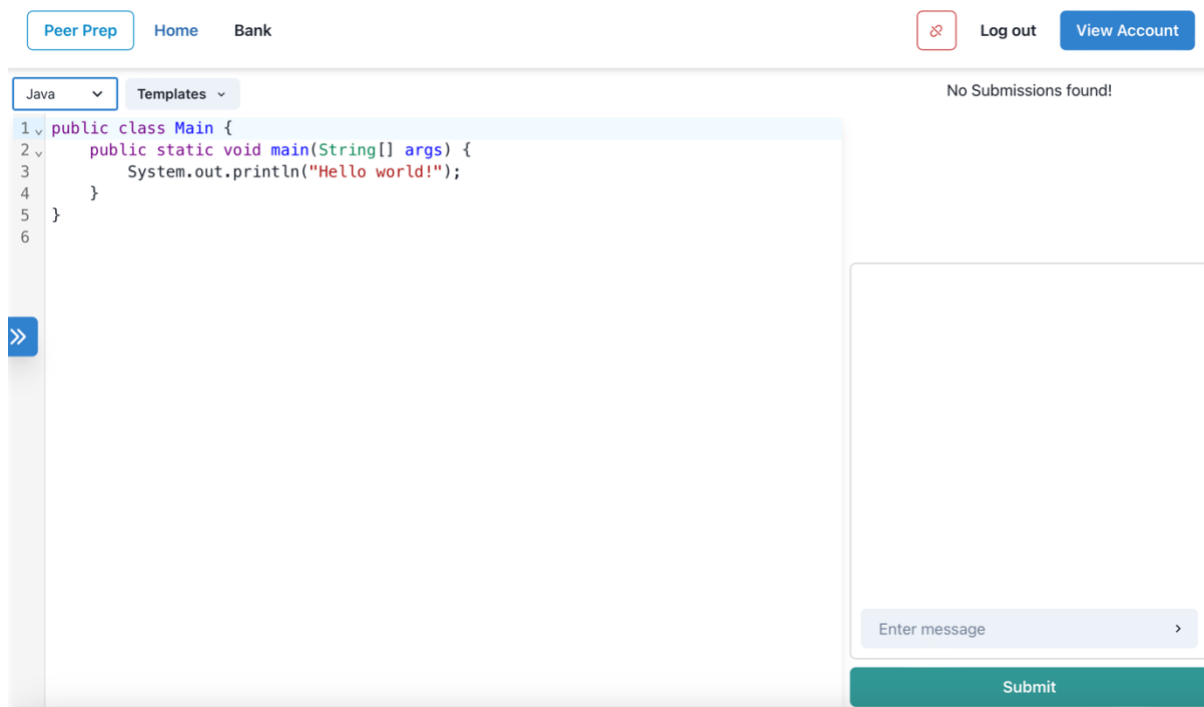
No Submissions found!

Enter message >

Submit

## 8.6 Collaborative Coding Session Page

### 8.6.1 Session Page



### 8.6.2 Session Page with Question Tab

#### 7. Course Schedule

Difficulty: 4.5

Data Structures

Algorithms

There are a total of `numCourses` courses you have to take, labeled from 0 to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [ai, bi]` indicates that you must take course `bi` first if you want to take course `ai`.

For example, the pair `[0, 1]`, indicates that to take course 0 you have to first take course 1.

Return true if you can finish all courses. Otherwise, return false.

Example 1:

Input: `numCourses = 2, prerequisites = [[1,0]]`

Output: true

Explanation: There are a total of 2 courses to take.

To take course 1 you should have finished course 0. So it is possible.

Example 2:

Input: `numCourses = 2, prerequisites = [[1,0],[0,1]]`

Output: false

Explanation: There are a total of 2 courses to take.

To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.

Constraints:

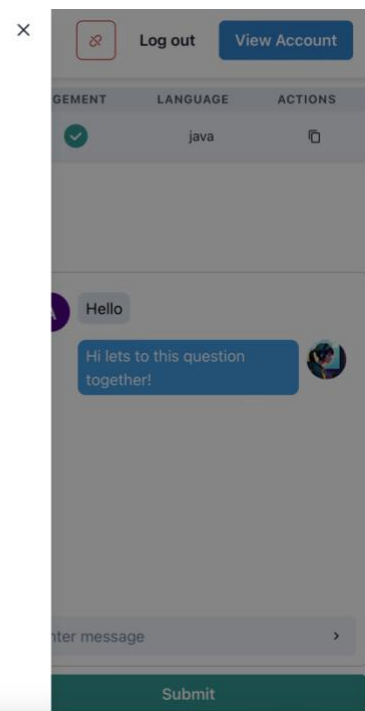
`1 <= numCourses <= 2000`

`0 <= prerequisites.length <= 5000`

`prerequisites[i].length == 2`

`0 <= ai, bi < numCourses`

All the pairs `prerequisites[i]` are unique.





### 8.6.3 Session Page with Chat

[Peer Prep](#) [Home](#) [Bank](#)

Java

Templates

```
1 public class Main {
2     public static void main(String[] args) {
3         System.out.println("Hello world!");
4     }
5 }
6
```

>>

JUDGEMENT

LANGUAGE

ACTIONS

✓

java

📄

A

Hello

Hi lets to this question together!

👤

Enter message

>

Submit

## 9 Appendix B: Use Cases

### Use Case: UC1 - Account Creation

Actor: User

MSS:

1. User navigates to the application webpage.
2. System redirects User to the sign up page.
3. User enters his desired username and password.
4. User confirms sign up.
5. System informs User that account creation is successful.
6. System prompts User to login.

Use case ends.

Extensions:

3b. Desired username is already taken.

3b1. System informs user that username is taken 3b2. User enters a different username.

Use case resumes from step 4.

### Use Case: UC2 – Login

Actor: User

Precondition: User has an existing account. MSS:

1. User navigates to the login page.
2. User enters username and password.
3. System logs user in and redirects User to home page.

Use case ends.

Extensions:

2a. User enters the wrong username or password.

2a1. System informs the user that the username or password is incorrect. Use case resumes from step 2.

### **Use Case: UC3 - Log out Actor: User**

Actor: User

Precondition: User is logged in.

MSS:

1. User chooses to logout.
2. System logs User out and redirects to the login page.

Use case ends.

Extensions:

\*a. User has been inactive for 30 mins.

\*a1. System automatically logs User out. Use case ends.

### **Use case: UC4 - Account Deletion Actor: User**

Precondition: User is logged in.

MSS:

1. User chooses to delete his account.
2. System prompts for confirmation.
3. User confirms.
4. System deletes account and redirects User to login page.

Use case ends.

Extensions:

3a. User decides not to delete the account. 3a1. User chooses to cancel. Use case ends.

### **Use Case: UC5 - Change Password Actor: User**

Precondition: User is logged in.

MSS:

1. User chooses to change his password.
2. System requests User for his current password.
3. User enters his current password.
4. System requests User for his new password.

5. User enters his new password.
6. System requests User to type his new password again.
7. User enters his new password again.
8. User confirms change password.
9. System informs User that the password change is successful.

Use case ends.

Extensions:

5b. User's new password matches his current password.

5b1. System informs User that the new password should not match his current password.

5b2. User enters a different new password.

Use case resumes from step 6.

7a. The second new password does not match the first new password.

7a1. System informs User that the two new passwords do not match. Use case resumes from step 7.

8a. User enters the wrong current password.

8a1. System informs User that the current password is wrong. Use case resumes from step 3.

#### **Use Case: UC6 - Find Match Actor: User**

Precondition: User is logged in.

MSS:

1. User navigates to the find match page.
2. User chooses his desired question difficulty from 'easy', 'medium', 'hard' and 'all' levels.
3. System displays a waiting screen while matching is in progress.
4. System successfully finds a match and redirects User (and matched User) to join the same room (UC7)

Use case ends.

Extensions:

3a. User decides not to continue matchmaking.

3a1. User chooses to cancel the matching.

3a2. System cancels matching and closes the waiting screen. Use case ends.

3b. System could not find a match in 30s.

3b1. System informs User that a match could not be found

Use case ends.

### **Use Case: UC7 - Joins Room**

Actor: Users

Precondition: Users are logged in and matched together.

MSS:

1. Users join the same room.
2. System displays a random question of the chosen difficulty level.
3. Users write code to answer the question in the code editor (UC9).
4. Users chat with each other in the same room (UC10).

Use case ends.

### **Use Case: UC8 - Leave Room**

Actor: User

Precondition: User is logged in and in a room. MSS:

1. User chooses to leave the room.
2. System prompts for confirmation
3. User confirms.
4. System redirects User to the find match page.

Use case ends.

Extensions:

3a. User decides not to leave the room. 3a1. User chooses to cancel.

Use case ends.

### **Use Case: UC9 - Code Collaboration**

Actor: User A and B

Precondition: Users are logged in and in the same room. MSS

1. User A enters a code snippet into the code editor on the room page.
2. System syncs the changes and reflects them onto User B's screen.
3. User B sees where User A is editing.

Use case ends.

Extensions:

\*a. User B can be the one making changes in the code editor.

\*a1. User B enters a code snippet into the code editor on the room page. \*a2. System syncs the changes and reflects them onto User A's screen. \*a3. User A sees where User B is editing.

Use case ends.

## **Use case: UC10 – Chat**

Actor: User A and B

Precondition: Users are logged in and in the same room.

MSS:

1. User A sends a message.
2. User B receives the message.

Use case ends.

Extensions:

\*a. Sender and recipient of messaging can swap.

\*a1. User B sends a message. \*a2. User A receives the message. Use case ends.