CS3219 Software Engineering Principles and Patterns

Milestone 3 Final Report

Team G43

Name	Matriculation Number
Clevon Wong	A0234468X
Lee I-Shiuan	A0180050X
Chang Jing Yan	A0234495X
Tan Wei Xuan Rachel	A0240685B
Ang Min Cher, Branda	A0237759L

Table of Contents

Github Repository

https://github.com/CS3219-AY2324S1/ay2324s1-course-assessment-g43

Contributions

Name	Sub-Group (Features)	Technical Contributions	Non-Technical Contributions
Clevon Wong Jun Yong	M2: User Service M4: Collaboration Service M6: Deployment N4: Enhance question service N9: Deployment N11: API Gateway	Clevon set up the React app with Vite, ChakraUI and MobX. Implemented frontend codebase foundation and a Modal Provider as a shared global store and context. Containerised all services using Docker and implemented a docker compose for this multi-container application Deployed the application to GCP using GCE, GKE and NGINX Ingress. Implemented all yaml files for all deployment pods, services, secrets, networking policy, and ingress resources.	Clevon created documentation for deployment on all development environments. He wrote individual portions of the report.
Chang Jing Yan		Jing Yan handled the user collaboration services. He implemented most of the backend logic for the collaborative Monaco Editor, such as shared cursors and language persistence. He also implemented the enhanced question service nice-to-have feature.	Jing Yan created API documentation with Swagger UI in user service. He helped to delegate tasks in sprint planning. He wrote individual portions of the report.
Lee I-Shiuan	M2: Matching Service M3: Question Service M5: User Interface N1: Communication Feature (Chat) N2: History Feature N3: Code Execution Feature	I-Shiuan handled the Question Service and Communication Feature and integrated them into the frontend. He also added question filtering-by-difficulty functionality in the frontend, along with the Matching Service.	I-Shiuan created API documentation with Swagger UI in the question service. He also did code reviews for Pull Requests. He wrote individual portions of the report.

	N5: Syntax Highlighting		
Tan Wei Xuan Rachel		Rachel handled the History and Code Execution backend logic. She also integrated them with the frontend.	Rachel created API documentation with Swagger UI in the history and code execution service. She also wrote individual portions of the report.
Ang Min Cher, Branda		Branda came up with the overall frontend design (UI) of the app including page layout for all services, app theme and logo, along with preliminary integration of Question Service to frontend. She also handled one nice-to-have: Syntax Highlighting using Monaco Editor	Branda helped with populating the question bank. She also set the general structure of the report and wrote individual portions of the report.

1. Introduction

1.1 Background

In the realm of engineering job applications, it is imperative to undergo technical interviews. Consequently, candidates must exhibit proficiency in problem-solving and effectively conveying technical ideas to enhance their prospects of securing employment. While many individuals rely on established platforms like LeetCode and AlgoExpert for honing their technical interview skills, it's crucial to note that these platforms often lack the resources for practising vital communication abilities required in technical interviews. These essential skills encompass articulating one's thought process to a peer and engaging in algorithmic analysis discussions.

1.2 Purpose

The objective at hand involves the development of a web application designed to assist students in enhancing their readiness for technical interviews. PeerPrep is focused on providing a distinctive and efficient platform for students to collectively review fundamental algorithmic principles. Through a peer-based learning framework, students have the opportunity to engage in collaborative coding, thereby facilitating mutual learning and alleviating the tedium often associated with individual study sessions.

1.3 Project Scope

PeerPrep represents an intuitive and user-centric collaborative tool that enables users to pair up with others for the purpose of practising technical questions. The platform offers three distinct difficulty levels, fostering a gradual and progressive learning experience. Importantly, user identities are kept confidential throughout the session, with only usernames being utilised. This approach is employed to cultivate a secure and inclusive learning environment for all participants.

1.4 Target Audience

Our intended audience comprises individuals in search of peer programming exercises to ready themselves for technical interviews. It is our intention to maintain the application's availability at no cost for the foreseeable future.

2. Overall Description

2.1 Product Perspective

In light of the limited chances for honing communication abilities in existing technical platforms, PeerPrep seeks to provide a platform for job-seeking students to review fundamental algorithmic concepts through collaborative coding. Collaboration fosters an environment where users are prompted to express their ideas and algorithms using straightforward and readily comprehensible language, a crucial soft skill in the technical field. Additionally, by pairing users with varying levels of question complexities, students are exposed to a diverse range of problem sets, which serves to bolster their technical prowess.

2.2 Product Features

The list of features implemented in the PeerPrep is as follows.

- User Interface (Frontend)
- User Service
- Question Service
- Matching Service
- Collaboration Service
- Code Execution Service
- History Service

2.3 Operating Environment

The PeerPrep back-end services operate in Windows, macOS and Linux operating system environments. The frontend operates within web browser environments such as Google Chrome and Safari.

3. Functional Requirements (FRs)

In the upcoming sections, the different Functional Requirements (FRs) are categorised based on the services they pertain to. The majority of these FRs have been successfully fulfilled, although there are a few that were either partially attempted or ultimately discontinued. A comprehensive list of the dropped FRs can be found in Section 3.7.

3.1 User Service

User Service is responsible for the authentication and authorisation of a user, as well as the Create, Read, Update and Delete (CRUD) operations on a user's profile information.

ID	Functional Requirement		Priority
FR1.1	User Service allows users to sign up.		High
FR1.2	User Service allows users to log in.		High
FR1.3	User Service allows users to log out of the account.		High
FR1.4	User Service allows users to modify their account information.		
	FR1.4.1	User Service allows users to modify their usernames.	High
	FR1.4.2	User Service allows users to modify their emails.	High
FR1.5	User Service allows users to delete their accounts.		High
FR1.6	User Service does authentication and authorisation on API requests for all protected resources.		
	FR1.6.1	User Service creates and passes a JWT to the client upon successful login.	High
	FR1.6.2	User Service denies access to protected resources to unauthenticated users.	High
	FR1.6.3	User Service denies access to protected resources to unauthorised users.	High

3.2 Matching Service

Matching Service is responsible for pairing up two users and creating a session for them to collaborate in.

ID	Functional Requirement	Priority
FR2.1	Matching Service matches users based on their chosen question complexity.	High
FR2.2	Matching Service allows users to start searching for their matches	High
FR2.3	Matching Service allows users to cancel their search for matches.	High
FR2.4	Matching Service creates a session with a unique room ID for the matched users to collaborate in.	High
FR2.5	Matching Service notifies the user to try again after a 30-second timeout if no match is found.	Medium

3.3 Question Service

Question Service is responsible for the basic CRUD operations for a question.

ID	Functional Requirement	Priority
FR3.1	Question Service allows Admin users to CREATE questions.	High
FR3.2	Question Service allows Admin users to READ questions.	High
FR3.3	Question Service allows Admin users to UPDATE questions.	High
FR3.4	Question Service allows Admin users to DELETE questions.	High
	N4: Enhance Question Service	
FR3.5	Question Service should allow users to get random questions, by complexity, during a collaboration session on the fly.	High
FR3.6	Question Service should allow admin users to filter questions by complexity for easy management.	Medium
FR3.7	Question Service should allow admin users to search questions by title for easy retrieval and management.	Medium

3.4 Collaboration Service

Collaboration Service is responsible for providing a shared code editor for a pair of users (“peers") to use during a live session.

ID	Functional Requirement		Priority
FR4.1	Collaboration Service supports multiple programming languages in the shared code editor.		High
FR4.2	Collaboration Service provides two users with a shared code editor.		
	FR4.2.1	Collaboration Service allows users to dynamically edit code concurrently	High
	FR4.2.2	Collaboration Service allows users to choose and switch between 4 programming languages (Python, Java, C++, JavaScript).	Medium
	FR4.2.3	Collaboration Service provides a default code template for each programming language.	Medium
	FR4.2.4	Collaboration Service allows users to see their peers’ cursor positions in the code editor.	Low
	FR4.2.5	Collaboration Service allows users to see their peers’ highlighted code within the code editor.	Low
FR4.3	N1: Collaboration Service allows users to communicate with their peer.		
	FR4.3.1	Collaboration Service allows peers to send messages to each other when both are online.	Medium
	FR4.3.2	Collaboration Service allows peers to receive messages from each other when both are online.	Medium
	FR4.3.3	Collaboration Service allows users to view all the messages exchanged for that session	Medium
FR4.4	Collaboration Service notifies users when their peer goes offline.		Medium
FR4.5	Collaboration Service should save the current code when users change between programming languages		High
FR4.6	Collaboration Service should allow users to reset their code back to the default code template		Medium
FR4.6	Collaboration Service prompts users to resume sessions when users accidentally navigate away from the session page without ending the session.		Medium
FR4.7	Collaboration Service allows users to continue their ongoing sessions when users refresh the page.		Medium

ID	Functional Requirement	Priority
FR4.8	Collaboration Service allows users to request to proceed with a new question.	High
	N5: Collaboration Service has syntax highlighting	
FR4.9	Collaboration Service supports syntax highlighting for all 4 languages	Medium
FR4.10	Collaboration Service supports improved code formatting such as auto indentation, bracket matching, etc.	Low
FR4.11	Collaboration Service supports autocompletion while coding.	Low

3.5 Code Execution Service

Code Execution Service is responsible for allowing users to run their code and receive feedback to debug their solutions.

ID	Functional Requirement	Priority
FR5.1	Code Execution Service should allow users to compile and run code in 4 different programming languages (Python, Java, C++, Javascript).	High
FR5.2	Code Execution Service should allow users to see the output of their print statements if there are no errors.	High
FR5.3	Code Execution Service should allow users to see the type of errors in their code.	High

3.6 History Service

History Service is responsible for allowing users to view their past attempts during past sessions.

ID	Functional Requirement	Priority
FR6.1	History Service should create an attempt for every question a user attempts.	High
FR6.2	History Service should sort the past attempts in descending order so that users can see their most recent attempts first.	High
FR6.3	History Service should allow users to see question details for each attempt.	Medium
FR6.4	History Service should allow users to see their last saved code for each attempt.	Medium
FR6.5	History Service should allow users to filter attempts based on difficulty level.	Medium
FR6.6	History Service should allow users to search for past attempts based on the question title.	Medium

3.7 Dropped FRs

We opted to remove certain feature requests (FRs) due to tight time constraints. While these were considered nice-to-haves, they are not essential to the core functionality of our application.

ID	Functional Requirement	Priority
DFR1	All services should include extensive (and automated) unit, integration, and system testing using CI.	Low
DFR2	User Service allows users to reset passwords.	Low
DFR3	User Service has email verification upon sign-up.	Low
DFR4	User Service allows users to create accounts using popular platform accounts (e.g. GitHub, Google).	Low
DFR5	User Service automatically logs users out after a period of inactivity.	Low
DFR6	Question Service allows users to save questions for future viewing (favourites).	Low
DFR7	Question Service allows users to write remarks for their questions.	Low
DFR8	Collaboration Service allows users to choose questions on specific topics.	Low
DFR9	Code Execution Service allows user to run their source code against test cases.	Low
DFR10	Code Execution Service allows users to know which test cases have passed or failed.	Low

4. Non-Functional Requirements (NFRs)

ID	Functional Requirement	Quality Attribute	Priority
1	Passwords must be minimally 8 characters long	Security	Medium
2	Passwords are salted and hashed before they are stored securely in the database	Security	High
3	All API calls should be authenticated (and authorised if needed) with a valid JWT	Security	High
4	Registration of account should not exceed 2 seconds	Performance	High
5	Login and creation of JWT should not exceed 2 seconds	Performance	High
6	CRUD of questions should not exceed 2 seconds	Performance	High
7	Dynamic live update of users' code editor should not exceed 2 seconds	Performance	High
8	Matching should happen within 5 seconds of 2 users being available	Performance	High
9	Matching should not load indefinitely -- UI should tell the user that no other users are currently available	Performance	High
10	Question Service should be able to support 500+ questions without noticeable lag	Performance	High
11	Live update of collaboration services should not have a delay greater than 3 seconds	Performance	High
12	User experience should be smooth and user interface should be clear	Usability	High
13	Ensure seamless integration with external APIs like Judge0	Third Party Integration	High

5. Technology Stack

Component	Framework, Technologies, Tools
Frontend	React ChakraUI – UI component library MobX – state management library
Microservices	Express.js
Databases	PostgreSQL hosted on ElephantSQL MongoDB hosted on MongoDB Atlas
External APIs	Judge0 API – code execution Monaco Editor& Yjs – collaborative code editor
Authentication and Authorisation	JSON Web Tokens (JWT)
Deployment	Docker – containerisation NGINX Ingress - API Gateway Google Cloud Platform – deployment on production system
Project Management	GitHub Google Drive

6. Architecture Considerations

6.1 Overall Architecture

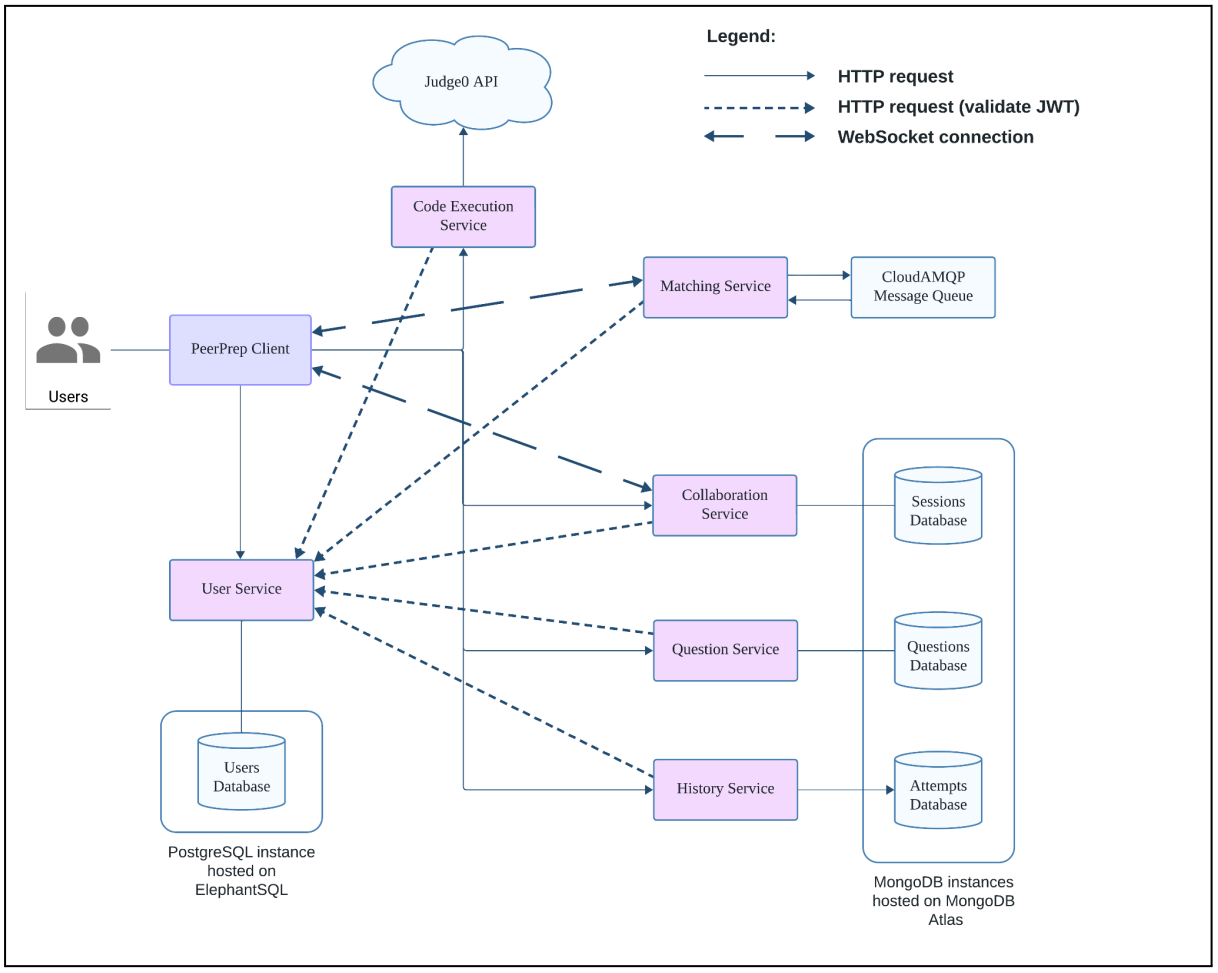

Figure 1: Architecture diagram

6.2 Microservices vs Monolith

When designing the application, two architectural approaches were mainly considered, the monolithic approach and the microservices approach.

In a monolith application, all components of the application are packaged into one single application.

In the microservices approach, the components are separated into smaller, independently deployable services where each service has a well-defined responsibility. The user service will handle all logic related to the management of users, and so on and so forth.

The diagram below illustrates the difference between the two approaches:

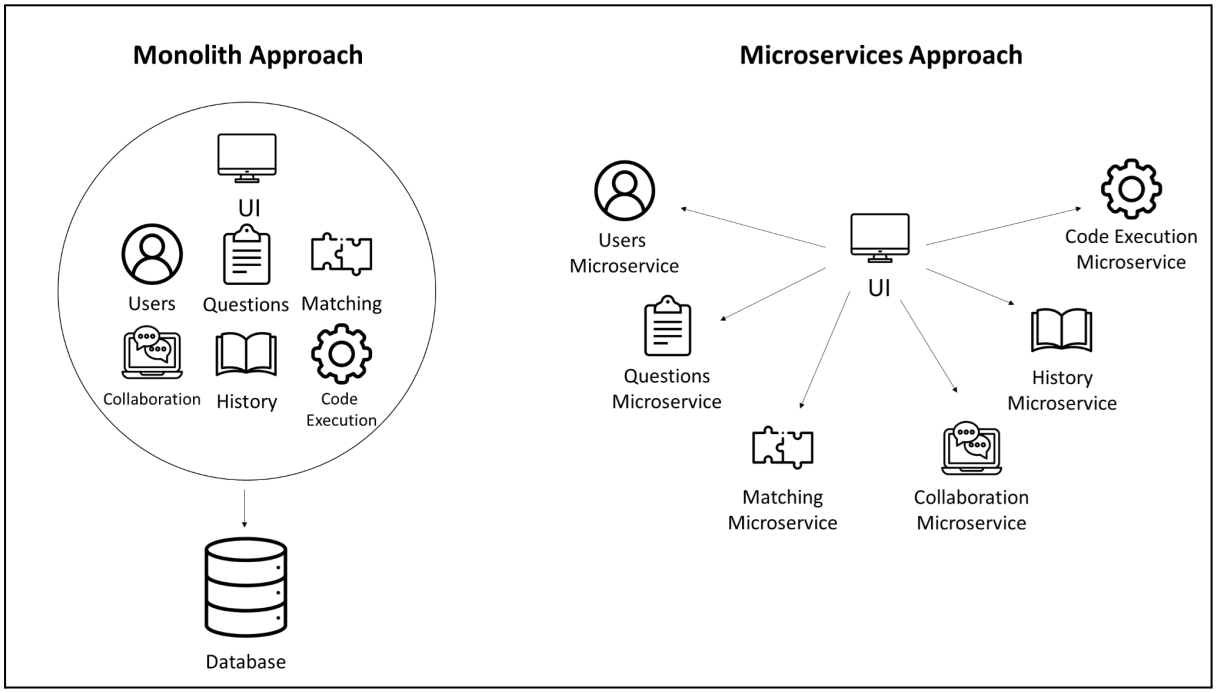

Figure 2: Monolith vs Microservices

The following shows some characteristics of monolithic and microservices architecture.

Monolithic	**Microservices**
Simple	More complex
Tightly coupled	Loosely coupled
Unified deployment	Independently deployable
Single point of failure	No single point of failure

Ultimately, our group has decided to go with the **microservices** architecture.

We acknowledge that the simplicity that a monolithic application brings will help to speed up development time. However, Peerprep is designed to be a collaborative platform, meant to handle a large number of concurrent users. A microservices application will be more scalable in the future to allow for a large number of users as each service can be scaled independently. Suppose that there is a high demand for user service, the user service can be scaled independently without having to allocate resources to the entire application. Moreover, we can employ orchestration tools such as Kubernetes to automate the replication of services on demand.

The loose coupling associated with microservices allows our application to be more robust to changes and could save development time instead. Any changes within a single microservice will most likely be contained within the same microservice, and there will be minimal propagation to the rest of the application. In a software engineering context where user requirements are always evolving, our application will adapt to changes more easily.

Last, but not least, microservices also allow us to develop features in parallel more easily, and each member of the team can focus solely on one microservice without needing to depend on one another. A member of the team can work on the user microservice, while another member may work on the question microservice at the same time. Every microservice has a single responsibility and hence does not need to depend on the implementation of other services.

6.2.1 Database-per-Service Pattern

Our group went with a Database-per-Service pattern instead of a Shared Database approach.

While there was some duplication of data across databases, we decided it was appropriate where it happened.

For example, the Attempts collection in the History Service contains question information also found in the Question Service's Questions collection. In this scenario, we felt that data duplication was more appropriate given that Question documents might be updated by Admin users, but we wanted to ensure that Attempt documents reflected the Question details that were attempted by users at the point in which they were attempted and not the most updated Question details.

The other reason for this decision was also to achieve loose coupling between microservices. Each microservice was responsible for its own database and could define the schema as required, without affecting the other microservices' schemas.

6.3 Model-View-Controller (MVC) Pattern

6.3.1 Benefits of using the MVC pattern

Our microservices mostly followed an MVC pattern in order to have a clear separation of concerns and achieve modularity within each microservice.

The MVC pattern's clear separation of concerns facilitated efficient development and maintenance; the UI of the app (View) could be developed independently of the data representation logic (Model).

By decoupling the different components, changes in one component were less likely to affect another. This was in line with the Dependency Inversion Principle, as our components depended on the abstractions (ie. interfaces) provided by the other components. For instance, the client depended on the REST API provided by the microservice, so it was not affected by changes in implementation in the microservice.

6.3.2 Example of using MVC pattern

To illustrate how the Model, View and Controller components interact, we will use the example of the interaction between the frontend client and the Question Service.

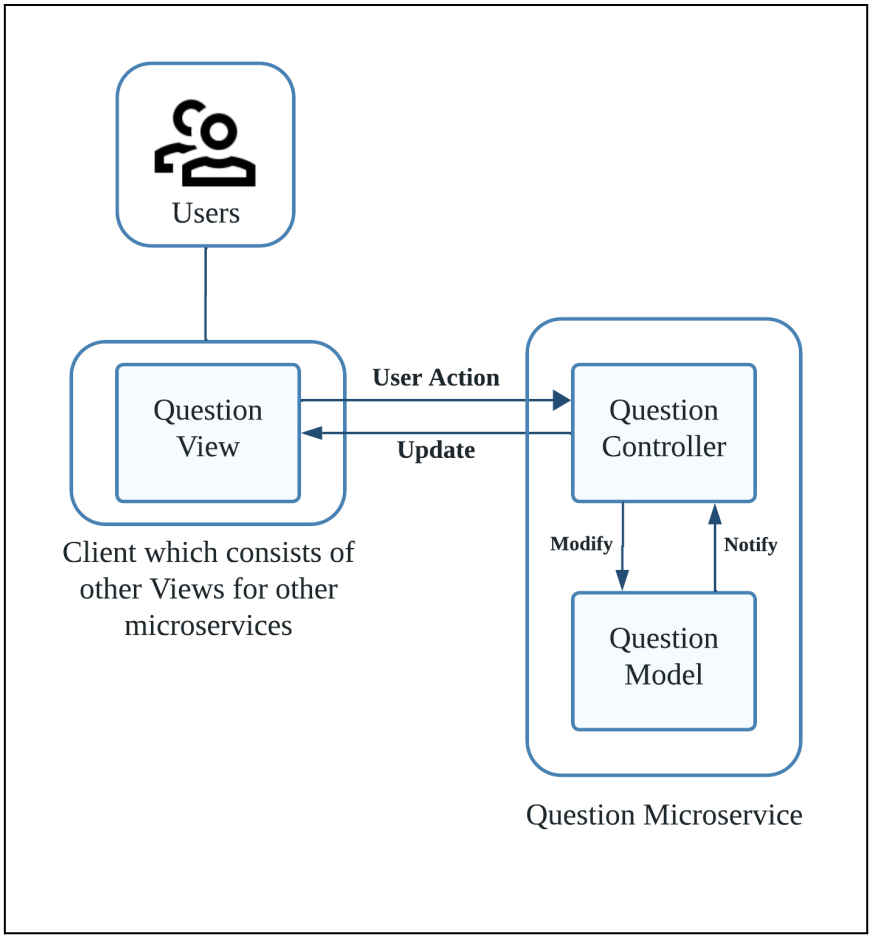

Figure 3: MVC pattern for Question Service

1. When an admin user updates a question (via the View component which resides in the frontend client), an `HTTP PUT` request is made to the Question Service's Controller component.
2. The Controller takes in the user input from the HTTP request and updates the Model.
3. The Model component models our question data. It updates the specified question and validates all changes to ensure that the data written to the database conforms to the schema defined by the Model.
4. Upon a successful update in the Model (a successful database write), the Controller updates the View with the new data in the form of an HTTP response.

6.4 GKE and NGINX

We mainly decided to go with Google Kubernetes Engine (GKE) due to the following advantages that GKE provide:

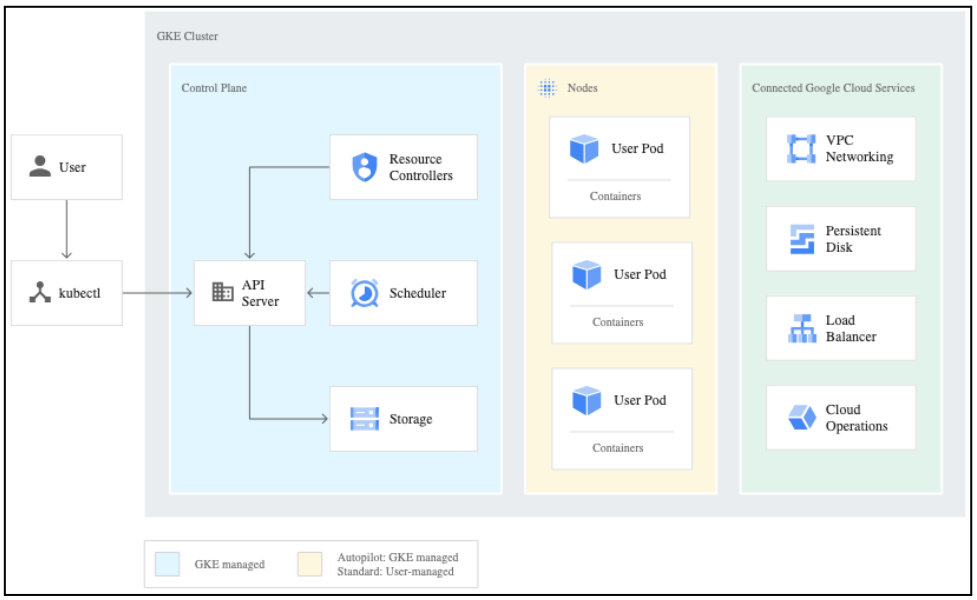

Figure 4: Cluster Architecture diagram for Kubernetes

Ref: https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture

1. **Orchestration with Kubernetes:** GKE is essentially Kubernetes managed by Google. It provides robust orchestration and management for our containerised microservices, ensuring efficient scaling, load balancing, and automation of deployment, scaling, and management tasks.
2. **Scalability and Flexibility:** With GKE, we can easily scale our application up or down as needed. It enables efficient resource utilisation and cost-effectiveness, crucial in a microservices environment where components might have varying resource demands.
3. **Google's Infrastructure:** Benefit from Google's robust and secure infrastructure. GKE leverages Google's global network, providing reliability and security while ensuring high availability and low latency for our application.
4. **Automatic Updates and Patches:** Google manages the Kubernetes master, providing automated updates and patches, freeing us from handling the operational overhead of maintaining the control plane.

We are also using an API gateway (NGINX ingress controller) of some kind that redirects requests to the relevant microservices. We did this because of the features and added benefits NGINX ingress provides. There were also some challenges that we were trying to solve by using NGINX ingress instead of leaving our microservices directly accessible from the public.

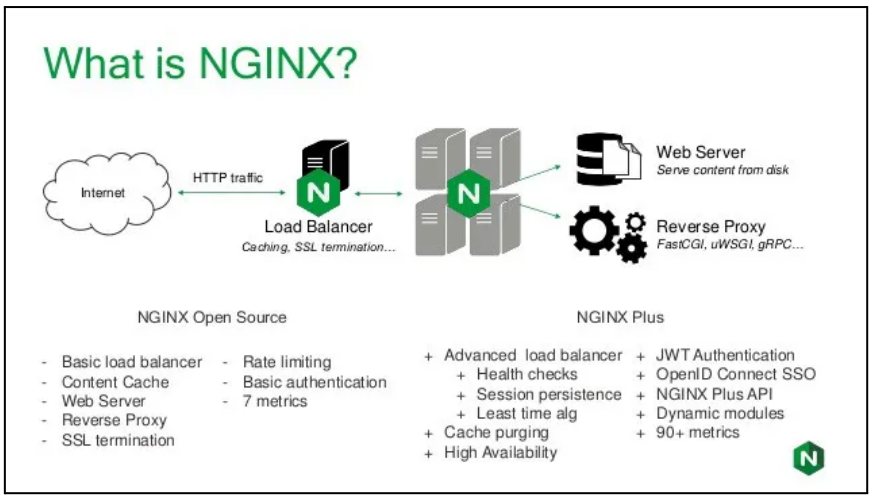

Figure 5: NGINX web server

Ref: https://medium.com/javarevisited/intro-to-nginx-web-server-part-1-bb590fad7035

1. **Centralised Control:** With ClusterIP, each service remains internal, accessible only within the cluster. The NGINX Ingress serves as a single entry point, controlling external access, leading to a centralised and controlled approach to managing external traffic.
2. **Efficient Load Balancing:** The NGINX Ingress load balancer handles routing and load balancing for incoming traffic, distributing it to appropriate services based on defined rules or configurations. This offers more granular control over routing and traffic distribution.
3. **Reduced Public IP Consumption:** When using LoadBalancer services, each service deployed can potentially acquire a separate external IP, which might not be necessary. The NGINX Ingress, with ClusterIP services, optimises IP consumption by consolidating external access under a single IP address which in turn is more efficient in terms of cost and routing, as well as enhancements in security.
4. **Simplified Network Configuration:** Configuring and managing multiple LoadBalancer or NodePort services can become complex, especially as the number of services grows. Using NGINX Ingress with ClusterIP services simplifies the network configuration, making it easier to manage and maintain.
5. **Enhanced Security:** External access is controlled by the Ingress, adding an additional layer of security by reducing the exposure of individual services to external networks. This centralised control helps in implementing security measures at a single point.
6. **Scalability Challenges:** Managing multiple LoadBalancer or NodePort services can become challenging as the system scales, potentially leading to network congestion or configuration issues.

7. Design Patterns

7.1 Observer Pattern

In React development, MobX is often used as a state manager, employing the observer pattern to facilitate reactive updates. The observer pattern allows components to automatically re-render when the observed state changes. This enhances the responsiveness and maintainability of the application.

The observer pattern involves defining a one-to-many dependency between objects so that when one object (the subject) changes state, all its dependents (observers) are notified and updated automatically. MobX seamlessly integrates with React to implement this pattern.

MobX uses a unidirectional data flow where actions change the state, which in turn updates all affected views.

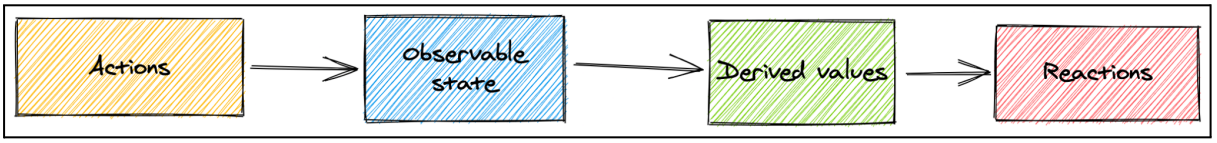

Figure 6: Unidirectional data flow

Ref: https://mobx.js.org/the-gist-of-mobx.html#principles

All derivations are updated automatically and atomically when the state changes. As a result, it is never possible to observe intermediate values.

All derivations are updated synchronously by default. This means that, for example, actions can safely inspect a computed value directly after altering the state.

7.2 Adaptor Pattern

The Adapter pattern serves as a structural design solution, facilitating cooperation between objects with incompatible interfaces. In the context of PeerPrep, a disparity arises when the PeerPrep Client endeavours to execute source code.

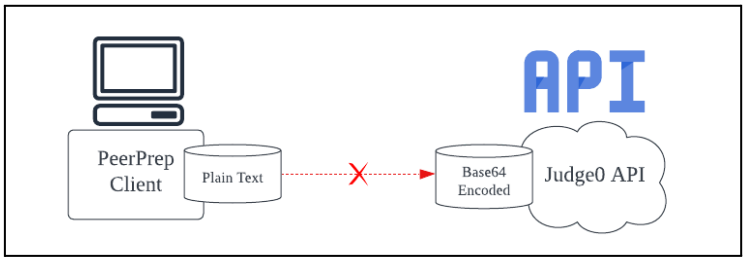

Figure 7: Disparity between client and code execution

The client transmits the code in plain text, whereas the Judge0 API anticipates Base64-encoded binary data. This interface incongruity hinders direct collaboration between the PeerPrep Client and the Judge0 API.

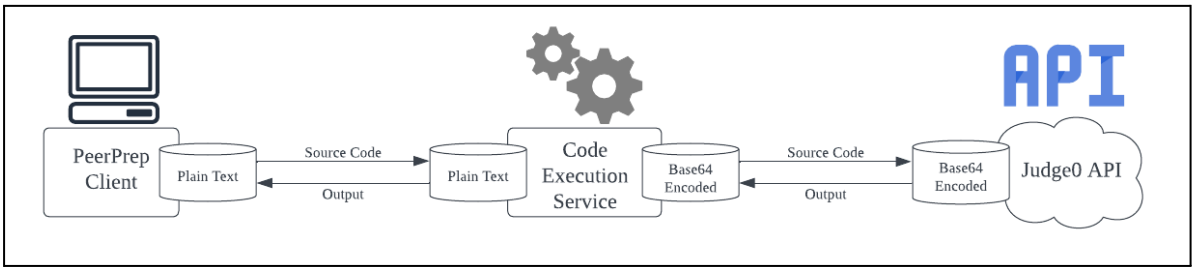

Figure 8: PeerPrep flow with adapter

To overcome this interface mismatch, the Code Execution Service functions as an adapter. It bridges the gap by converting the interfaces from plain text to Base64 encoding and vice versa. Upon receiving plain text source code from the client, the Code Execution Service encodes it as a Base64 string. Subsequently, it dispatches a GET request to the Judge0 API with the encoded source code. Post-execution, Judge0 responds with a Base64-encoded output, which the Code Execution Service decodes into plain text using UTF-8 encoding. The resulting plain text output is then sent back to the PeerPrep Client for display on the frontend.

This adaptation mechanism ensures seamless collaboration between the client and Judge0, mitigating the impediment caused by incompatible interfaces and fostering a more efficient workflow.

7.2.1 Benefits Of Adaptor

Undoubtedly, encoding the code on the client side before sending it to the Judge0 API is feasible. However, this approach risks violating the Single Responsibility Principle. Opting for the Code Execution Service to manage the data conversion introduces a clear separation of responsibilities.

Moreover, the Adapter aligns with the Open/Closed Principle. The code execution mechanism remains open to extension, allowing for the seamless inclusion of additional third-party software to enhance functionality. Simultaneously, it remains closed for modification, minimising the necessity to alter existing code when incorporating new functionalities. This adherence to the Open/Closed Principle promotes a more modular and maintainable codebase, fostering adaptability and scalability in the system.

8. Messaging Patterns

8.1 Event-Driven Communication with WebSockets

Some features of our app benefited from the use of event-driven communication.

Specifically, the real-time collaborative code editor and live chat features utilised WebSocket connections to our Collaboration Service server. This allowed for asynchronous communication between clients in a shared collaboration session (“peers”), and our server -- neither peer would be blocked while waiting for a response from the other as would be the case using HTTP request-response communication.

Further implementation-specific details can be found in Section 9.3 Collaboration Service.

8.2 Point-to-Point (P2P) Messaging Channel

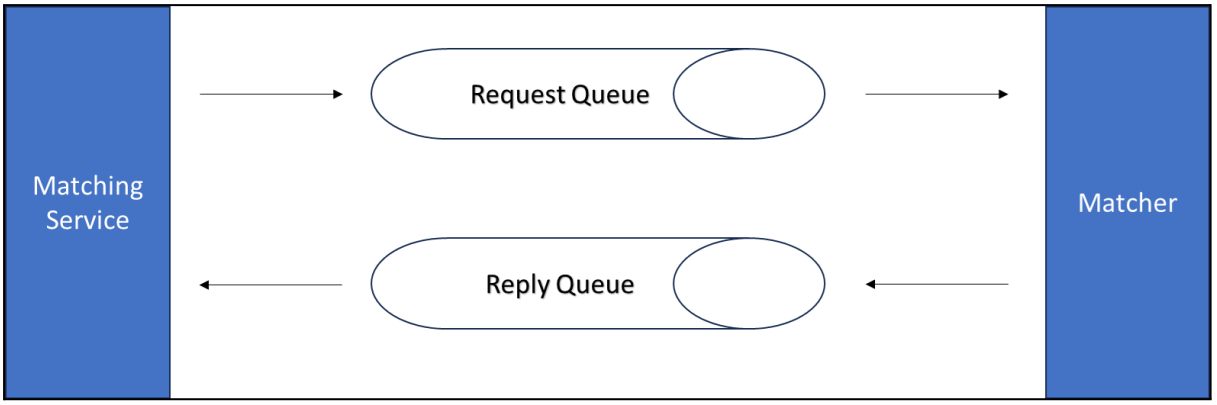

Figure 9: P2P Channel between Matching Service and Matcher

Our group employed the P2P messaging pattern using request and reply queues, within our matching service, using RabbitMQ.

Request and reply queues help us to decouple the requestor from the responder, and allow them to scale independently. The matching service need not know how the logic of the matching is implemented, and the matcher does not need to know the client details. As such, this allows the matching service and the matcher to be highly cohesive, each with a single well-defined responsibility.

Furthermore, request and reply queues also help the matching service process replies asynchronously. When the matching service receives a request, it can quickly send the message to the queue and go on to process other requests. This will improve the efficiency of the application, as it does not need to wait for a successful match before processing other requests.

Further implementation-specific details can be found in Section 9.2 Matching Service.

9. Implementation Details

9.1 Authentication and Authorisation With JWT

In the initial deliberations for our authentication and authorisation strategy, we considered both JSON Web Tokens (JWT) and Auth0 due to their robustness and widespread adoption. Given that PeerPrep is a relatively small-sized project that has straightforward authentication and authorisation requirements, we opted for JWT. The decision was influenced by its simplicity, control in implementation, and the desire to minimise external dependencies on third-party services to reduce the likelihood of code breakage.

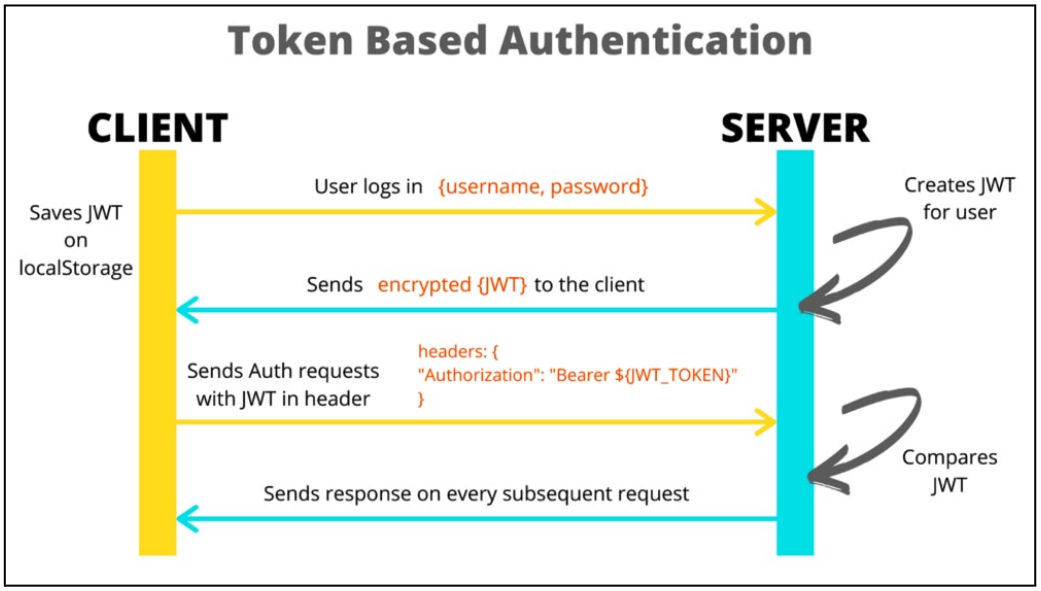

Figure 10: Token-Based Authentication

Ref: https://hackernoon.com/using-session-cookies-vs-jwt-for-authentication-sd2v3vci

9.1.1 Creation Of JWT

Upon successful user login, the User Service generates a new JWT that would be stored in the client's local storage. The JWT comprises 3 parts: the header, payload and signature.

- Payload:
  - The token is created with a payload consisting of the user ID and user type. This payload helps differentiate the user for whom the token is created, with a particular emphasis on the user's type which is crucial for authorisation.
- Signature:
  - We generated a lengthy random string as the secret access token, stored in the .env file. This ensures the integrity of the token and safeguards against tampering.

9.1.2 Authentication Process

When the client sends a request to the server, it retrieves the JWT from the local storage and includes it in the request header. The user service then validates the token using the `process.env.ACCESS_TOKEN_SECRET`. If the validation is successful, the server processes the request. Otherwise, a 401 error is thrown, indicating either the absence of a token or an invalid token, leading to a denial of the client's request.

For expired JWTs, the user is prompted to log in again to get a fresh JWT.

9.1.3 Authorisation Process

In PeerPrep, users fall into two categories: admin and participant. Only admins can create, update and delete (CUD) questions in our database.

The authorisation check is done on the server side.

- For CUD requests to the Question Service, in addition to authenticating the token, an extra authorisation check ensures that the token belongs to an admin. Only when both authentication and authorisation checks succeed does the server proceed with the request. Otherwise, a 403 error is thrown, signifying that the user is unauthorised to make such a request, leading to a denial of the request.
- For User Service, Collaboration Service and History Service, similar authentication and authorisation checks are done. The difference is instead of checking for the user's role before access is granted, the microservices check that a user can only modify their own resources (ie. their own profile information, session information etc).

Should a 403 error be thrown by the server, the client would catch it and display an error page to deny access to the client.

9.2 Matching Service

The matching service helps to match two users together, based on a common complexity.

We implemented the matching service through the use of request and reply queues as message channels. The following sequence diagram shows how the matching is implemented.

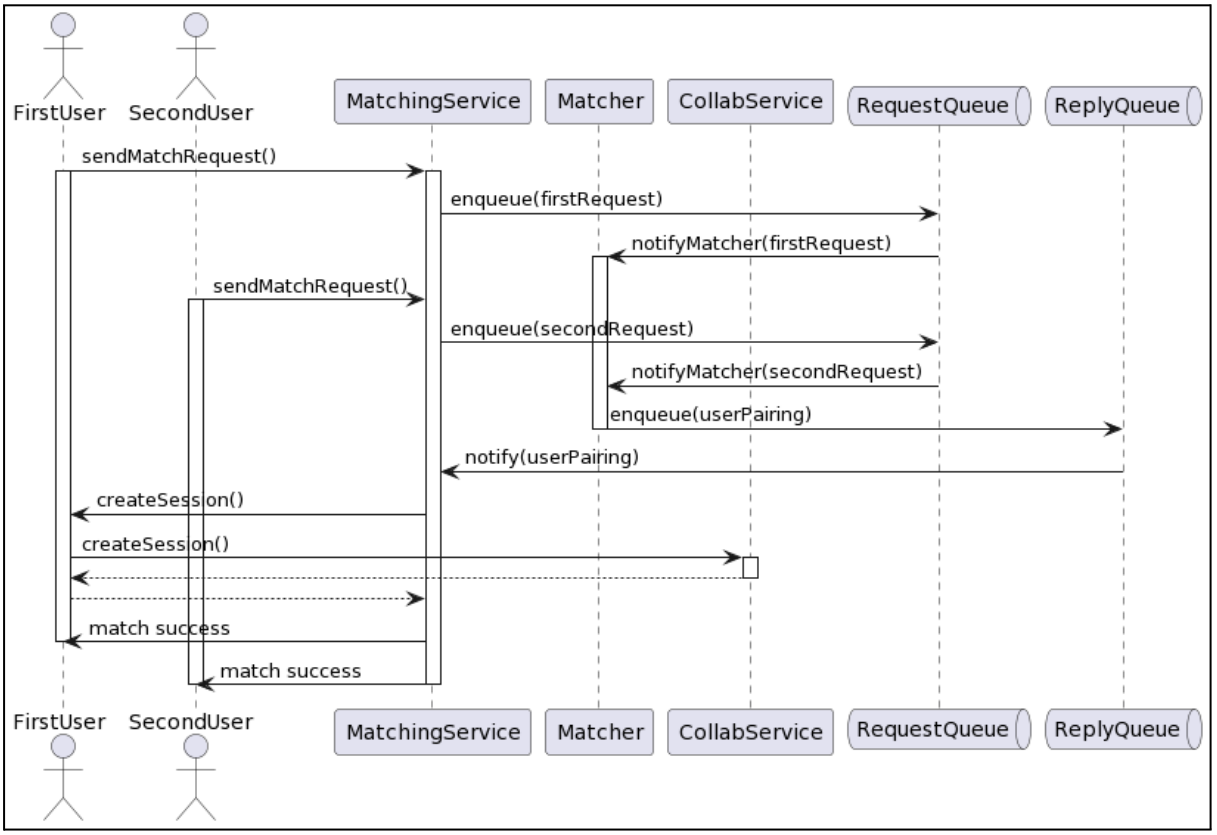

Figure 11: Matching Service sequence diagram (Match success)

The first user sends a request to the matching service, which will send a request into the request queue. The matcher will receive the message, and save the request in memory for 30 seconds. If the matcher receives a second request within 30 seconds, it will pair the users up and send it to the reply queue, which will be picked up by the matching service and notify the users of their match.

Furthermore, users are able to cancel their requests as well. A cancel request is sent through the same request queue. The matcher checks if the current request matches the details of the cancel request, and sends the successful cancel request through the reply queue. The matching service will notify the user if the cancel request is successful.

9.3 Collaboration Service

The Collaboration Service has 2 main components, both of which use WebSockets for its full-duplex two-way communication capabilities.

As WebSockets allow clients to communicate with our microservice by passing messages, we opted for both Command and Event message types -- which helps us in handling the incoming messages.

9.3.1 Collaborative Code Editor

The collaborative code editor must handle simultaneous updates from 2 users. To prevent race conditions where 1 user makes changes which overwrite the other user's changes, it is imperative to use conflict-free replicated data types (CRDTs) to store our shared text, as is used by apps such as Google Docs.

We used Yjs to handle this for us. Yjs also provides a WebSocket server package which we ran in our Collaboration Service on another port. This Yjs server handles the shared code editing in a live collaboration session between 2 users via socket connections.

We ran a Socket.io server on another port which we used to listen for Socket.io events we defined for specific collaboration features. "Sender" and "Receiver" refer to the user's web clients.

Some of the Command message types we defined are in the following table:

Command Message	**Sender action (which triggered the socket message)**	**Receiver action (upon receiving the socket message)**
`change-language`	Sender has changed programming language.	Receiver should change the language on their code editor, along with other UI changes such as syntax highlighting and formatting configurations of the new programming language.
`initiate-next-question`	Sender has requested to move on to the next question.	Receiver's UI should make the necessary changes (prompt the user to accept/reject the request).
`retrieve-next-question`	Sender has accepted the request to change the question.	Receiver should fetch the new question from the Collaboration Service via an HTTP request.
`leave-room`	Sender has left the session.	Receiver should leave the session too and call the necessary functions (to save the code attempt to History Service etc).

9.3.2 Communication Feature (Live Chat)

The live chat feature utilises WebSocket connections (via the Socket.io library). It shares the same socket as described in 9.3.1.

We considered two implementations:

1. Re-using the WebSocket from 9.3.1 that handles collaborative code editor functionality.
2. Creating another WebSocket via another microservice for Communication (live chat).

We eventually decided that Collaboration and Communication belonged in the same domain, and thus could belong in the same microservice.

Secondly, doing so reduced the overhead of creating multiple WebSocket connections to different microservices on users' clients.

Furthermore, as each collaborative session has exactly one code editor instance and one live chat instance, the need to scale these functionalities horizontally is likely to occur simultaneously. By consolidating both collaborative code editing and live chat within the same microservice, we eliminated the complexity of managing separate horizontal scaling considerations.

The Communication feature's socket messages are of the Event message type:

Event Message	**Sender action (which triggered the socket message)**	**Receiver action (upon receiving the socket message)**
`new-chat-message`	Sender has sent a chat message.	Receiver updates the UI with new chat message contents.
`peer-connected`	Sender has established a Socket.io connection to the server.	Receiver displays peer's status as "Online".
`user-disconnected`	Sender's Socket.io connection has been closed.	Receiver displays peer's status as "Offline".

9.3.3 Session State Persistence

We decided it was important to persist some session information so that if a user disconnected, they would not lose their progress. If a user changed programming language (eg. if their peer prefers to use a different language), our implementation of Collaboration Service also ensured the code of the previous language would not be lost, by persisting the state to the database. Persisting the code in the database was also crucial for the History Service which saves a record of the user's attempts along with their code.

10. DevOps and Software Development Life Cycle (SDLC)

10.1 Scrum Development Process

In the realm of software development, the scrum methodology emerges as a dynamic and effective approach, fostering iterative and incremental processes. This agile framework, known for its adaptability and speed, aims to deliver continuous value to customers throughout the project's evolution.

The scrum development process serves as a facilitator for our team to collectively and continuously meet evolving requirements. It provides a visual representation of project requirements and facilitates the management of product backlogs, prioritised to deliver essential features at various milestones.

Our team embraces the scrum methodology for several compelling reasons:

1. **Ease of Scalability** Scrum's iterative nature, confined within specific time frames, allows our team to concentrate on a defined set of functionalities during each iteration.
2. **Expectation Compliance:** The methodology ensures that all team members are well-informed about project expectations and their respective priorities. Regular verification by the Project Mentor guarantees adherence to requirements.
3. **Flexibility to Change:** Scrum's flexibility enables seamless adaptation to evolving requirements, promoting agility throughout the development process.
4. **Enhanced Software Quality:** Continuous feedback, improvement, and testing after each iteration upholds Scrum's commitment to delivering higher-quality software.
5. **Timely Predictions:** Through the establishment of sprints and goals within specific timeframes, scrum enables our team to forecast the availability of features at designated times.

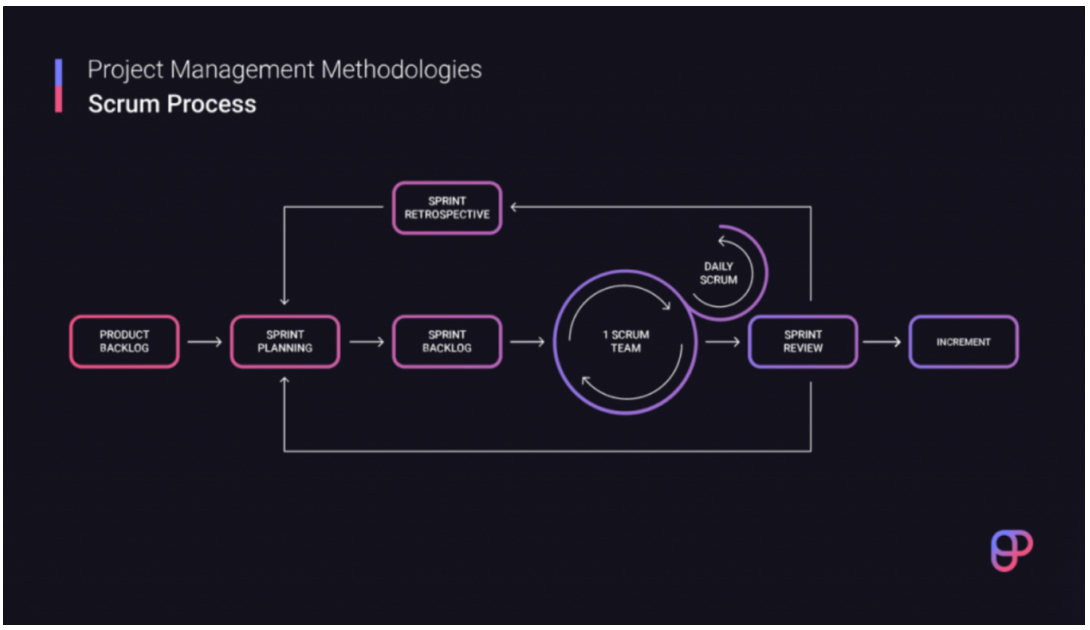

Figure 12: Typical Scrum Process
Ref: https://plaky.com/learn/project-management/scrum-project-management/

10.1.1 Sprint

A sprint, a brief iteration typically lasting between 2 weeks to 1 month, serves as a focused period for the development team to conceive, implement, and test new features.

Given the time constraint of our 13-week project, we opted to condense the sprint duration to 1 week instead of the standard 2 weeks. This adjustment facilitates more frequent meetings, enhancing our ability to stay abreast of ongoing tasks, address immediate needs, make informed decisions, and strategise the development trajectory of our project.

In our team's weekly sprint planning sessions, conducted either physically or virtually, we systematically compile the product backlog and deliberate on their respective priorities. Subsequently, we prioritise tasks with higher importance as the focal deliverables for the upcoming sprint. This iterative and adaptive approach ensures a dynamic response to evolving project requirements and promotes efficient progress within our constrained time frame.

Our team employs Google Drive as the primary tool for planning, monitoring, and overseeing our sprints. The rationale behind this choice is rooted in the widespread familiarity of Google Drive among team members. Moreover, it offers additional advantages such as seamless file sharing and a plethora of third-party extensions, like diagrams.net and Text Editor, which prove beneficial for future report writing and managing environment files.

This decision is in contrast to opting for widely-used platforms like Jira. Our evaluation led us to believe that Jira, tailored for large-scale corporations, provides features that surpass our project's needs. The associated learning curve and time investment required to fully harness and integrate these features were deemed excessive, especially considering the project's tight timeline. In essence, the simplicity, user-friendliness, and immediate applicability of Google Drive align more closely with our project's specific requirements and resource constraints.

The figure presented below depicts our sprint planning document for Week 4. A well-defined meeting agenda ensures that all participants arrive prepared and with a focused understanding of the tasks at hand. This streamlined approach expedites discussions and enhances overall efficiency. Comprehensive notes are meticulously recorded during the meeting, encompassing all discussions and to-dos for the upcoming sprint. This meticulous documentation serves to provide clarity regarding tasks that need attention and decisions made during previous sessions.

Date	12/09/2023	**Time**	1300H
Location	COM3 Terrace	**Prepared By**	Clevon Wong

1. ATTENDANCE OF MEETING	
Attendees	**Absentee (s)**
Branda, JingYan, Ishuan, Rachel, Clevon	-

2. MEETING AGENDA		
Agenda Items		**Presenter**
1	Share what you have learnt about the project NXs	Everyone
2	Plan and decide on project scope	Everyone
3	Establish Todos for this week	Everyone

3. CARRY OVER			
	Task	**Est. Effort**	**Est. Due Date**
1.	-	-	-

3. MEETING DISCUSSION			
	Discussions	**Action plan/ Action By**	**Due Date**
1.	What nice to haves we decided on. Refer to this document for the discussion notes.	-	-
2.	Project FRs and NFRs. Refer to this document for the discussion notes.	-	-

4. ACTION ITEMS			
	Actions	**Action By**	**Due Date**
1.	Clean up on Assignment1, branch out from master, and push to Github	Clevon	13/09/2023
2.	Each person take 1 green and 1 yellow and research in depth about it 1. Find out what it is and a possible service that we can implement 2. Find out how we can build it	Everyone	15/09/2023

Figure 13: Week 4 Sprint Planning Document

10.1.2 Product Backlog

The product backlog functions as a dynamic, prioritised catalogue outlining the enhancements required for the product. Following consensus on the project's goals and features, our team methodically dissects each feature into manageable tasks suitable for individual members. These tasks are then seamlessly integrated into the product backlog.

To monitor the overarching objectives and features of the project, we utilise an Excel sheet. However, for tracking weekly tasks, we rely on Meeting Notes generated after each planning session. This approach allows for a quick overview, providing access to all resources referenced in discussions and a comprehensive understanding of individual responsibilities.

The figure below shows how we use Google Excel to keep track of the functional requirements and product backlog of the project while prioritising and visualising a roadmap on what our upcoming sprints will look like.

ID	Functional Requirement		Priority	Sprint
1	User Authentication			
1.1	User log in.			
	1.1.1	Users can login with username and password	High	2
	1.1.2	Users can use JWT for tokens	High	3
1.2	User sign up functionality.			
	1.2.1	Users can create accounts using username/email + password.	High	2
	1.2.2	Users can create accounts using other existing accounts (Google/GitHub etc).	Low	5
	1.2.3	New users can be verified by email.	Low	5

Figure 15: User authentication portion of product backlog

10.2 Deployment

We have a single Github repository encapsulating all of the code for the entire application.

The main structure of the repository is:

- `client` – this is our frontend folder which contains the entire frontend application
- `src` – this folder contains all the applications that run on the backend
- Other miscellaneous files that affect the entire application. For example, deployment files, READMEs, package.json, etc

Our application supports several deployment environments.

1. Local deployment using NPM scripts
2. Local deployment using Docker
3. Deployment on Google Cloud

10.2.1 Cloning the repository and setting up environment variables

Before running the application in any environment, please follow the steps mentioned below to set up our application on your machine of choice. You will need accounts with ElephantSQL, MongoDB, CloudAMQP, and RapidAPI to set up our project locally.

Steps:

1. Clone our Github Repository from here
2. In the `./client` folder, create a file called `.env` which contains the following variables (make sure you add the `.` in front of "`env`")
    a. `VITE_USER_BASE_PATH=http://localhost:8000/api`
    b. `VITE_QUESTION_BASE_PATH=http://localhost:3000/api/questions`
    c. `VITE_MATCHING_ENDPOINT=http://localhost:5001`
    d. `VITE_HISTORY_BASE_PATH=http://localhost:3001/api`
    e. `VITE_CODE_EXECUTION_BASE_PATH=http://localhost:5002/api`
    f. `VITE_COLLABORATION_BASE_PATH=http://localhost:8001`
    g. `VITE_COLLABORATION_WS_ORIGIN=ws://localhost:8002`
3. In the `./src/user-service` folder, create 2 files called
    a. `.env.local` which contain the following variables
        i. `PORT=8000`
        ii. `PSQL_HOST=rain.db.elephantsql.com`
        iii. `PSQL_PASSWORD=<Elephant SQL account password>`
        iv. `PSQL_USER=<Elephant SQL account>`
        v. `PSQL_DATABASE=<Elephant SQL database name>`

vi. `ACCESS_TOKEN_SECRET=<your own access token secret>`

b. `.env.docker` which contains the following variables

i. `PORT=8000`

ii. `PSQL_HOST=rain.db.elephantsql.com`

iii. `PSQL_PASSWORD=<Elephant SQL account password>`

iv. `PSQL_USER=<Elephant SQL account>`

v. `PSQL_DATABASE=<Elephant SQL database name>`

vi. `ACCESS_TOKEN_SECRET=<your own access token secret>`

4. In the `./src/question-service` folder, create 2 files called
   a. `.env.local` which contain the following variables
      i. `PORT=3000`
      ii. `QUESTION_DATABASE_URL=<A MongoDB URL>`
      iii. `USER_BASE_PATH=http://localhost:8000/api`
   b. `.env.docker` which contains the following variables
      i. `PORT=3000`
      ii. `QUESTION_DATABASE_URL=<A MongoDB URL>`
      iii. `USER_BASE_PATH=http://user-service:8000/api`

5. In the `./src/matching-service` folder, create 2 files called
   a. `.env.local` which contain the following variables
      i. `PORT=5001`
      ii. `AMPQ_HOST=<A CloudAMQP instance URL>`
   b. `.env.docker` which contains the following variables
      i. `PORT=5001`
      ii. `AMPQ_HOST=<A CloudAMQP instance URL>`

6. In the `./src/history-service` folder, create 2 files called
   a. `.env.local` which contain the following variables
      i. `PORT=3001`
      ii. `HISTORY_DATABASE_URL=<A MongoDB URL>`
      iii. `USER_BASE_PATH=http://localhost:8000/api`
   b. `.env.docker` which contains the following variables
      i. `PORT=3001`
      ii. `HISTORY_DATABASE_URL=<A MongoDB URL>`
      iii. `USER_BASE_PATH=http://user-service:8000/api`

7. In the `./src/collaboration-service` folder, create 2 files called
   a. `.env.local` which contain the following variables
      i. `PORT=8001`

ii. `COLLABORATION_DATABASE_URL=<A MongoDB URL>`
iii. `USER_BASE_PATH=http://localhost:8000/api`

b. `.env.docker` which contains the following variables
i. `PORT=8001`
ii. `COLLABORATION_DATABASE_URL=<A MongoDB URL>`
iii. `USER_BASE_PATH=http://user-service:8000/api`

8. In the `./src/code-execution-service` folder, create 2 files called
   a. `.env.local` which contain the following variables
      i. `REACT_APP_RAPID_API_HOST="judge0-ce.p.rapidapi.com"`
      ii. `REACT_APP_RAPID_API_KEY=<Rapid API Key> // TODO`
      iii. `REACT_APP_RAPID_API_URL="https://judge0-ce.p.rapidapi.com"`
      iv. `USER_BASE_PATH=http://localhost:8000/api`
   b. `.env.docker` which contain the following variables
      i. `REACT_APP_RAPID_API_HOST="judge0-ce.p.rapidapi.com"`
      ii. `REACT_APP_RAPID_API_KEY=<Rapid API Key>`
      iii. `REACT_APP_RAPID_API_URL="https://judge0-ce.p.rapidapi.com"`
      iv. `USER_BASE_PATH=http://user-service:8000/api`

10.2.2 Local Deployment using NPM scripts

Please follow the steps mentioned below to run the application locally using NPM scripts. Make sure you have completed the instructions mentioned in 10.2.1.

Steps:

1. Ensure you are in our project's root folder
2. Run `npm i concurrently` if you don't have Concurrently installed
   a. We are using Concurrently to concurrently run the same commands for every microservice in our application.
3. Run `npm run install-all`
   a. This will run the command:

```
concurrently -c
\"red,green,yellow,blue,magenta,cyan,gray\"
\"npm:install:*\"
```

   b. Which essentially runs `npm install` in each individual microservice.
4. Run
   a. On Windows (PowerShell): `$env:PEERPREP_ENV="local"; npm run dev-all`
   b. On Unix: `PEERPREP_ENV=local; npm run dev-all`

c. This will run the command:

```
concurrently -c
\"red,green,yellow,blue,magenta,cyan,gray\" \"npm:dev:*\
```

d. Which essentially runs `npm run dev` in each individual microservice.

Now you can open http://localhost:5173/ to view the application.

10.2.3 Local Deployment using Docker

Please follow the steps mentioned below to run the application locally using Docker. Make sure you have completed the instructions mentioned in 8.2.1.

Steps:

1. If you don't already have Docker installed, download the right version of Docker corresponding to the machine you are using and complete the Docker installation.
2. After you have successfully downloaded and installed Docker, run the Docker Desktop application.
3. Once Docker Desktop is up and running, ensuring you are in our project's root folder, run
    a. On Windows (PowerShell): `$env:PEERPREP_ENV="docker"; docker compose up`.
    b. On Unix: `PEERPREP_ENV=docker docker compose up`.
    c. This will build run all the Dockerfiles in each microservice which handles the installing of node modules and starting the application

Now you can open http://localhost:5173/ to view the application.

10.2.4 Deployment on Google Cloud

To host the application, we are deploying it on Google Cloud Platform and utilising Google Container Registry, Google Kubernetes Engine, as well as NGINX Ingress.

We build and submit images of our microservices to Google Container Registry and then those images are deployed to a private cluster on Google Kubernetes Engine. We then use NGINX Ingress for its features such as rewriting and rerouting to route requests from users to the right microservice in our cluster. By doing so, we keep all of our microservices private and inaccessible directly from external sources. You may refer to the diagram below to see how our cluster works.

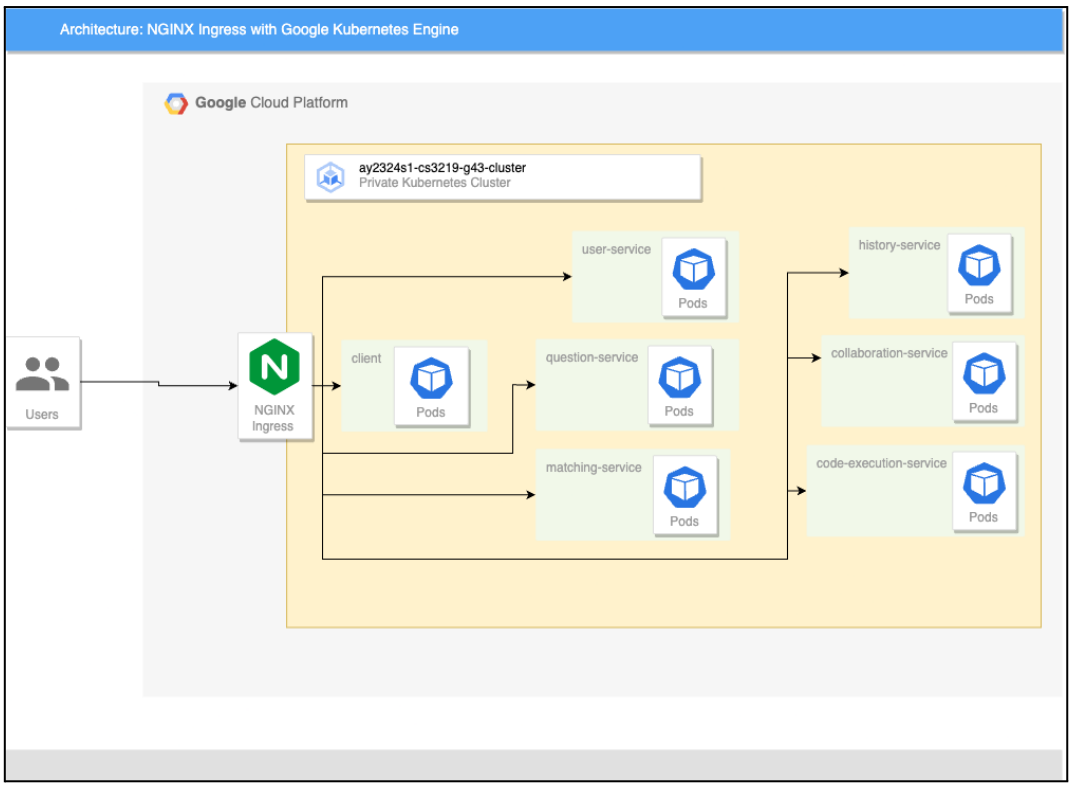

Figure 16: Architecture diagram for GCP

Please refer to 6.4 for justification of our choices of design patterns.

11. Application Screenshots

11.1 Landing Page

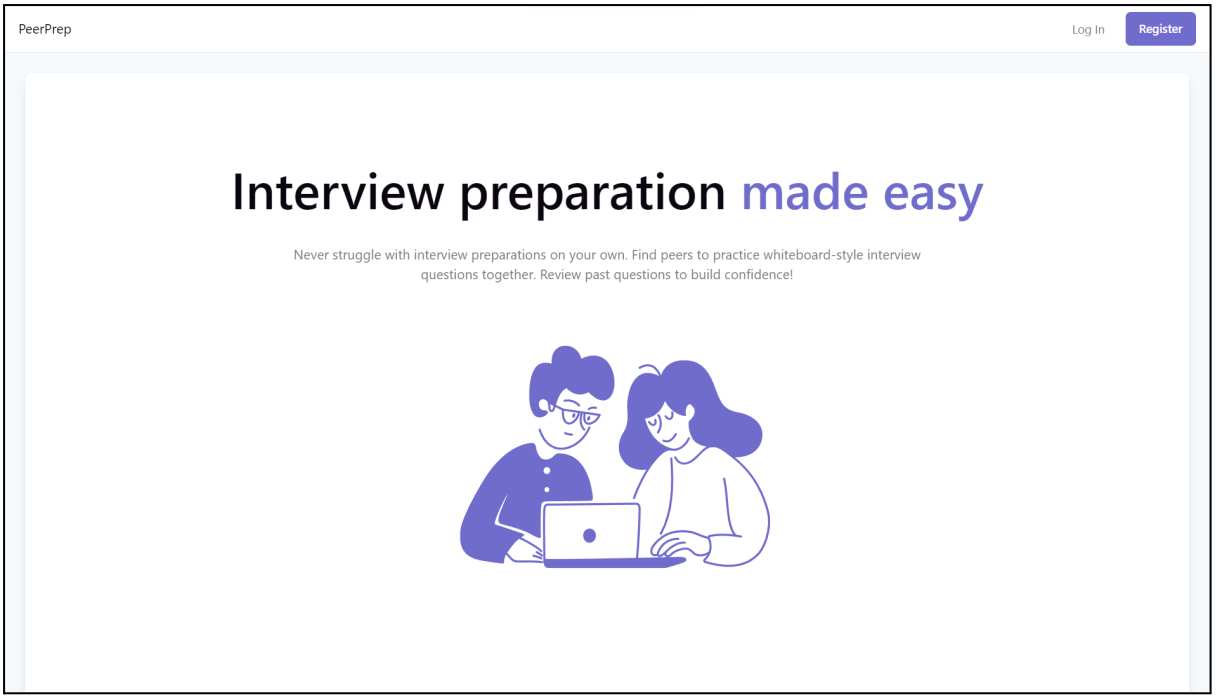

Figure 17: Landing page for Visitors (Not logged in)

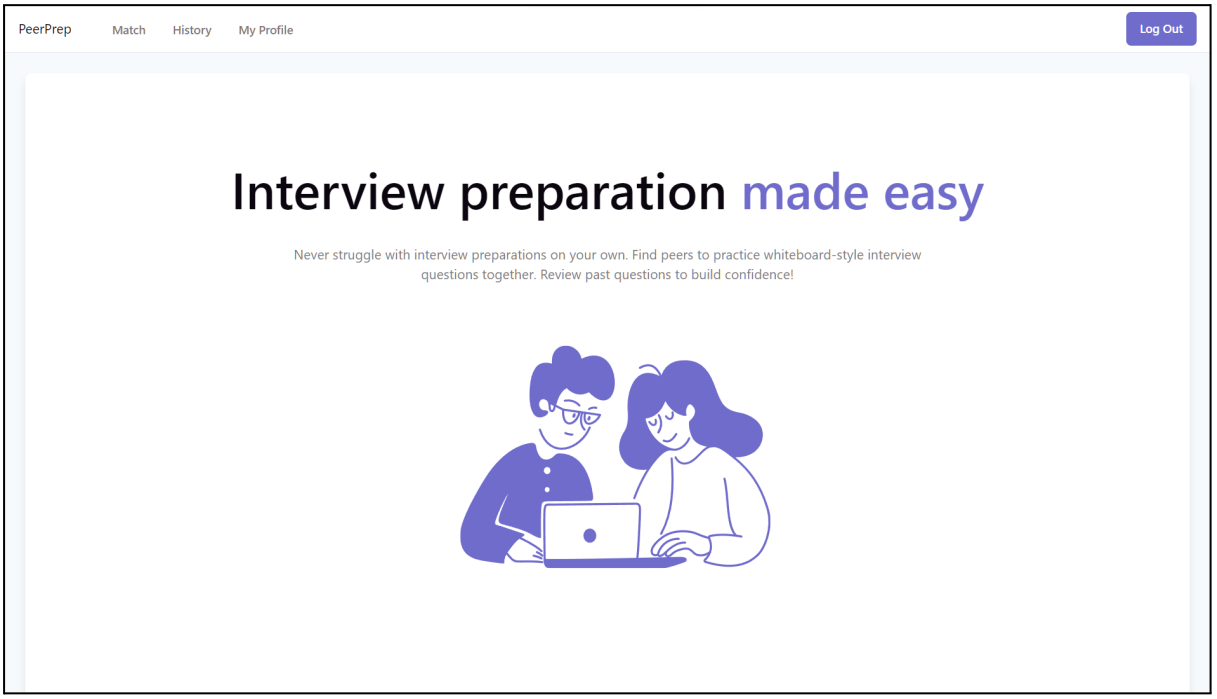

Figure 18: Landing page for Users

11.2 Register Page

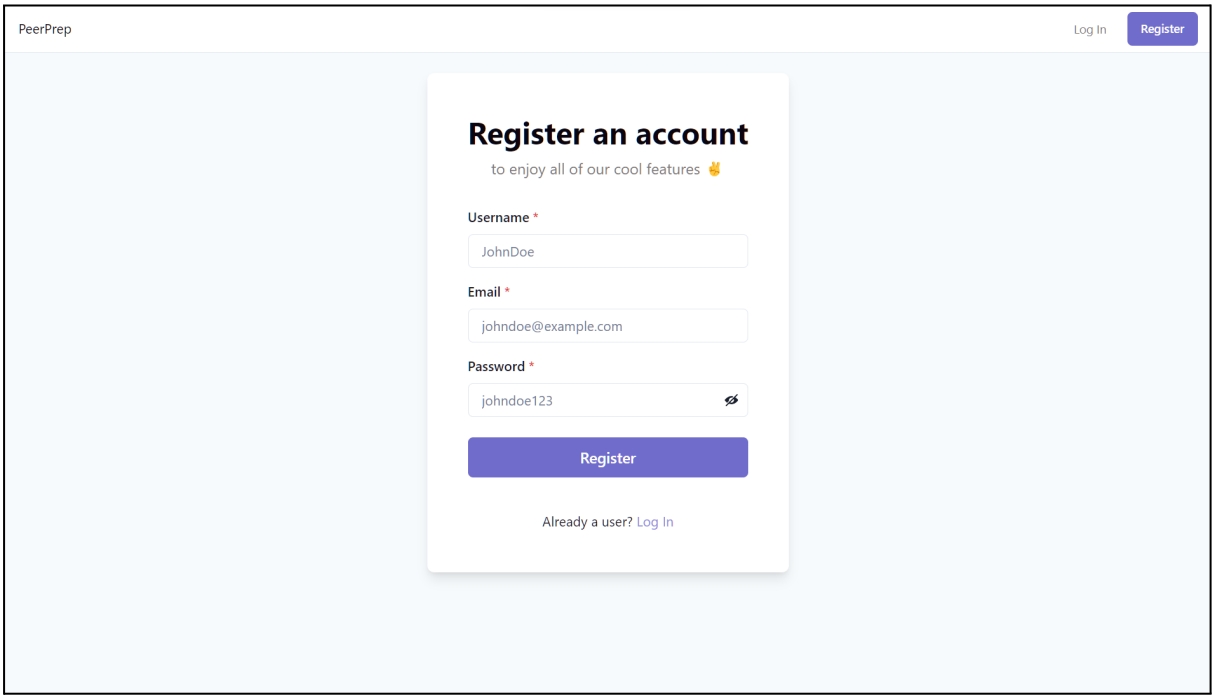

Figure 19: Allow users to register for an account with a unique username and email and set a password

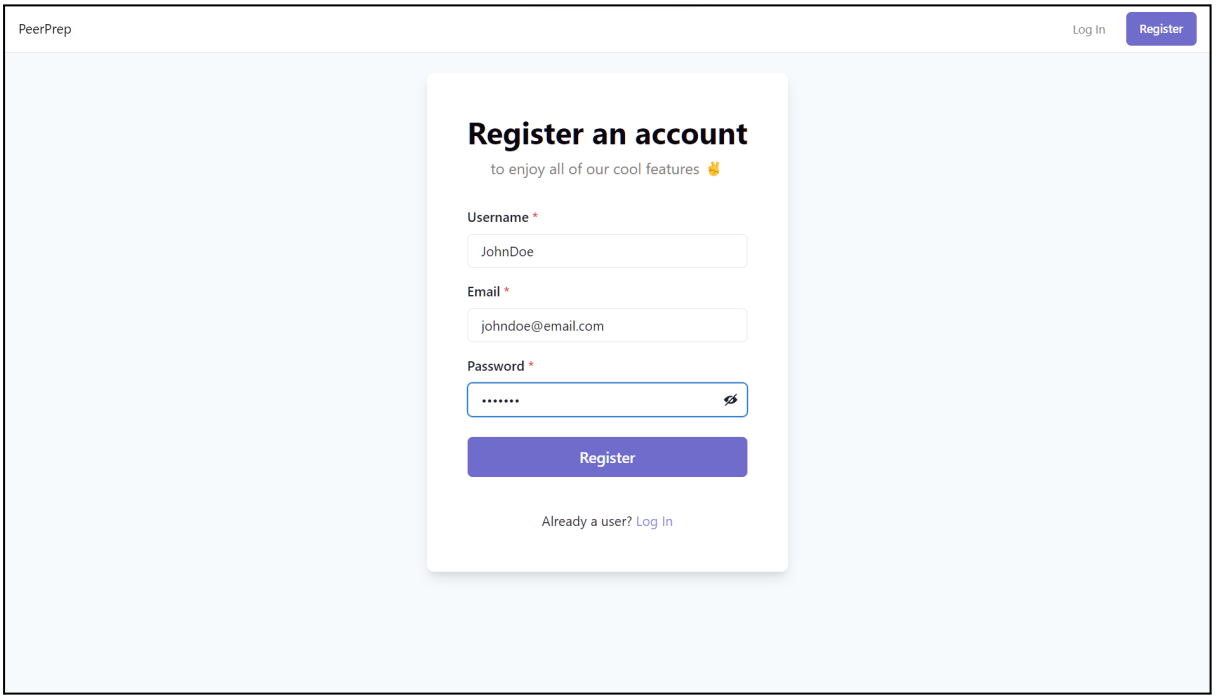

Figure 20: Passwords are hidden

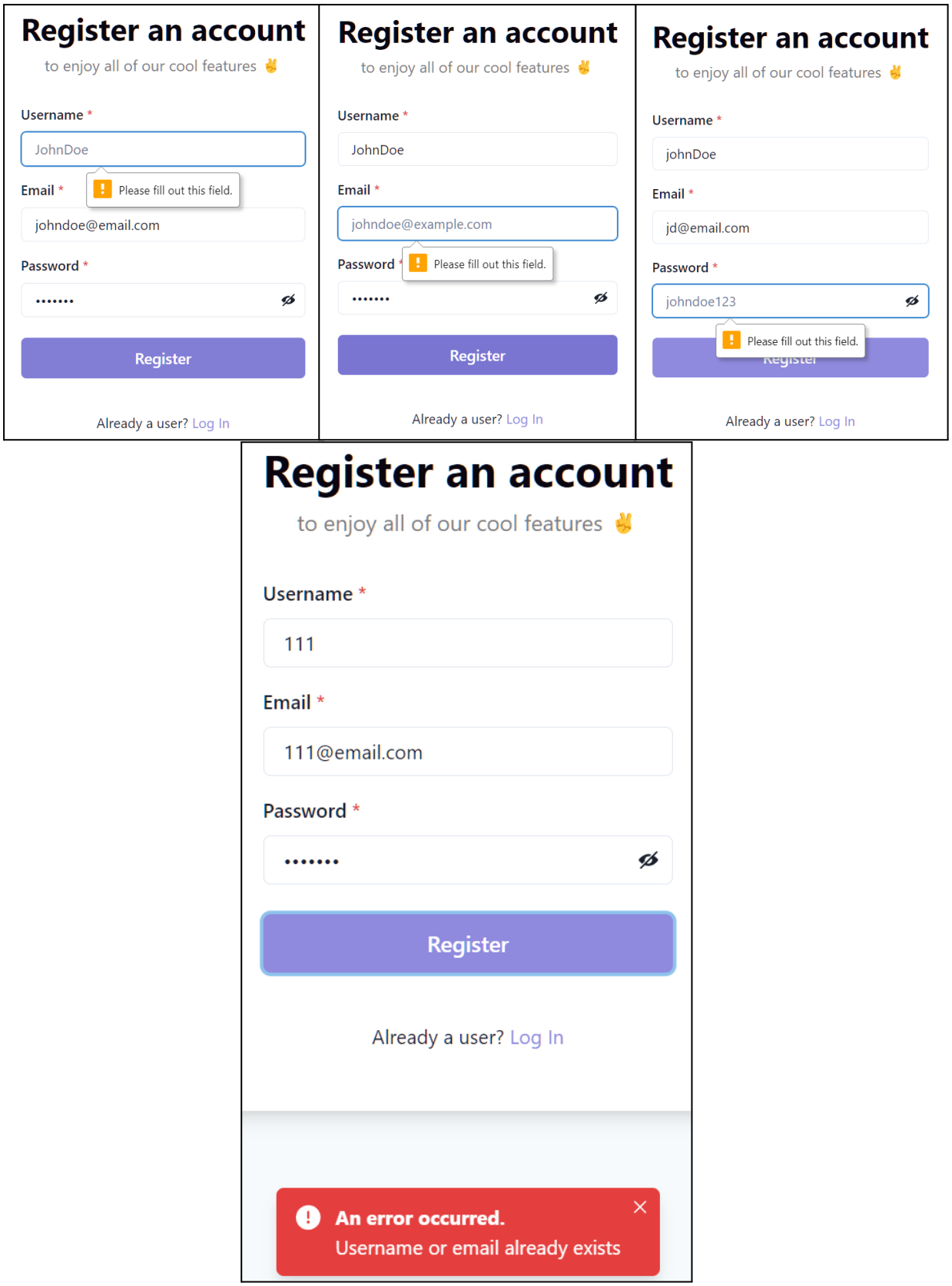

Figure 21: Variety of error validations and messages such as blank fields and existing account details.

11.3 Login Page

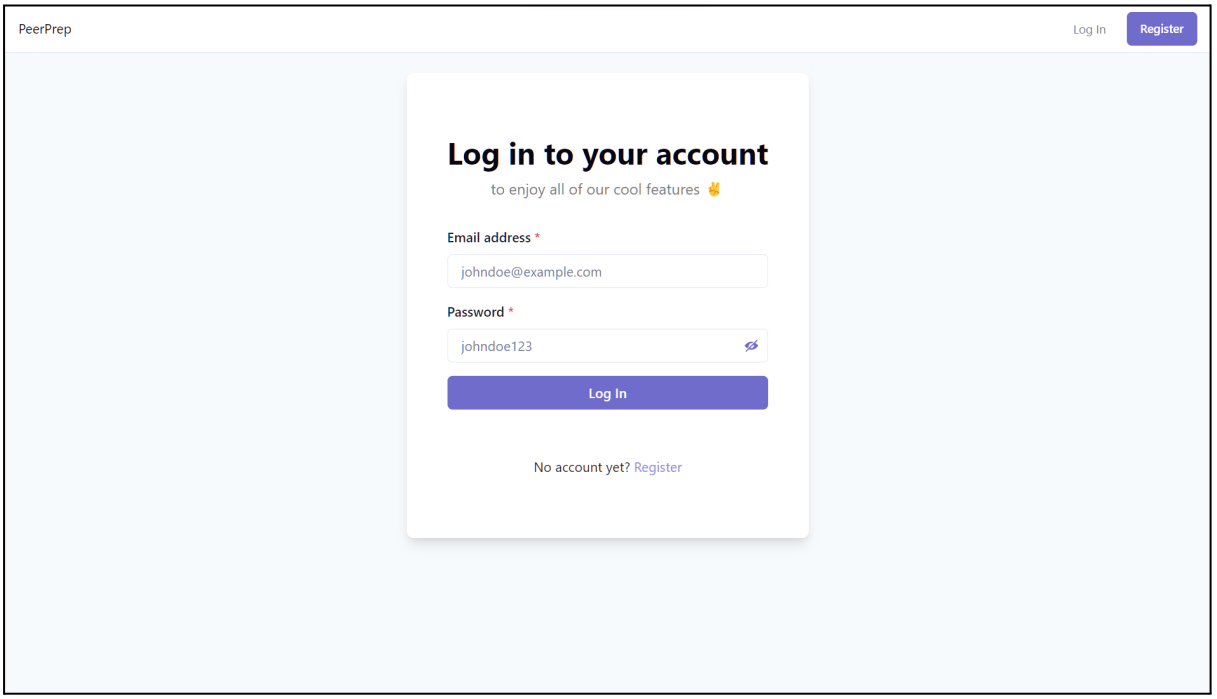

Figure 22: Allow existing users to login with email address and password

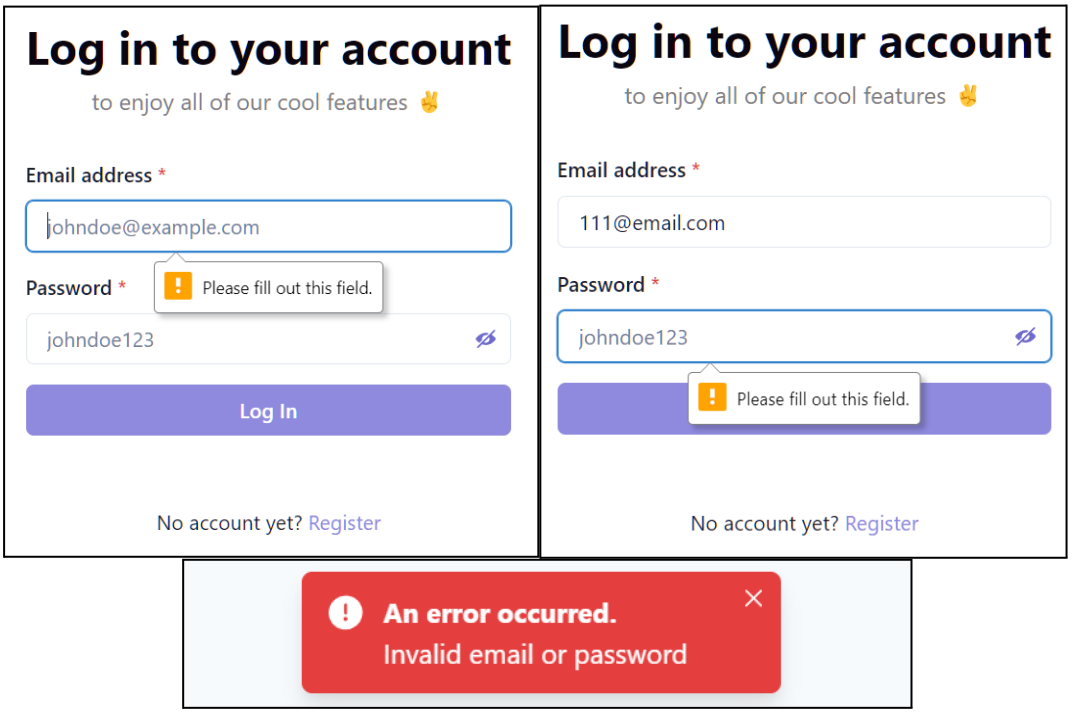

Figure 23: Variety of error validations and messages

11.4 User Profile Page

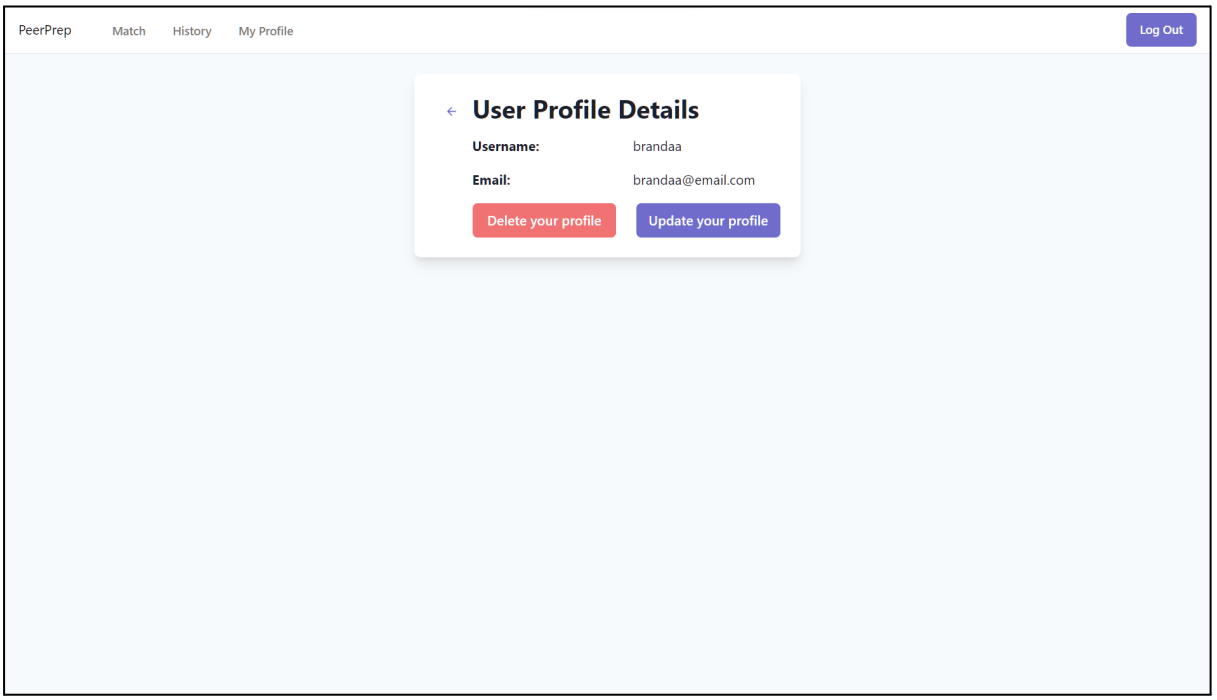

Figure 24: Allows users to see their account details

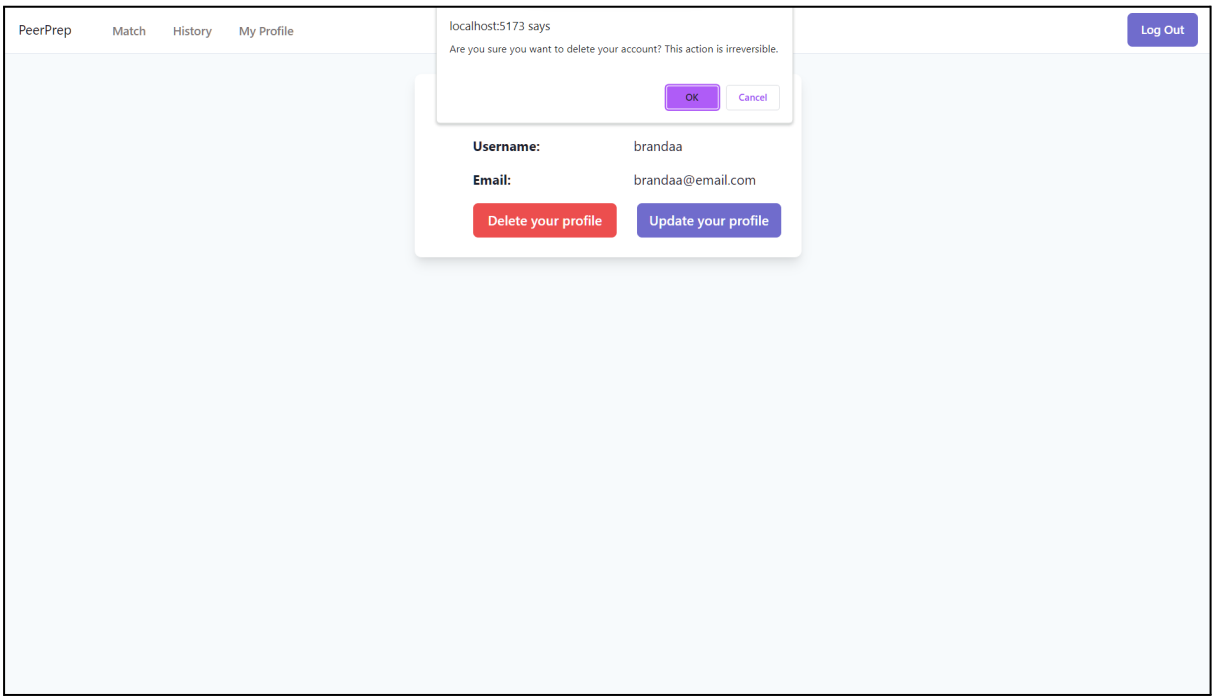

Figure 25: Allow users to delete their account with a confirmation window

11.5 User Update Page

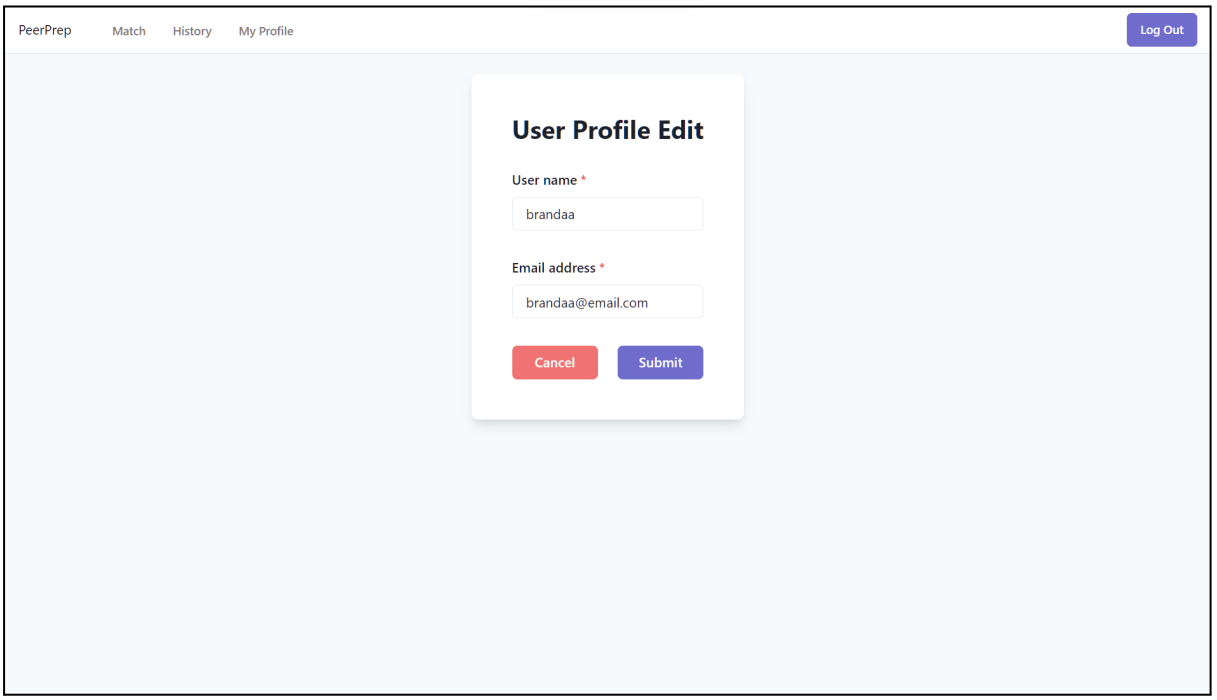

Figure 26: Allow users to update username and/or email

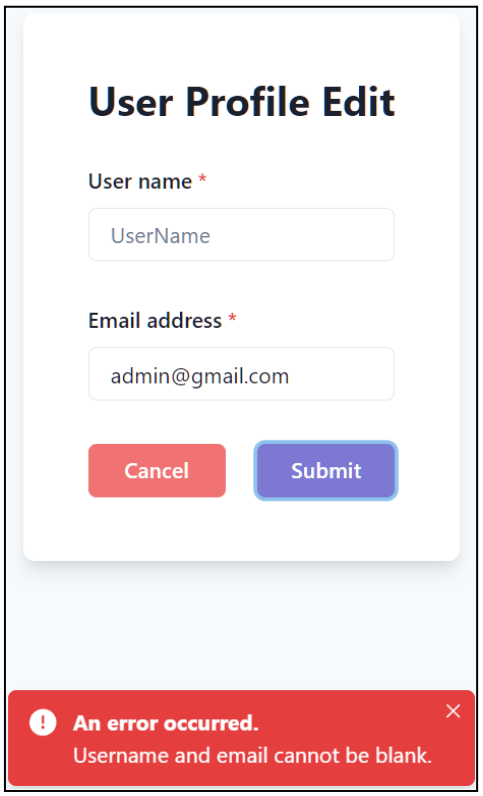

Figure 27: Validation check

11.6 Browse Questions Page (Admin)

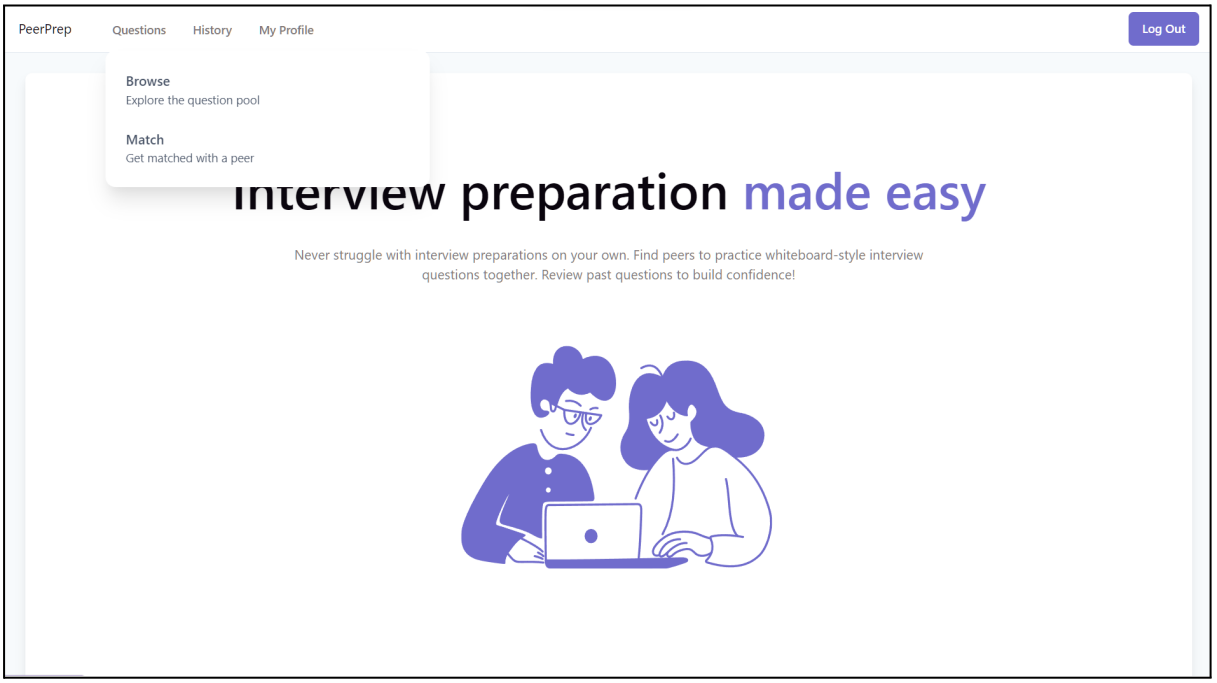

Figure 28: Admin users' navigation bar shows link to browse questions page

11.6.1 View Questions

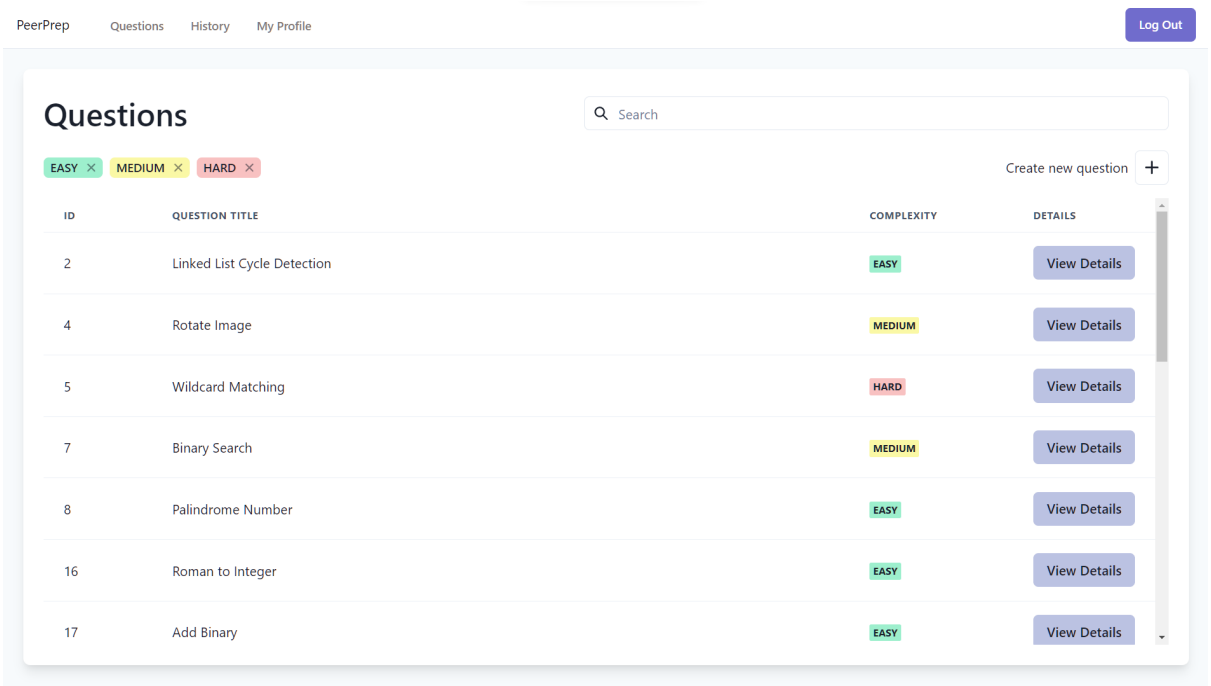

Figure 29: Allow admin users to browse the question bank

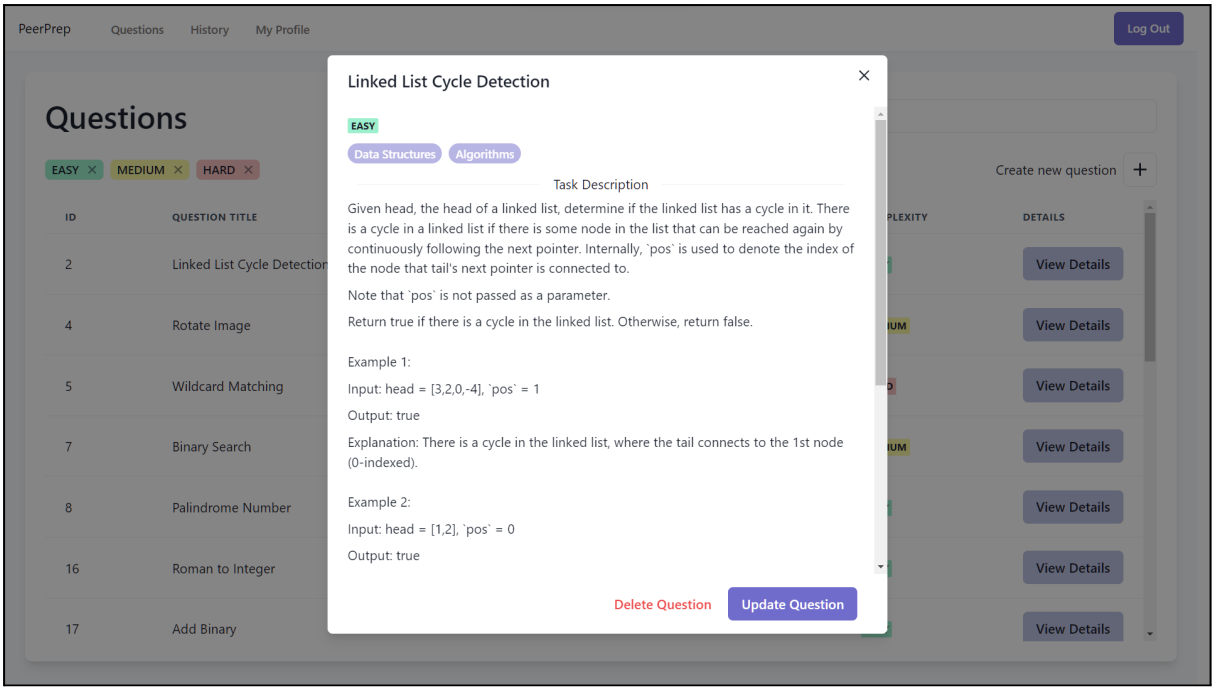

Figure 30: Allow admin users to view full details of questions

11.6.2 Create Question

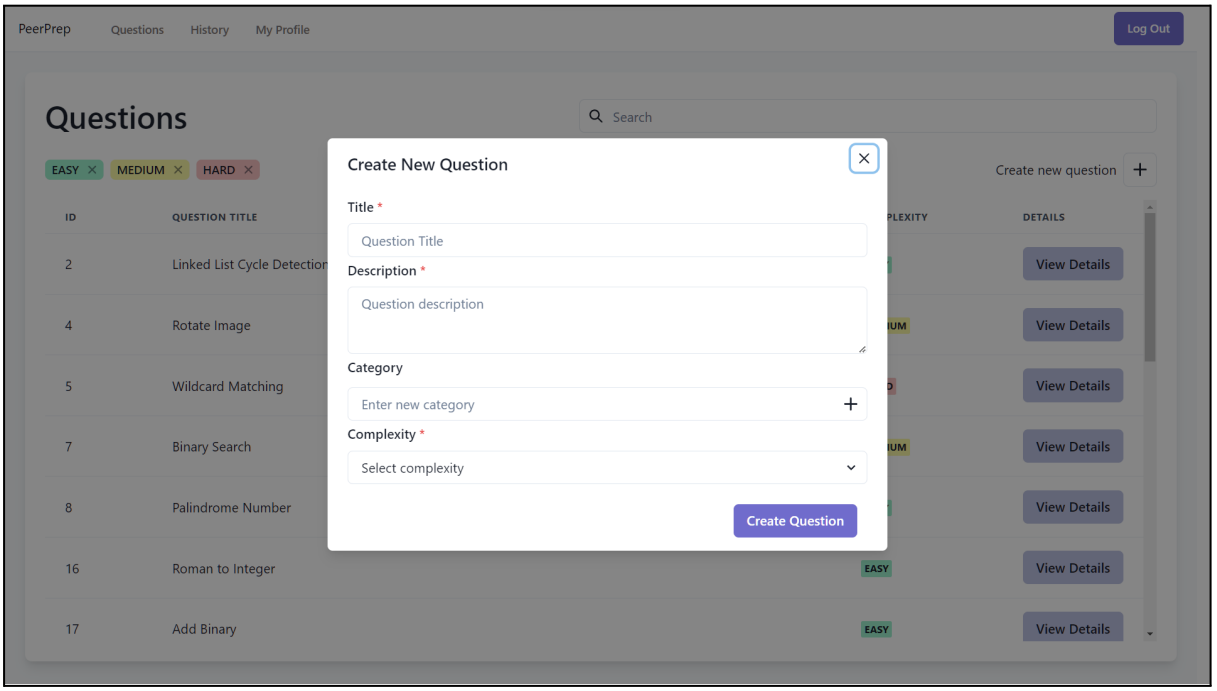

Figure 31: Allow admin users to create new questions

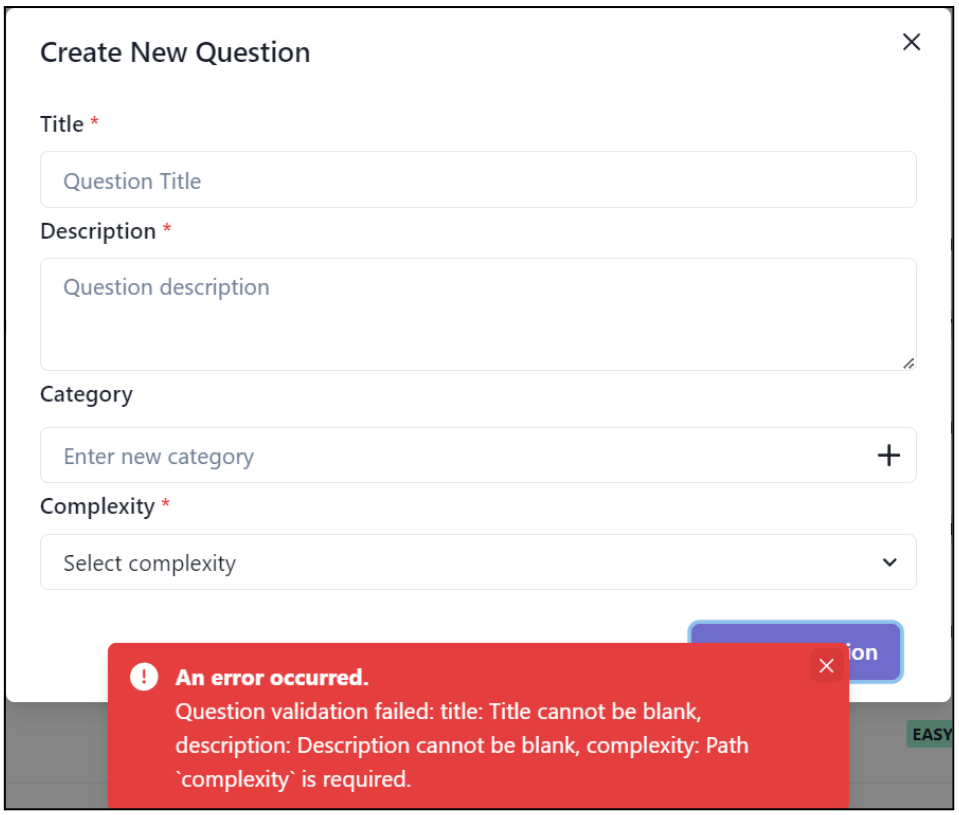

Figure 32: Validation check

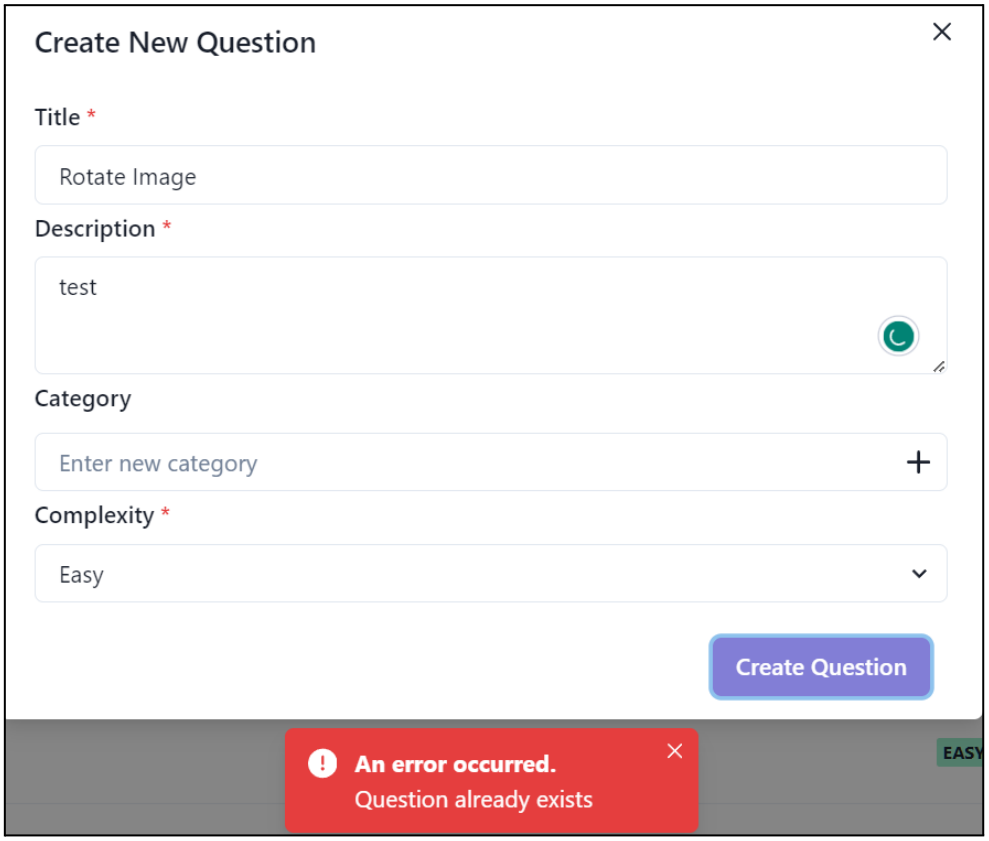

Figure 33: Duplicated questions not allowed

11.6.3 Delete Question

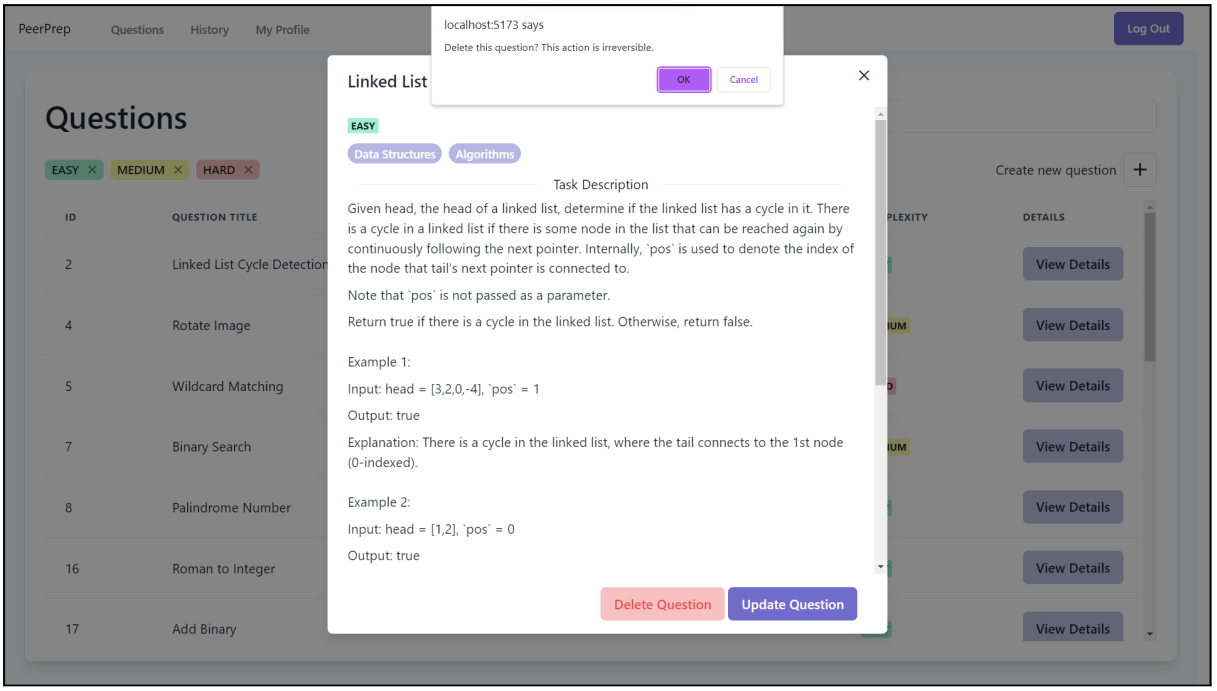

Figure 34: Allow admin users to delete questions

11.6.4 Question Filter and Search

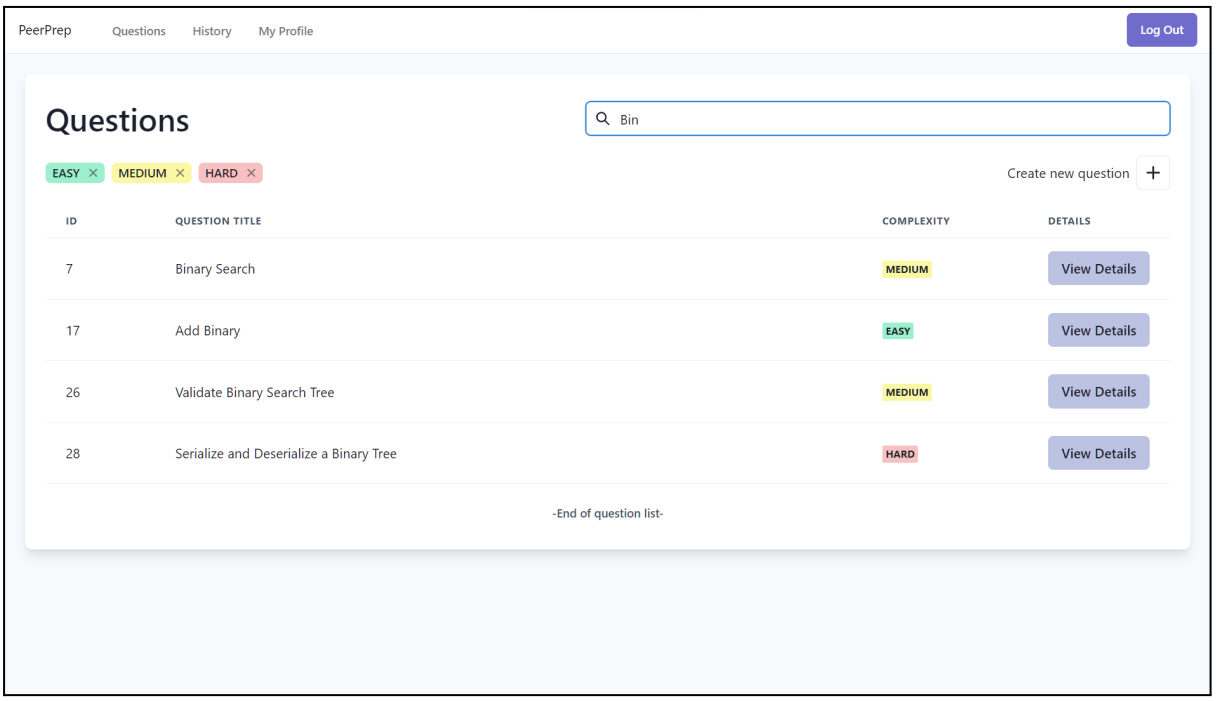

Figure 35: Allow admin users to search for questions

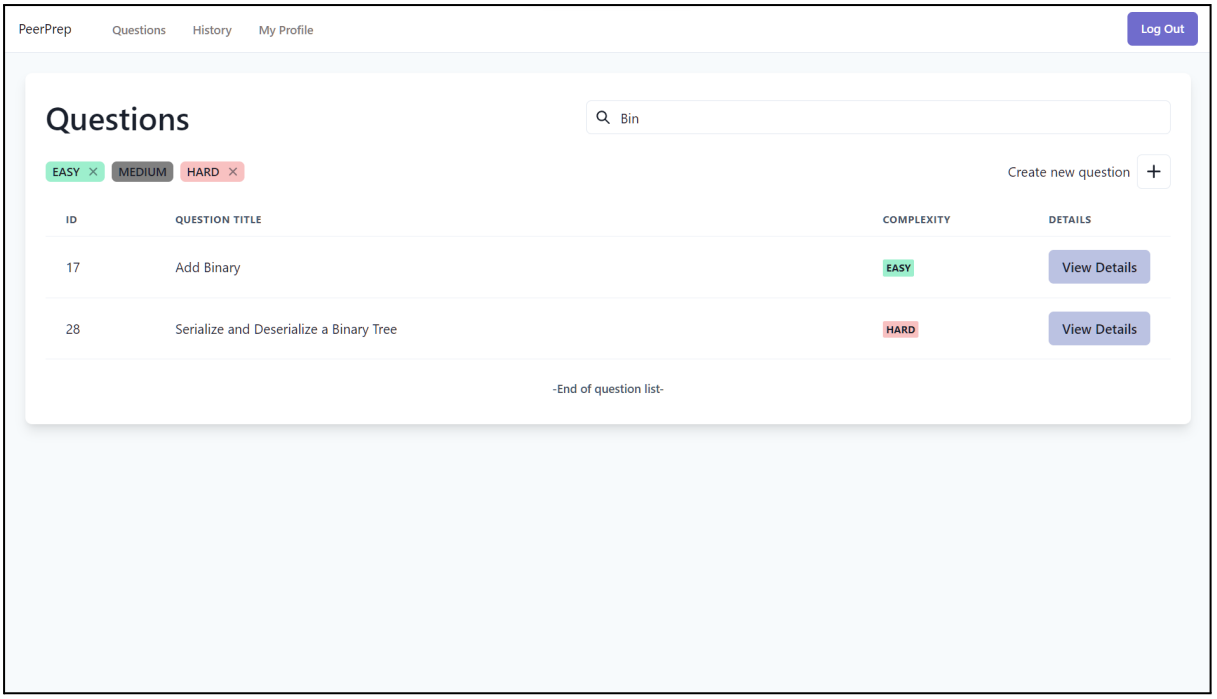

Figure 36: Allow admin users to filter questions by complexity

11.7 Update Question Page (Admin)

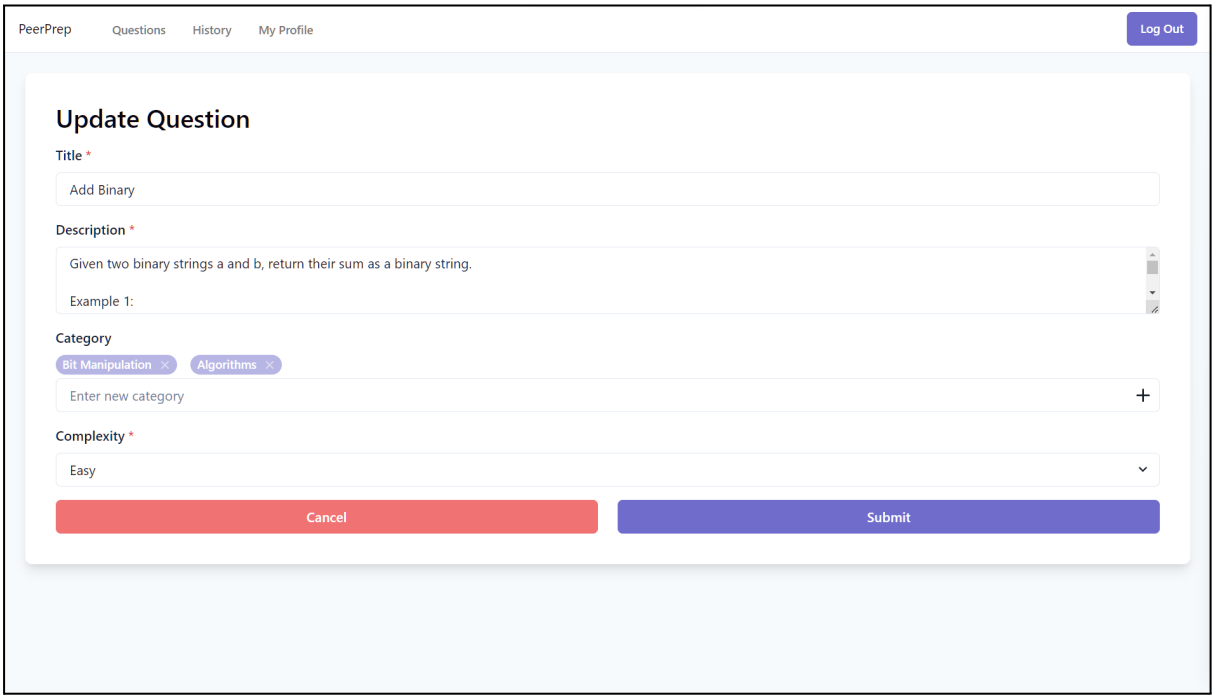

Figure 37: Allow admin users to update questions

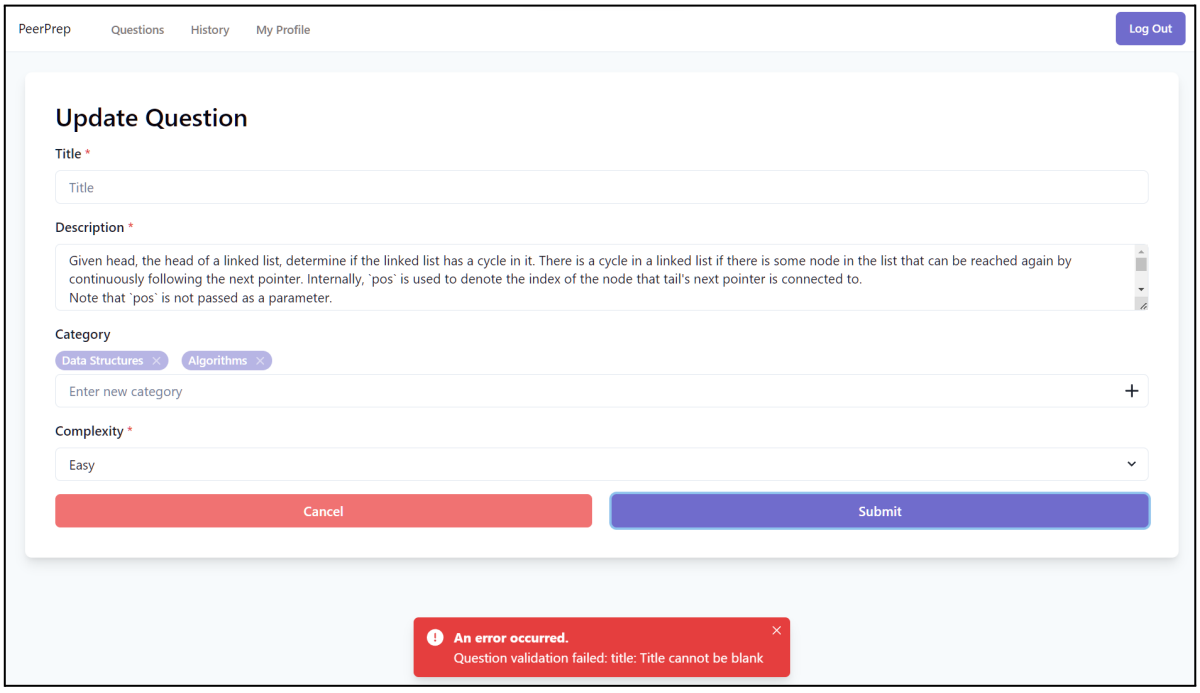

Figure 38: Validation Check

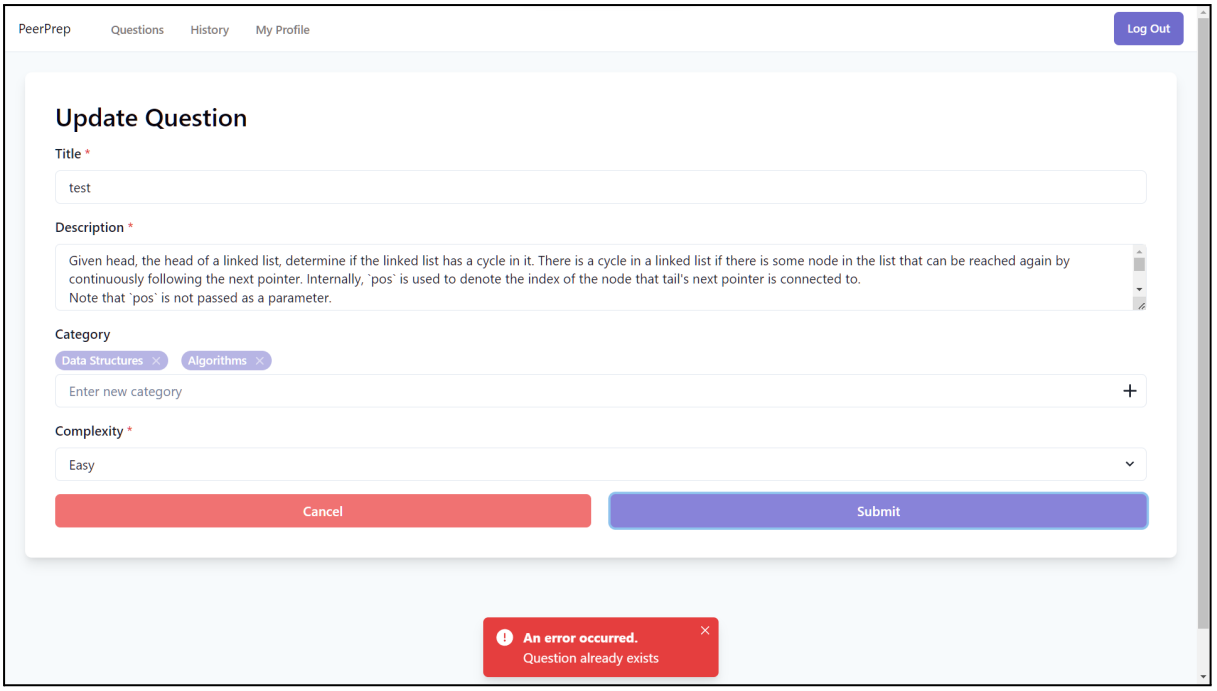

Figure 39: Duplicated questions not allowed

11.8 Matching Modal

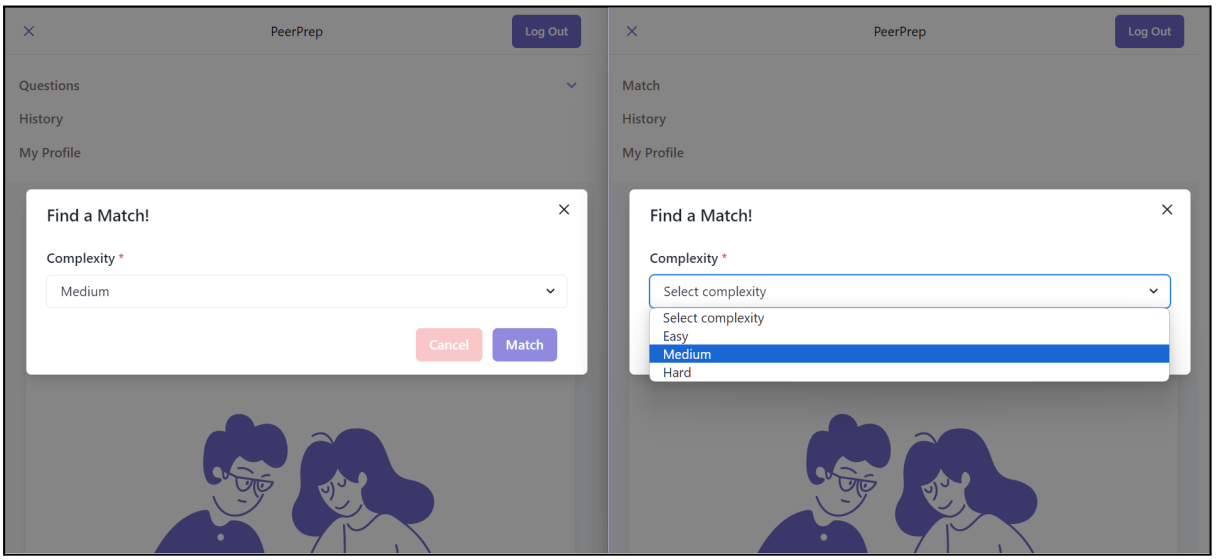

Figure 40: Allow users to match with peers based on difficulty

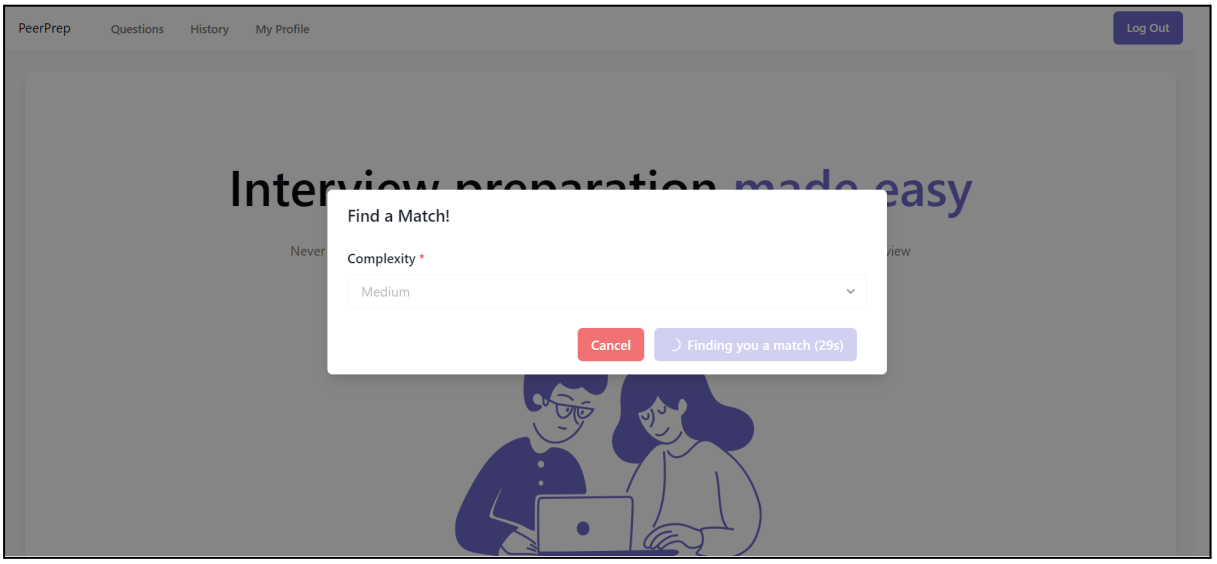

Figure 41: 30 second timer for finding matches. If two users are successfully matched, they are redirected to the session page to start their attempt.

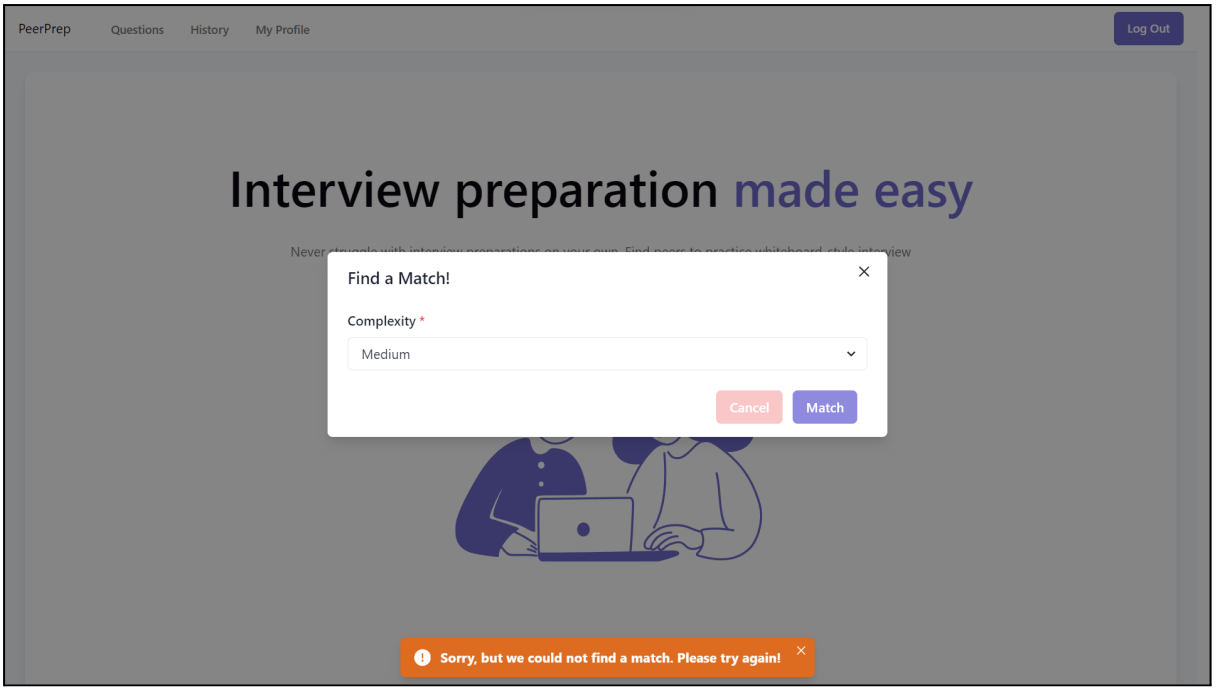

Figure 42: Matching cancels after 30 seconds.

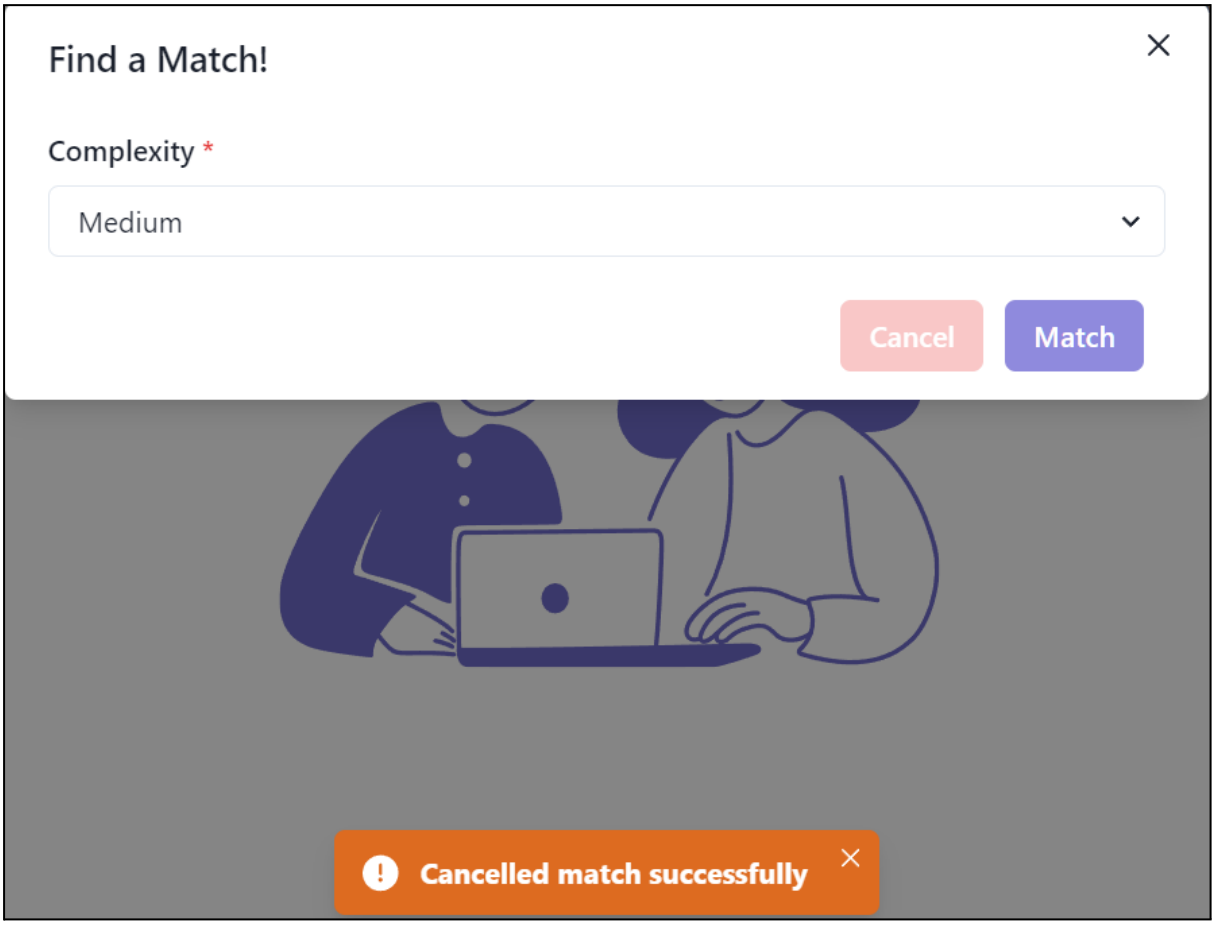

Figure 43: Users can cancel matching.

11.9 Session Page

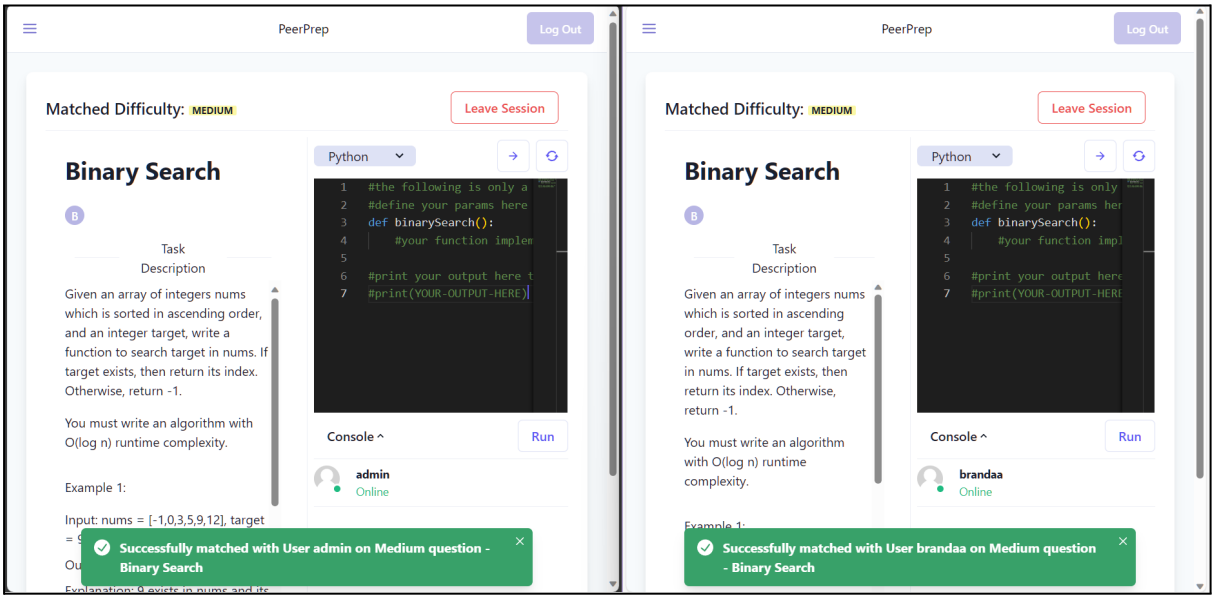

Figure 44: Random question of selected complexity is chosen.

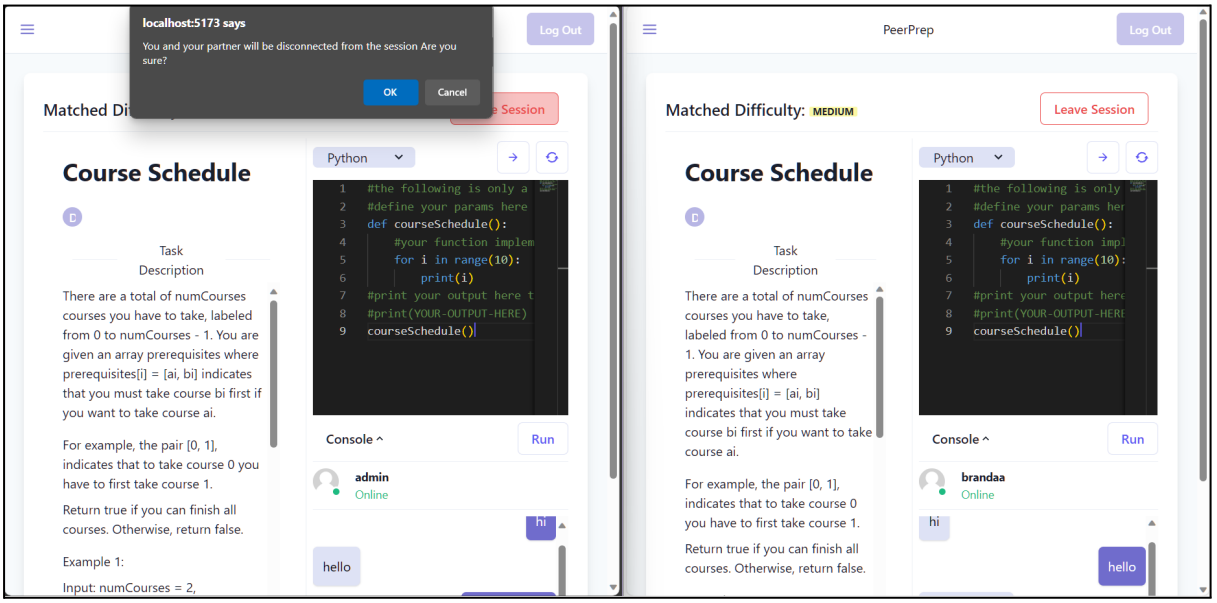

Figure 45: Users can leave a session. This will kick their peer out of the session as well.

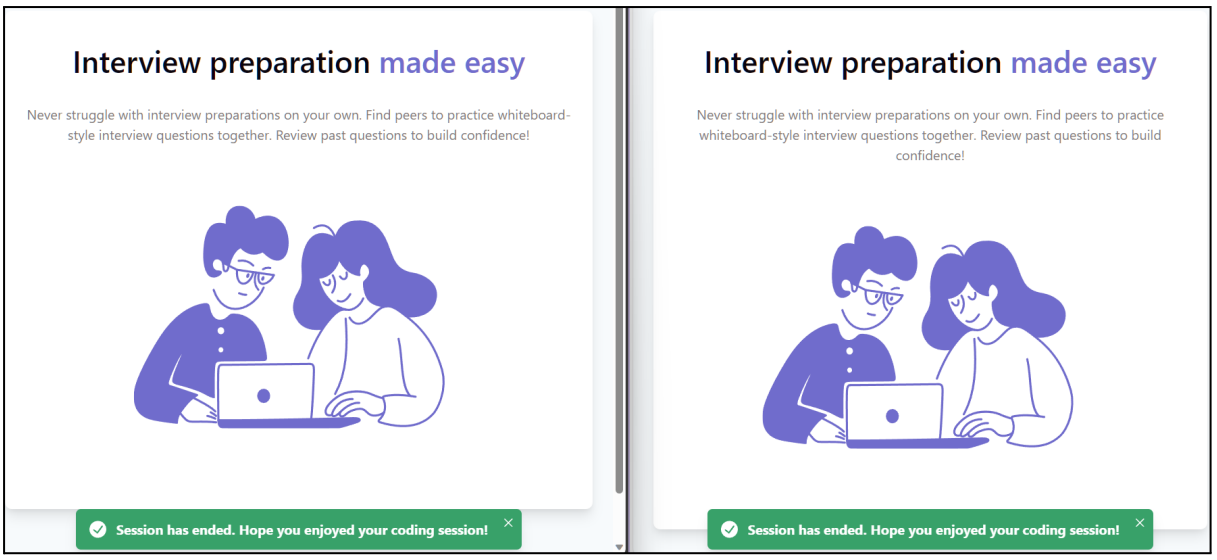

Figure 46: Users will return to the previous page.

11.9.1 Code Editor

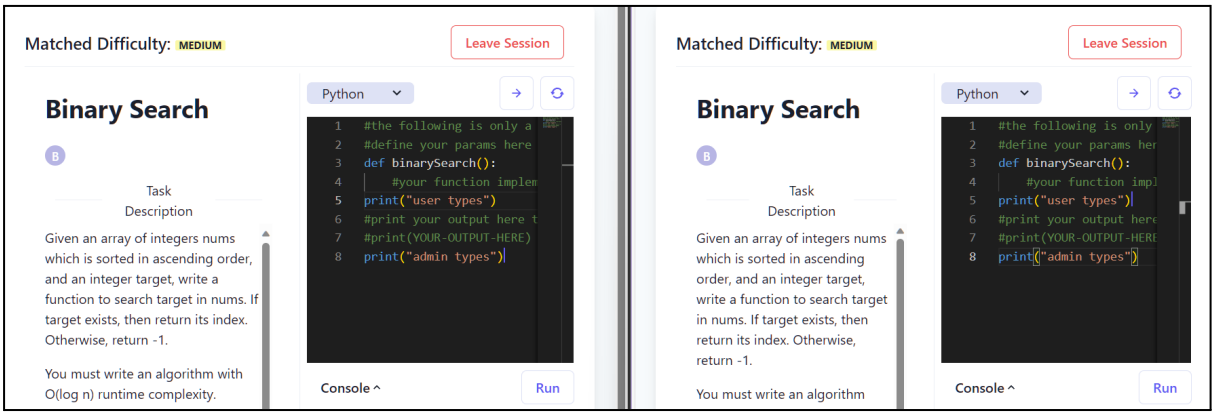

Figure 47: Users are able to code concurrently with their peer and are able to see their peer's cursor

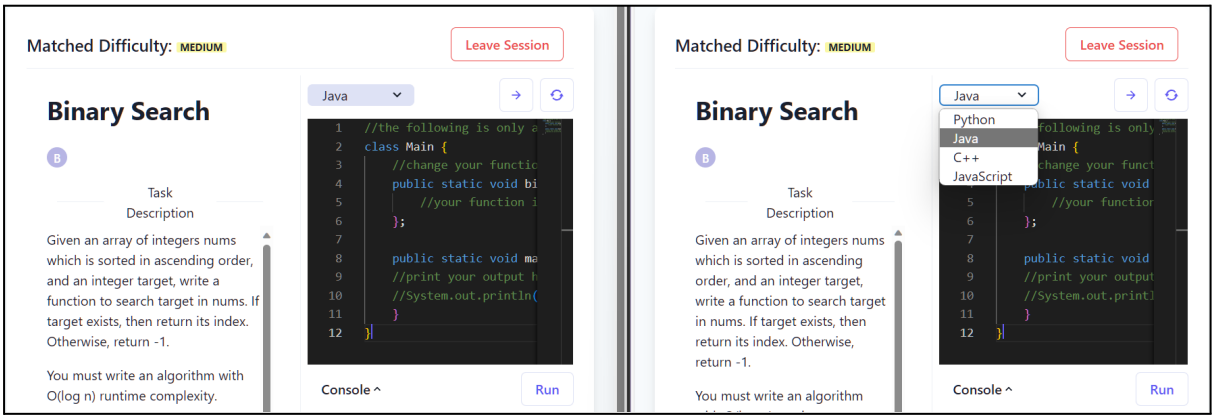

Figure 48: Users are able to change programming languages. (code will persist for each language)

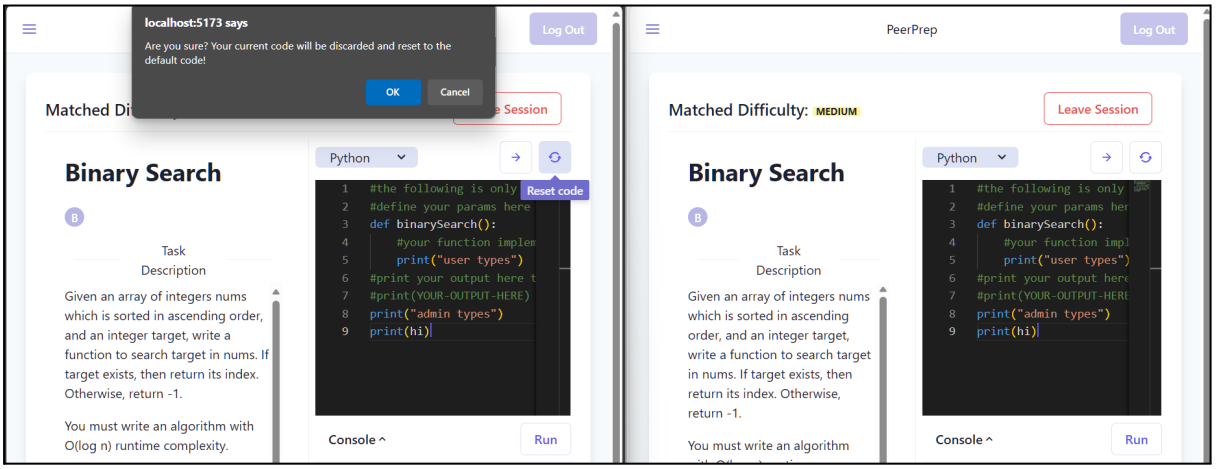

Figure 49: Users can reset the code back to the original template

11.9.2 Code Execution

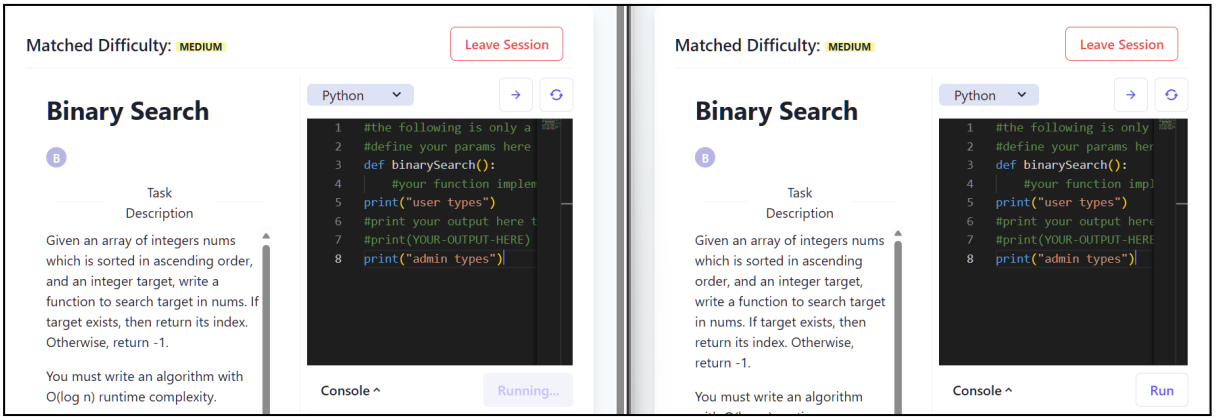

Figure 50: Users are able to run their code independently to see the output

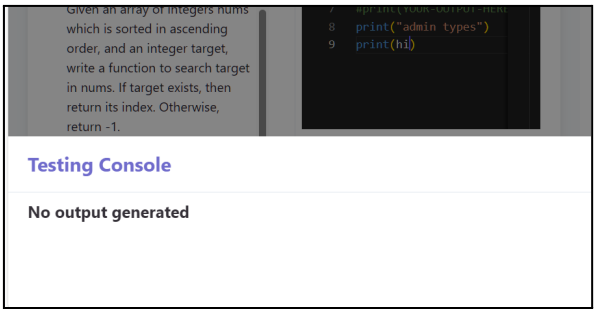

Figure 51: Console for testing code

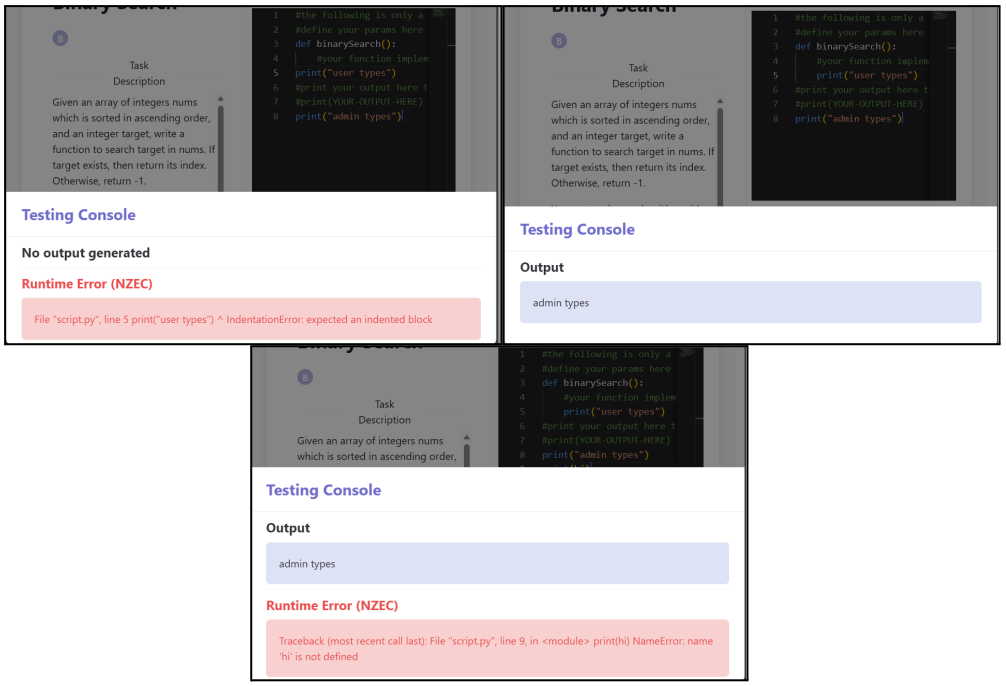

Figure 52: Console will open to show output and/or errors

11.9.2 Chat Box

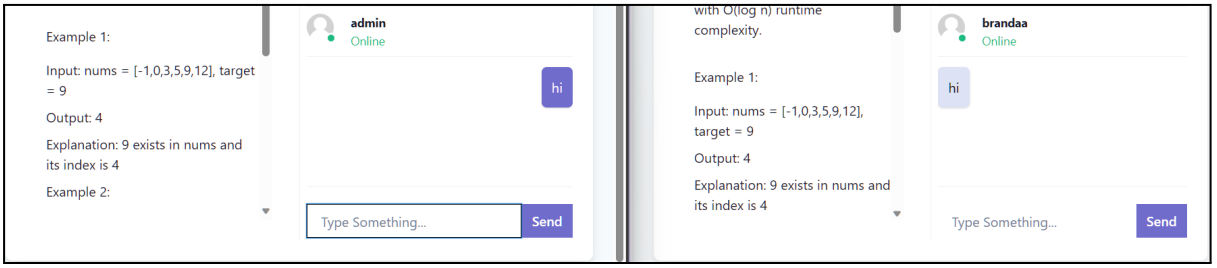

Figure 53: Users can send and receive messages in real-time

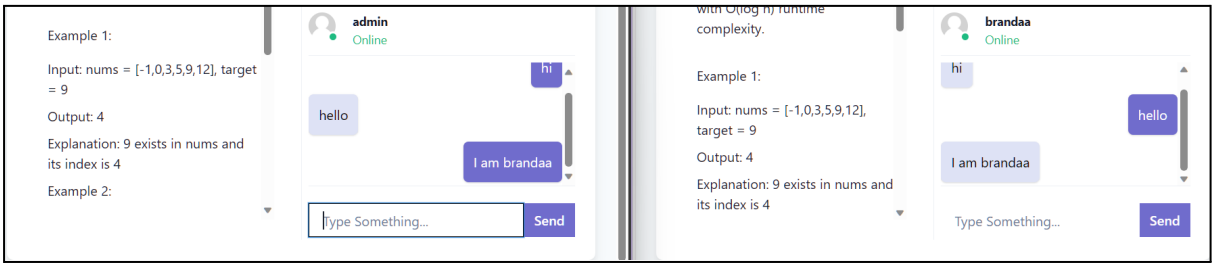

Figure 54: Users can scroll to see previous messages

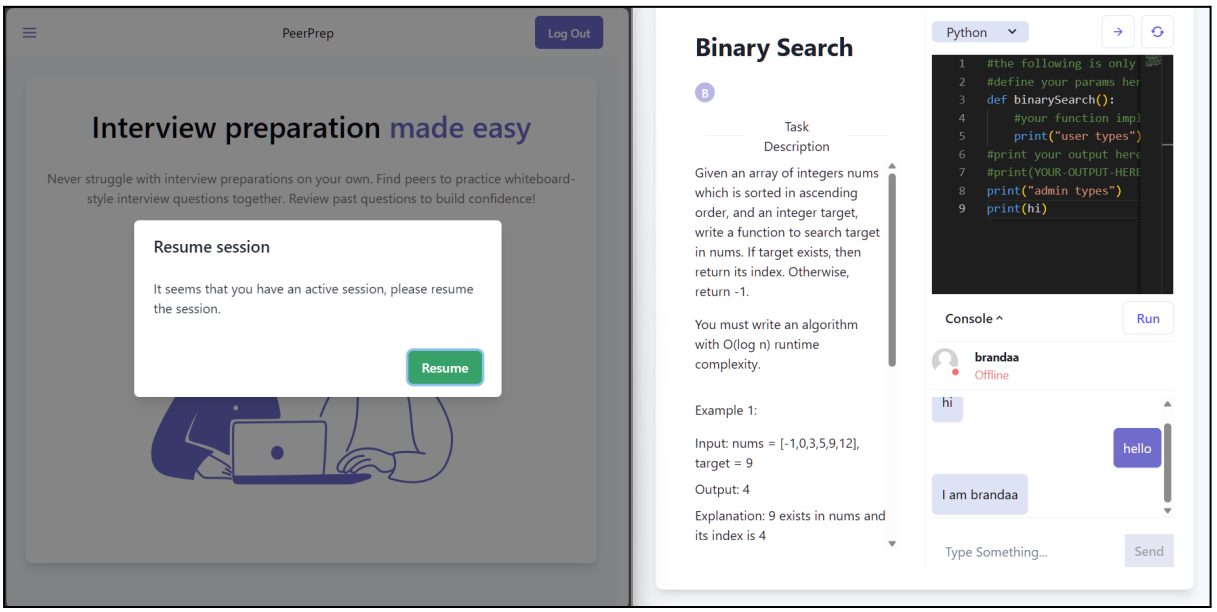

Figure 55: When one peer accidentally leaves the room, they will be prompted to resume the session. The chatbox will show that the peer is not in the room and will disable the sending of messages until the peer is back in the room.

11.9.3 Retrieve new question

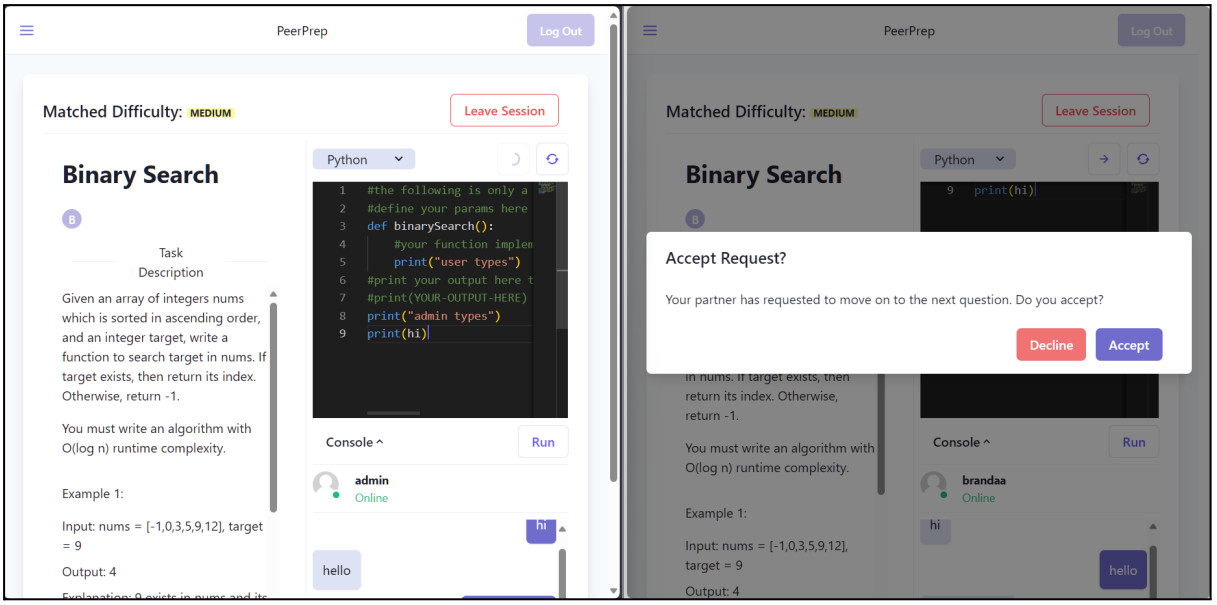

Figure 56: Users can request to move on to the next question with the same peer. The question picked will be of the same complexity level as the matched level.

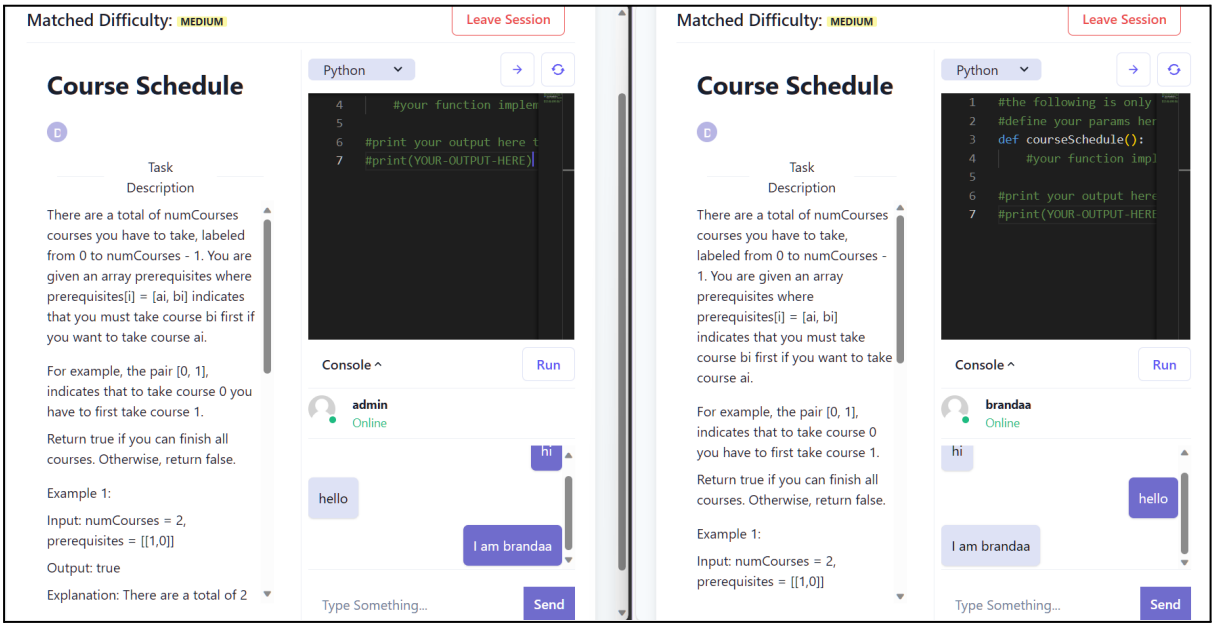

Figure 57: A new question is retrieved while the chatbox persists

11.10 History Page

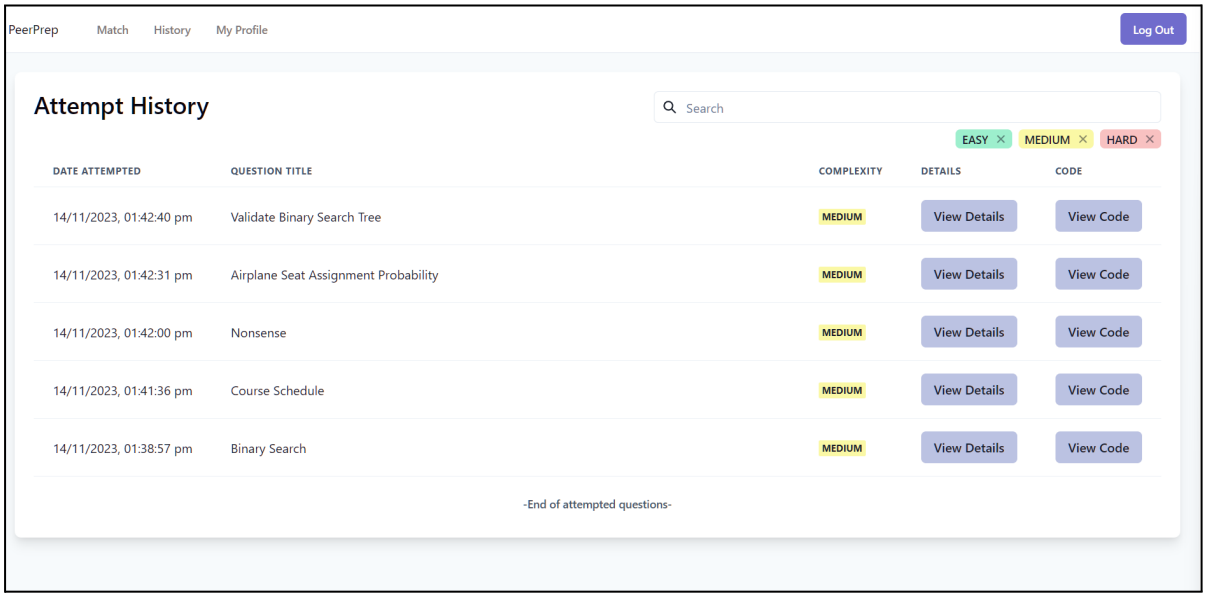

Figure 58: Users are able to view their past attempts. (with search and filter functionality like Browse Questions Page)

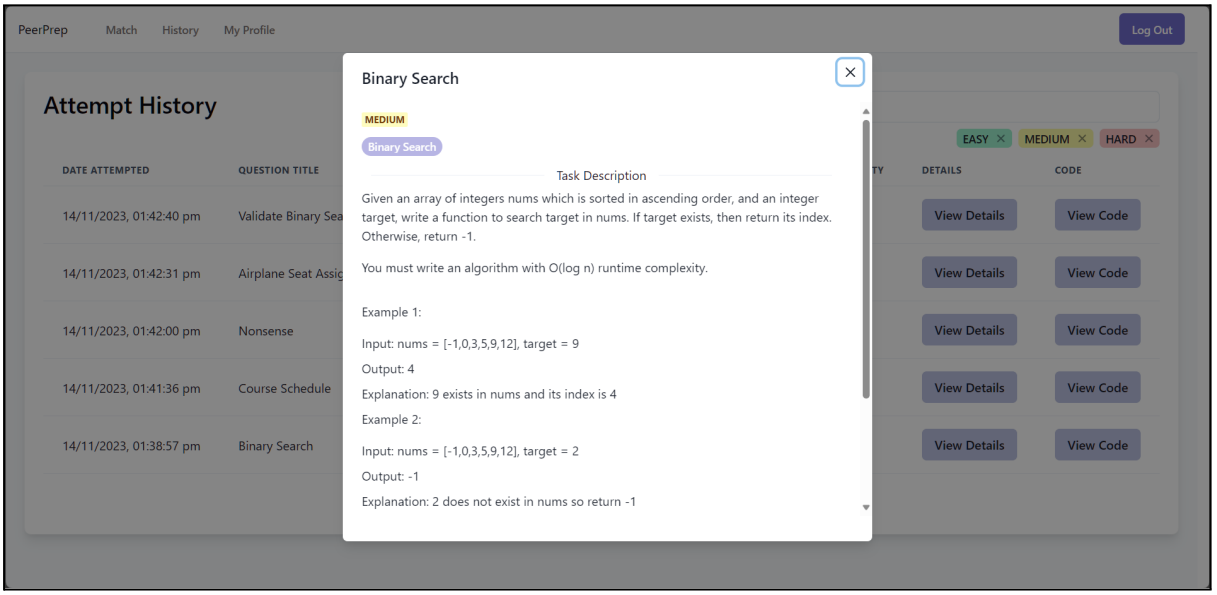

Figure 59: Users can view question details of their attempts

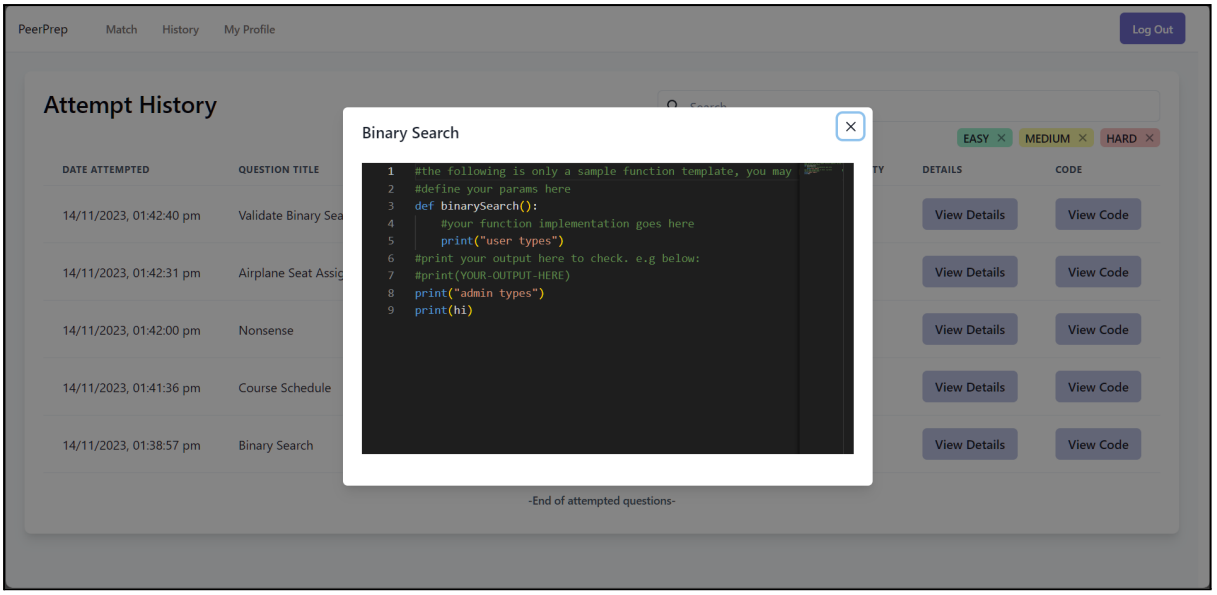

Figure 60: Users can view the latest state of their code editor. (Read-only)

12. Suggested Enhancements

This project holds significant potential for further development. While engaged in its implementation, we identified certain recommended enhancements that we consider pivotal improvements.

1. The security and personalization of PeerPrep can be bolstered by implementing more nice-to-have features in our user service. For instance, we can include email verification during signup to ensure the integrity of user accounts. A secure password reset method would also enhance user-centric reliability and safeguard against potential data loss. Additionally, enabling sign-ups through popular platforms like GitHub and Google streamlines registration, offering a familiar and tailored experience for our targeted users.

2. PeerPrep's collaboration service can also be elevated with two key enhancements. Firstly, a new functionality that can allow users to retrieve questions based on specific topics would enable a more tailored collaborative experience aligned with individual interests and learning goals. This adds flexibility to the collaborative coding process, addressing unique preferences and enhancing overall engagement. Secondly, the introduction of video call functionality goes beyond the existing chat feature, providing users with a more comprehensive communication experience. This face-to-face connection fosters quicker issue resolution and a deeper understanding, further enriching the collaborative coding interactions on the platform.

3. Our DevOps processes also have potential for improvement. Leveraging additional features from NGINX, such as rate limiting, authentication, and automatic SSL termination, will bolster the security and comprehensiveness of our API management solution. Exploring service mesh technologies like Istio is also on the horizon, as it can provide robust traffic management, security, and observability capabilities. Furthermore, as our application and team continue to scale, the introduction of CI/CD pipelines to Github is crucial, whether through Github Actions or custom scripts This enhancement will empower our team to develop, test, and deploy more efficiently, ensuring a seamless and agile development process.

13. Reflections and Learning Points

1. Learning collaboratively through efficient communication has been a valuable experience. We've discovered that there is a wealth of knowledge within the group, and by simply asking, someone often has the answers to our questions. This process not only fosters mutual learning but also contributes to each individual's growth, helping us become better software engineers.

2. It is important to start integration and deployment early. In our case, development was largely done locally. However, we encountered many other issues during deployment that were not relevant when developing locally. Rerouting, environment variables, and ingress resources are just some of these issues. This serves as a valuable lesson for upcoming tasks and applies to our professional endeavours, helping us steer clear of last-minute challenges. Nevertheless, the deployment of our application to a production system served as a significant learning opportunity for our entire team, as it marked our inaugural experience in this domain.

3. Maintaining incremental documentation is crucial. We need to go beyond code and emphasise conveying the significance of their programs. This practice will enhance communication within the team and facilitate effective communication with stakeholders in subsequent stages.

Overall this journey has been beneficial for the entire team. We've gained insights from each other's experiences and learned from our mistakes. Here's to creating more with our newfound skills!