



CS3219

Software Engineering Principles and Patterns

AY 2022/2023 Sem 1

FINAL REPORT

Team 7

| Members | Matriculation Number |
|----------------|-----------------------------|
| Kerk Pei Yong | A0197079L |
| Kenneth Eer | A0199705N |
| Tey Xing Rong | A0201708N |
| Kwek Zhan Hao | A0206238J |

Table of Content

| | |
|--|-----------|
| 1. Introduction | 4 |
| 1.1 Purpose | 4 |
| 1.2 Background | 4 |
| 1.3 Contributions | 5 |
| 2. System Features and Requirements | 7 |
| 2.1 Functional Requirements | 7 |
| 2.1.1 User Service | 7 |
| 2.1.2 Matching Service | 7 |
| 2.1.3 Collaboration Service | 8 |
| 2.1.4 Question Service | 8 |
| 2.1.5 Webcam Service | 8 |
| 2.2 Non-Functional Requirement | 9 |
| 2.2.1 Security Requirements | 9 |
| 2.2.2 Usability Requirements | 9 |
| 2.2.3 Performance Requirements | 10 |
| 3. High-level Architecture | 11 |
| 3.1 High-level diagram | 11 |
| 3.2 Model-View-Controller | 12 |
| 3.3 4-tier architecture diagram | 13 |
| 4. Design of the Key Components | 15 |
| Facade Pattern | 15 |
| Repository Pattern | 15 |
| Publish Subscribe Pattern | 16 |
| 4.1 User Service | 17 |
| 4.1.1 Design of User Service | 17 |
| 4.1.2 Rationale | 19 |
| 4.2 Matching Service | 20 |
| 4.2.1 Design of Matching Service | 20 |
| 4.2.2 Rationale | 22 |
| 4.3 Collaboration Service | 23 |
| 4.3.1 Design of Collaboration Service | 23 |
| 4.3.2 Rationale | 24 |
| 4.4 Question Service | 24 |
| 4.4.1 Design of Question Service | 24 |
| 4.4.2 Rationale | 26 |
| 4.5 Webcam Service | 27 |

| | |
|---|-----------|
| 4.5.1 Design of Webcam Service | 27 |
| 4.5.2 Rationale | 30 |
| 4.6 UI/UX enhancement | 31 |
| 4.6.1 Design of UI/UX enhancement | 31 |
| 4.6.2 Design Decision and Implementation | 34 |
| 4.6.2.1 Design of Difficulty Selection | 34 |
| 4.6.2.2 Rationale | 35 |
| 4.6.2.3 Feature Implementation Design Consideration | 35 |
| 5. DevOps Practices | 36 |
| 5.1 Continuous Integration (CI) | 36 |
| 5.2 Continuous Deployment (CD) | 36 |
| 5.3 Deployment Details | 37 |
| 6. User Interface | 39 |
| 7. Improvements and enhancements | 46 |
| 7.1 Question Service: Increase variety of questions and viewing question bank | 46 |
| 7.2 Allow for the compilation of code + Running test cases | 46 |
| 7.3 Quality of life features for matching service | 46 |
| 7.4 Deployment of Application | 46 |
| 8. Project Management | 47 |
| 9. Reflection | 48 |
| 10. Others | 48 |

1. Introduction

1.1 Purpose

The final report outlines the development process and design decisions made during the course of our project. Through the report, we highlight the design of the individual components of the software as well as the requirements of the product.

1.2 Background

Increasingly, students face challenging technical interviews when applying for jobs which many have difficulty dealing with. Issues range from a lack of communication skills to articulate their thought process out loud to an outright inability to understand and solve the given problem. Moreover, grinding practice questions can be tedious and monotonous. PeerPrep aims to solve this issue, where students can find peers to practice whiteboard-style interview questions together.

Today, PeerPrep offers the following notable features.

1. Authentication and user account management
2. Selecting questions of different difficulty
3. Matching of users who have selected questions of similar difficulty
4. Real time coding in desired programming language
5. Collaboration through webcam and audio
6. View history of questions completed for revision

These features aim to help users easily prepare and revise for technical interviews.

1.3 Contributions

| Name | Technical Contributions | Non-technical contributions |
|---------------|--|--|
| Kerk Pei Yong | <p>MVP of matching service and pairing of the users, including the loading screen</p> <p>Backend of question service</p> <ul style="list-style-type: none"> - Retrieval of question when the users are paired - Retrieval of the history of questions attempted by user. - Other CRUD api end points to support creation of question <p>MVP of frontend for history of attempts</p> <p>Fix bugs</p> | <p>UML diagrams for question service</p> <p>Project management</p> <p>Contributed to the final report</p> |
| Kenneth Eer | <p>Backend of user service</p> <ul style="list-style-type: none"> - Implement CRUD API endpoints to handle signup, login, logout, delete user, change password, with user validation - JWT, Password Hashing <p>UI/UX (Frontend)</p> <ul style="list-style-type: none"> - Design the overall theme, font colours and styling of all app pages and layout - Optimise pages with animations, error/confirmation dialogs and navbar routing and improve user interaction for question difficulty selection <p>Deployment of collaboration service</p> <p>Fix bugs</p> | <p>UML diagrams</p> <p>Project management</p> <p>Contributed to the final report</p> |
| Tey Xing Rong | <p>Backend of matching service</p> <ul style="list-style-type: none"> - Matching two users with the same chosen difficulty and notifying them in real time when match is made - Notifying users when they are unable to match in 30 seconds <p>Frontend and backend of collaboration service</p> <ul style="list-style-type: none"> - Styling of code editor - Implement operational transformation algorithm for collaborative editing | <p>UML diagrams for matching service</p> <p>Project management</p> <p>Contributed to the final report</p> <p>Continuous Integration and Continuous Deployment to</p> |

| | | |
|---------------|---|---|
| | <p>User service</p> <ul style="list-style-type: none"> - Implementation of http-only cookies <p>Webcam service</p> <ul style="list-style-type: none"> - Socket.IO communication via room id - UI, bugfixing, accounting for different scenarios <p>Testing</p> <ul style="list-style-type: none"> - Added tests for the services for Continuous Integration | AWS cloud services |
| Kwek Zhan Hao | <p>MVP of webcam service, users have to send each other a key to establish connection.</p> <p>Changing of audio and video stream for webcam service.</p> <p>Backend of user service (authentication)</p> <ul style="list-style-type: none"> - Blacklist - Password Hashing - JWT token signing <p>Frontend(authentication)</p> <ul style="list-style-type: none"> - Protected Routes - JWT token storing | <p>Architecture diagrams</p> <p>Project management</p> <p>Contributed to the final report</p> |

2. System Features and Requirements

2.1 Functional Requirements

In this section, the requirements of PeerPrep are outlined. We have ranked the different functional requirements based on their priority levels - High, Medium and Low. We have implemented all the features that serve the functional requirements we have decided is a high priority, as we consider these essential to our core functionalities. Requirements tagged as 'Medium' are great extensions to have and also impactful, while requirements we have marked as 'Low' are perceived to contribute little to the user experience in our product, even though they are important.

2.1.1 User Service

| S/N | Functional Requirement | Priority | Status |
|-------|---|----------|--------|
| FR1.1 | The system should allow users to create an account with username and password. | High | Done |
| FR1.2 | The system should ensure that every account created has a unique username. | High | Done |
| FR1.3 | The system should allow users to log into their accounts by entering their username and password. | High | Done |
| FR1.4 | The system should allow users to log out of their account. | High | Done |
| FR1.5 | The system should allow users to delete their account. | Medium | Done |
| FR1.6 | The system should allow users to change their password. | Medium | Done |

2.1.2 Matching Service

| S/N | Functional Requirement | Priority | Status |
|-------|---|----------|----------|
| FR2.1 | The system should allow users to select the difficulty level of the questions they wish to attempt. | High | Done |
| FR2.2 | The system should be able to match two waiting users with similar difficulty levels and put them in the same room. | High | Done |
| FR2.3 | If there is a valid match, the system should match the users within 30s. | High | Done |
| FR2.4 | The system should inform the users that no match is available if a match cannot be found within 30 seconds. | High | Done |
| FR2.5 | The system should provide a means for the user to leave a room once matched. | Medium | Done |
| FR2.6 | The system should allow the user to proceed with doing the question even without a valid match, should they opt to. | Low | Not Done |

| | | | |
|-------|---|-----|----------|
| FR2.7 | The system should allow the users to continue waiting if no match can be found after 30s. | Low | Not Done |
| FR2.8 | The system should allow users to select the topic of the questions they wish to attempt | Low | Not Done |
| FR2.9 | The system should allow users to send an invite link to their peers whom they want to practice with | Low | Not Done |

2.1.3 Collaboration Service

| S/N | Functional Requirement | Priority | Status |
|-------|--|----------|----------|
| FR3.1 | The system should allow real-time collaboration (concurrent code editing) between participants in the room | High | Done |
| FR3.2 | The system should let the users know the username of the person they are collaborating with in the room | High | Done |
| FR3.3 | The system should let the users know when the person they are collaborating with leaves the room | High | Done |
| FR3.4 | The system should allow the users to compile and run their code. | Medium | Not Done |

2.1.4 Question Service

| S/N | Functional Requirement | Priority | Status |
|-------|--|----------|----------|
| FR4.1 | The system should provide a question of chosen difficulty to the pair of users. | High | Done |
| FR4.2 | The system should be able to display a history of questions attempted by the user. | Medium | Done |
| FR4.3 | The system should show a question bank by difficulty level | Low | Not Done |
| FR4.4 | The system should be able to allow user contributed questions | Low | Not Done |

2.1.5 Webcam Service

| S/N | Functional Requirement | Priority | Status |
|-------|--|----------|--------|
| FR5.1 | The system should allow users to open a video call with the matched person within the collaboration page to simulate an interview environment. | High | Done |
| FR5.2 | The system should provide a mute and unmute functionality. | High | Done |
| FR5.3 | The system should provide a 'hide webcam' and 'show webcam' functionality. | High | Done |
| FR5.4 | The system should allow users to change their audio and video stream (different microphone, different webcam device) without opening a | Medium | Done |

| | | | |
|--|-----------------|--|--|
| | new video call. | | |
|--|-----------------|--|--|

2.2 Non-Functional Requirement

The main 3 Non-Functional Requirements that we have decided to focus on are Security, Usability and Performance, in that order.

Firstly, the most important NFR that we have focused on is Security. We believe that the users need to feel confident that their data and information are well protected and secure before they would be willing to use the product.

Secondly, we focused on the usability of the product. We believe that having a product that is intuitive and usable to the users is important, as this will retain the users.

Finally, we focused on the performance of the product. Performance is also important, as the users may be deterred if all the operations are sluggish. However, we believe that it is the least important of the 3 NFRs we have picked as there could be some tolerance for lower performance in this use case.

2.2.1 Security Requirements

| S/N | Non-Functional Requirement | Priority | Status |
|--------|--|----------|--------|
| NFR1.1 | Users' passwords should be hashed and salted before storing in the DB. | High | Done |
| NFR1.2 | The system should allow users to remain logged in and pages(eg. homepage) inaccessible from public users | High | Done |
| NFR1.3 | The tokens are blacklisted when users change password or logout, ensuring that users cannot log in with the same token | High | Done |
| NFR1.4 | The system should allow only authenticated users with valid username and password to log in | High | Done |
| NFR1.5 | The system should ensure that no two users have the same username | High | Done |

2.2.2 Usability Requirements

| S/N | Non-Functional Requirement | Priority | Status |
|--------|---|----------|--------|
| NFR2.1 | The system should ensure the selection of question difficulty is interactive and easy to understand the different difficulty levels | High | Done |
| NFR2.2 | The front-end UI should be user-friendly and easy to use, intuitive | High | Done |

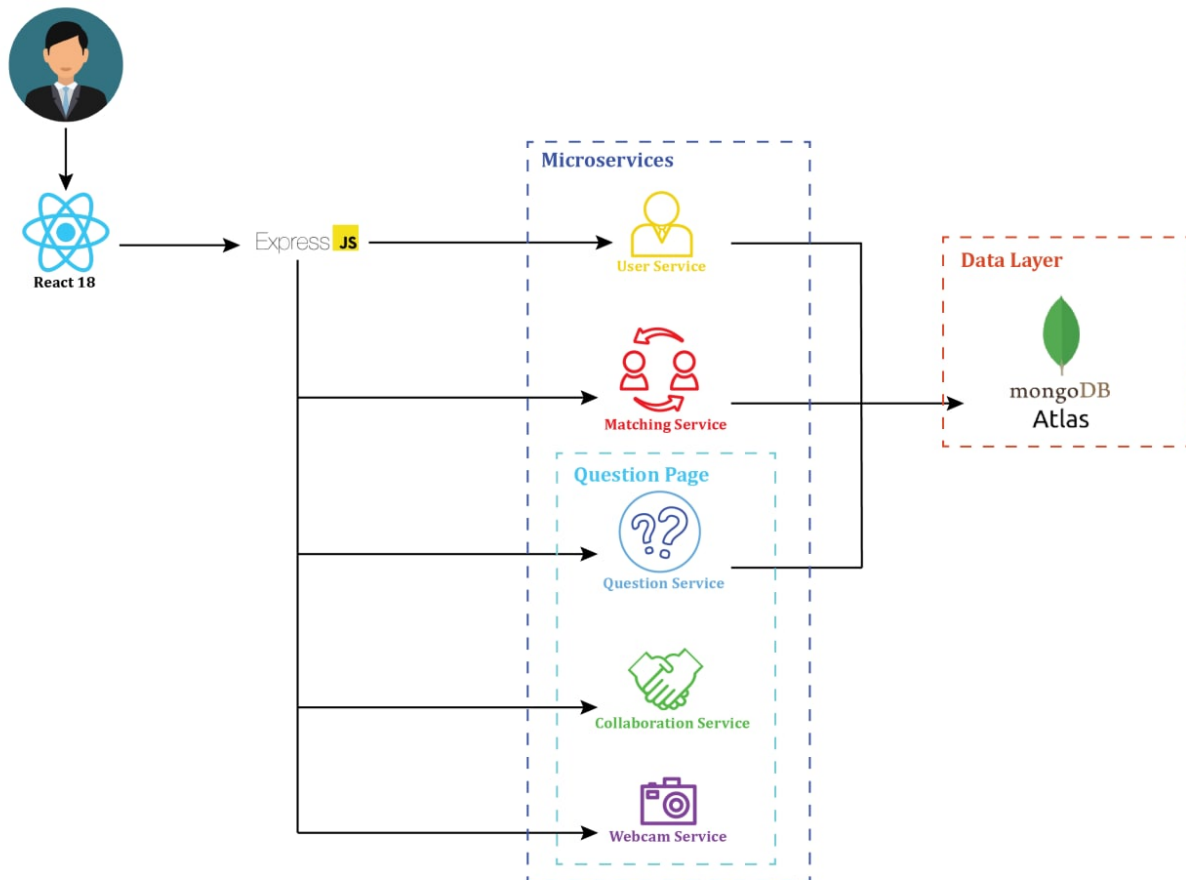
| | | | |
|--------|---|--------|------|
| | for new users to quickly familiarize with the platform | | |
| NFR2.3 | The system should have a navbar for easy navigation to different pages | High | Done |
| NFR2.4 | The system should be robust and able to handle errors gracefully, display loading status and error messages clearly | High | Done |
| NFR2.5 | The system should allow the user to login easily or redirect him/her to create a new account for a new user | Medium | Done |
| NFR2.6 | The front-end UI should be aesthetically pleasing, without too much clutter | Medium | Done |
| NFR2.7 | The users' code can be formatted based on the selected language. | High | Done |

2.2.3 Performance Requirements

| S/N | Functional Requirement | Priority | Status |
|--------|--|----------|--------|
| NFR3.1 | Matching the users should take less than 2s when there is a valid match | High | Done |
| NFR3.2 | Loading the different pages should take less than 1s. | High | Done |
| NFR3.3 | The collaborative editor should appear almost real time without conflicts | High | Done |
| NFR3.4 | Retrieving the history of questions attempted should take less than 1s on average even in cases where there's large number of attempts | High | Done |
| NFR3.5 | The video and chat function should not have significant delay, should be close to real-time. | High | Done |

3. High-level Architecture

3.1 High-level diagram

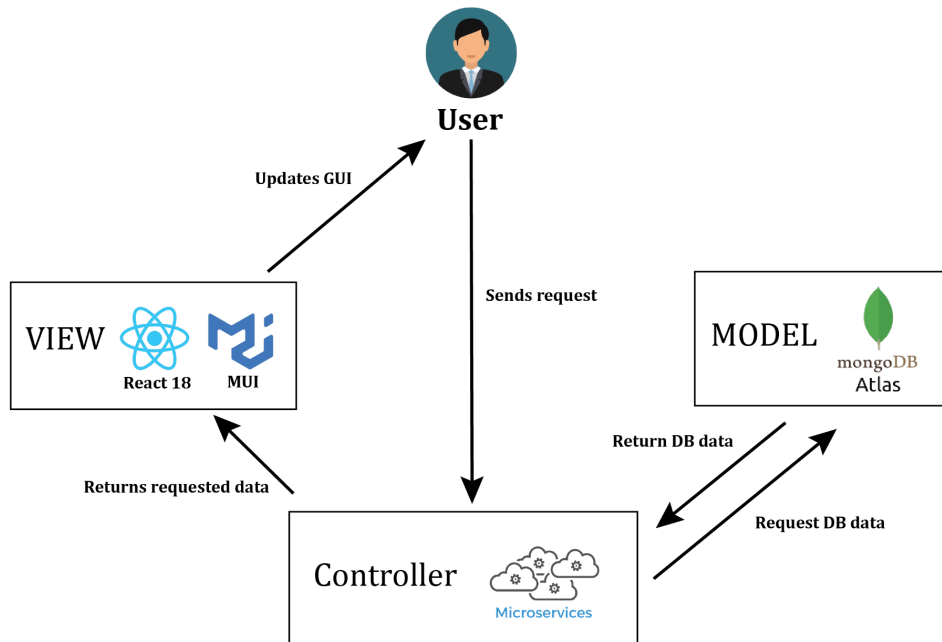


In our high-level architecture diagram, the user interacts with the frontend which uses React 18. The front end communicates with the backend microservices via ExpressJS. We have also opted to adopt the **Microservice Architecture** rather than the Monolithic Architecture. While a Monolithic Architecture could be easier to develop initially and also easier to deploy, a Microservice Architecture allows us to develop more easily in parallel, extend the functionalities of the services more easily and more scalable. Considering the short sprint cycles and potential complexity of the product with many different functionalities, the microservice architecture is highly appropriate for our use case.

There are a total of 5 microservices as mentioned above. Only 3 microservices, User Service, Matching Service, and Question Service, utilize a MongoDB Atlas database.

Also, 3 microservices, the Question Service, Collaboration Service, and Webcam Service, are only run within the Question page after the user matches with another user.

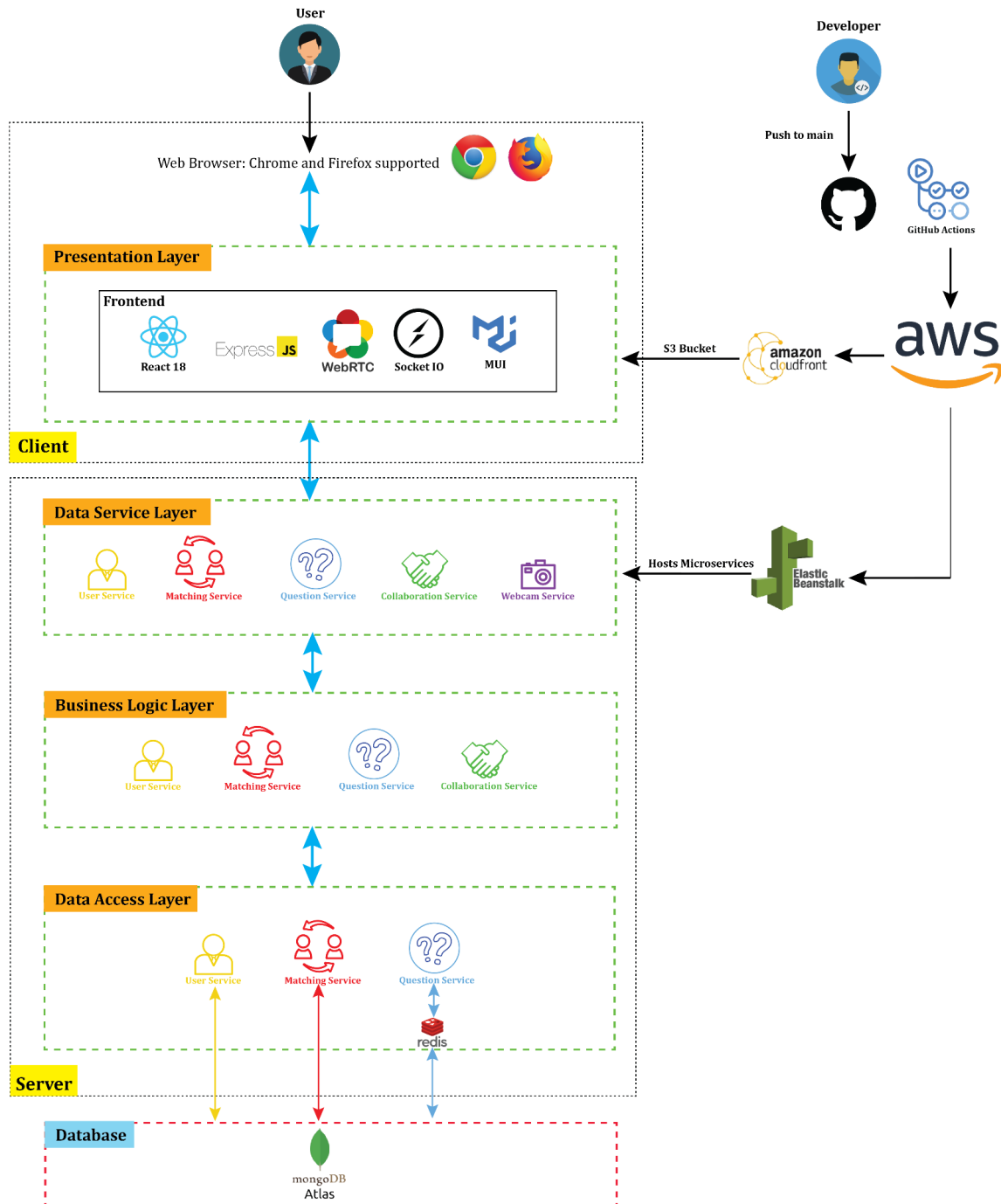
3.2 Model-View-Controller



Our application adheres to the Model-View-Controller architecture.

The user sends a request to the controller(microservices), which may then request the model(mongoDB) for required data. The model returns the requested data, after which the controller sends and updates the view which updates the GUI to inform the user.

3.3 4-tier architecture diagram



In the 4-tier architecture diagram, there are a total of 4 different layers. The first layer is the presentation layer. It is run by the client, and supports usage with Chrome or Firefox.

The second layer is the data service layer. All of the 5 microservices are part of the data service layer as they receive and transfer data to the presentation layer.

The third layer is the business logic layer. All microservices except the Webcam Service are part of the business logic layer as they manipulate data and handle business logic.

The fourth and final layer is the data access layer. The User Service, Matching Service and Question Service are part of this layer as they store and retrieve data from the mongoDB Atlas cloud database. The Question service utilizes Redis for caching to minimize unnecessary access to the cloud database.

Currently, our app is able to deploy the frontend using Amazon Cloudfront, and deploy the 5 microservices separately using Elastic Beanstalk. When a developer pushes to main, Github Actions will redeploy the services via AWS.

In the future, the microservices can be further broken down to conform with the 4-tier architecture diagram, with components of each microservice being hosted on different servers.

4. Design of the Key Components

In this section, we will first discuss some of the general design patterns that we have adopted in the design of our system. Subsequently, we will go into greater detail about the individual subcomponents and the implementation.

Facade Pattern

Facade is a structural design pattern that provides a simplified interface to a library, a framework, or any other complex set of classes¹. In our use case, the set of APIs that we have defined in our respective microservices provides a simplified interface to the data layer for the client (frontend) to call and retrieve the necessary information. This helps to isolate the business logic, thus reducing the complexity of the system and increasing maintainability.

Repository Pattern

Repositories are classes or components that encapsulate the logic required to access data sources.² In our microservices, we have incorporated the repository pattern in order to separate the business logic from the data layer, and we have used the Object-Relational Mapping (ORM) to achieve this.

Below, there are some snippets of code to illustrate how the Repository pattern is used in the Question Service. First, a Question Model is declared. The second snippet of code from question-orm.js shows some of the logic within the code, calling functions from repository.js. The third snippet of code from repository.js shows the implementation of some of the functions, and calls the database.

This pattern is helpful as we can now separate the main logic from the interactions with the database. In the event we decide not to use MongoDB and consider other options such as MySQL, most of the changes only need to be made in repository.js instead of all the related files.

¹ <https://refactoring.guru/design-patterns/facade>

² <https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design>

```
import mongoose from 'mongoose';
var Schema = mongoose.Schema
let QuestionModelSchema = new Schema({
  title: {
    type: String,
    required: true,
    unique: true,
  },
  question: {
    type: String,
    required: true,
  },
  difficulty: {
    type: String,
    required: true,
  },
  attempts: {
    type: Array,
    default: []
  }
})

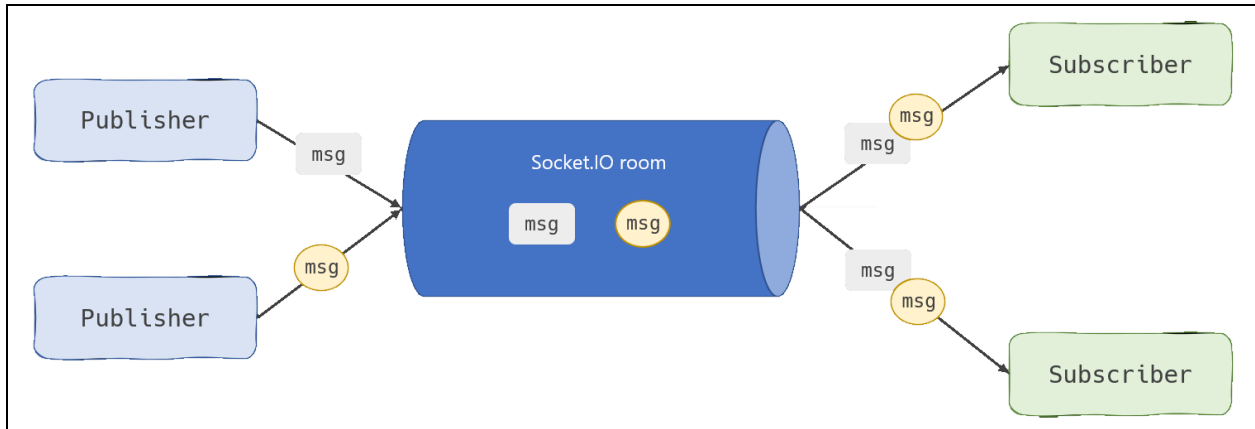
export default mongoose.model('QuestionModel', QuestionModelSchema)
```

```
export async function ormCreateQuestion(title, question, difficulty) {
  try {
    const isInDb = await checkQuestionInDatabase(title)
    if (isInDb) {
      throw new Error("Duplicate Question")
    }
    const newQuestion = await createQuestion({title, question, difficulty});
    newQuestion.save();
    return true;
  } catch (err) {
    console.log('ERROR: Could not create new question');
    return { err };
  }
}
```

```
export async function checkQuestionInDatabase(inputTitle) {
  return db.collection("questionmodels")
    .countDocuments( { title: inputTitle } )
    .then(num => {
      if (num > 0) {
        return true;
      }
      return false;
    });
}
```

Publish Subscribe Pattern

In a publish subscribe messaging, publishers post messages to an intermediate message broker, and the broker can broadcast the message to subscribers who are subscribed to the broker. In our use case, Socket.IO is used to implement pub-sub messaging in some of our services. These include the Matching Service, Collaboration Service and Webcam Service. Socket.IO has pub-sub channels also known as “rooms”, and messages can be filtered to broadcast to subscribers of a channel. Below is an illustration of how Socket.IO works.



4.1 User Service

4.1.1 Design of User Service

The User Service is responsible for maintaining information about users of PeerPrep, providing CRUD operations such that the users can create a new account, login, logout, change password and delete account. Authentication is also implemented to ensure that only authenticated users can login.

Functionality #1: Adding, deleting, modifying accounts

The user is able to create an account with an associated username and password. The username is unique, and is used for the matching service. The user sends the username and password to the User Service, which adds them into the database. To prevent loss of private data should the database information be leaked, the password is hashed using the bcrypt library by the User Service. This makes it such that compromise of database data would not reveal the plain-text passwords of each user, increasing the security of the application. The passwords are also “salted” before hashing, so that users with the same password will not have the same hash in the database. Attackers would not be able to identify users with the same password, further increasing security.. User Service also allows users to change their passwords and delete their account, but the username is not changeable.

Functionality #2: Authentication for protected routes

Authentication is required whenever the user accesses routes which contain confidential or private information. These routes would then be protected by a PrivateRoute component as shown below.

```

{ /* <Route exact path="/" element={<Navigate replace to="/login" />></Route> */}
<Route exact path="/" element={
  <PrivateRoute>
    <HomePage />
  </PrivateRoute> } />

  <Route path="/signup" element={<SignupPage /> } />
  <Route path="/login" element={<LoginPage /> } />

  <Route path="/settings" element={
    <PrivateRoute>
      <SettingsPage />
    </PrivateRoute> } />
  <Route path="/changepw" element={
    <PrivateRoute>
      <ChangepwPage />
    </PrivateRoute> } />
  <Route path="/home" element={
    <PrivateRoute>
      <HomePage />
    </PrivateRoute> } />
  <Route path="/loading" element={
    <PrivateRoute>
      <HomePage />
    </PrivateRoute> } />
  <Route path="/question" element={
    <PrivateRoute>
      <HomePage />
    </PrivateRoute> } />
  <Route path="/history" element = {
    <PrivateRoute>
      <HistoryPage />
    </PrivateRoute>
  } />
  <Route path="*" element={<div>
    <h2>404 Page not found</h2>
  </div> } />

```

Code block showing PrivateRoute protection

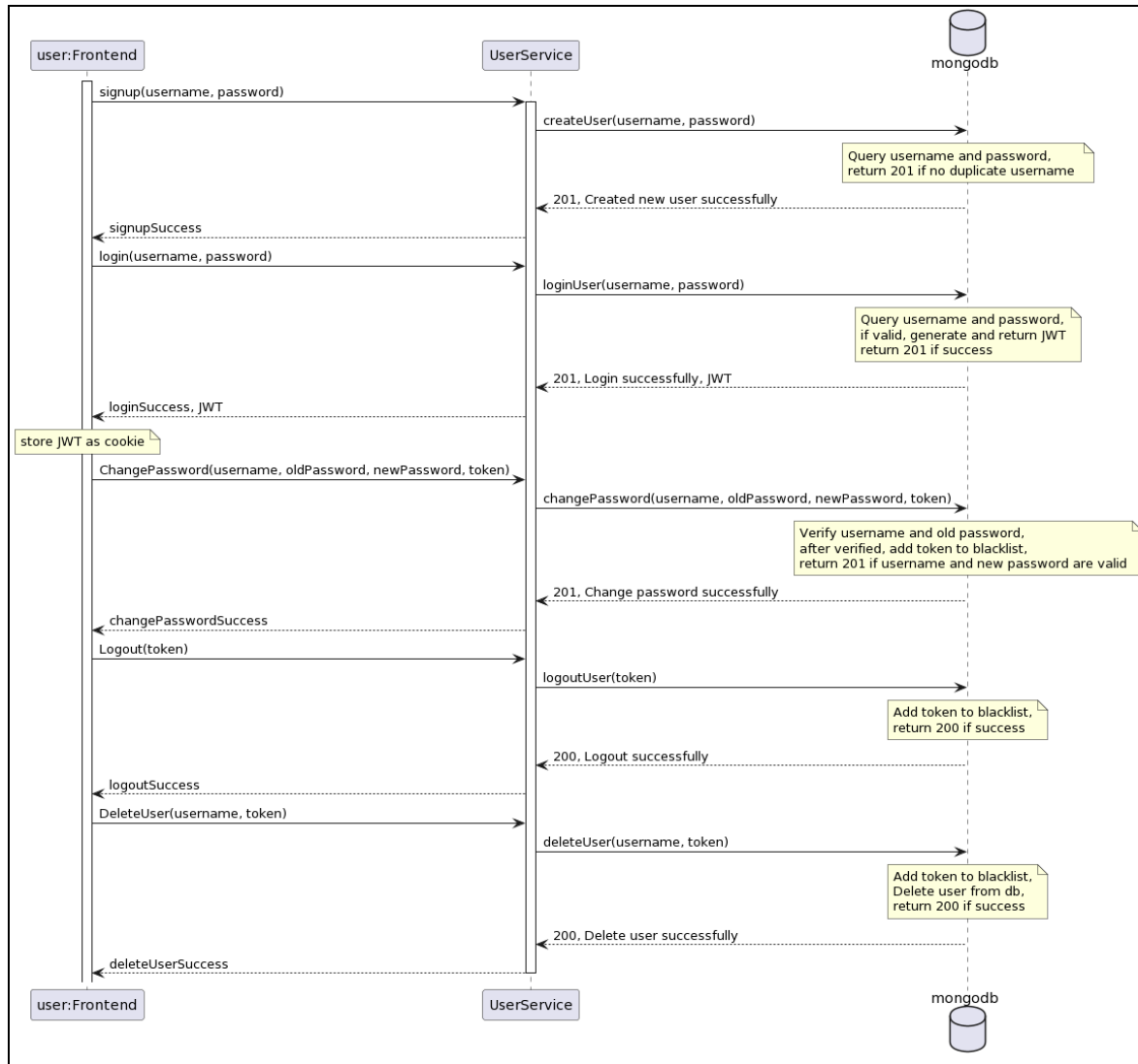
When accessed, these routes will require authentication before loading their respective content. Should authentication fail, the user is redirected to the login page.

Authentication is done using JWT tokens from the “jsonwebtoken” library. After the user submits a correct password and successfully logs in, the User Service then signs and returns a JWT token to the user, which is stored as a cookie. Access to PrivateRoutes will check for a JWT token from the client, who sends the JWT token to be authenticated by the User Service. Only after the User Service returns an authentication message can the user view the protected route.

Upon user logout, the authentication token is added into a blacklist database on mongoDB, so that future authentications with the same token will fail. The cookie is also cleared for the user.

Cookies are added to the client with the “httpOnly” flag set to true. This makes it such that the cookie is not accessible via Javascript at the frontend browser. This protects against Cross-Site-Scripting (XSS), as users are not able to view, modify, and submit fabricated cookies for authentication.

The overall flow of the User Service can be seen in the sequence diagram below.



Sequence Diagram of User Service

4.1.2 Rationale

For the first functionality, we chose to make usernames immutable. This is to reduce complexity in the implementation of the app, since the username is used for storing question history and matching users. This will not likely inconvenience users since users are able to create multiple accounts with different usernames.

For the second functionality, we decided to make the authentication more secure albeit with higher latency by keeping the JWT token blacklist. In actuality, the JWT token functions like a session cookie, since it is added and deleted upon user login and logout. With the “httpOnly” flag set, users or attackers are not able to snoop for this token, hence the token can be assumed to be confidential. Attackers also cannot submit their own fabricated cookie. However, there might be other security issues, man-in-the-middle attacks which may reveal this token, hence we decided to keep the Blacklist check as

an added safety measure. The penalty to latency is also minimized by “fast-failing” of the check for the cookie.

```
export async function authenticate(req, res, next) {
  try {
    const token = req.cookies["token"];
    if (!token) throw new Error("No token provided") ← Fast Fail

    const user = verifyJWT(token);
    if (user.err) throw new Error("Invalid Token")
    const username = user.username
    const resp = await _checkValidToken(token)
    if (resp.err) throw new Error("Invalid Token")

    let userInDb = await _searchUser(username)
    if (userInDb.err) throw new Error("Invalid Token")
    const password = userInDb.password

    req.userInfo = { username, password }
    console.log(`Authentication successful ${username}`)

    res.cookie("user", username, { maxAge: ONE_HR_IN_MS })

    next()
  } catch (err) {
    console.error(err)
    res.cookie("token", "", { sameSite: "None", httpOnly: true, secure: true, maxAge: 0 })
    res.cookie("user", "", { maxAge: 0 })
    return res.status(401).json({ message: 'Authentication failed' })
  }
}
```

Code block showing fast-fail of cookie check before Blacklist check (_checkValidToken)

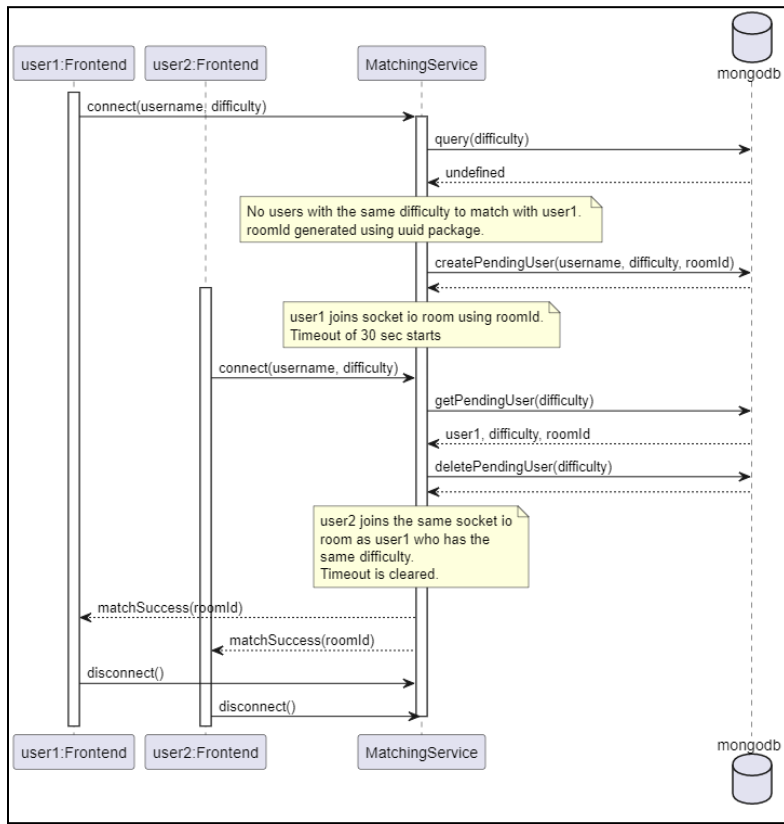
4.2 Matching Service

4.2.1 Design of Matching Service

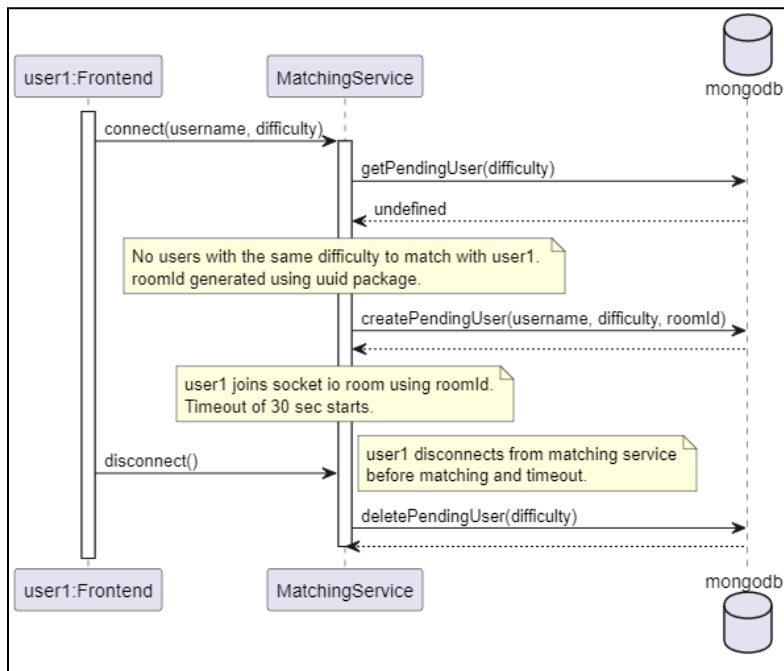
The Matching Service matches users who choose the same question difficulty in PeerPrep. Immediately after choosing the question level difficulty, clients will establish connection to the Matching Service via Socket.IO connections, while providing their username and chosen difficulty level. The Matching Service then determines if the connecting client is able to match with another user. After two users are matched, they will be given a unique id that represents their matching.

Functionality #1: Matching two users who choose the same difficulty level

There are two possible cases when a user connects to the Matching Service. The first case is that there is no other user who chose the same difficulty and is waiting to be matched. In this situation, a 30 seconds timeout is set for the user to wait for a match. During the 30 seconds duration, the user is able to choose to leave and disconnect from the server before getting matched. The second case is that there is another user who chose the same difficulty and is waiting to be matched. Both users are successfully matched, and are given the same unique id and disconnect from the server.



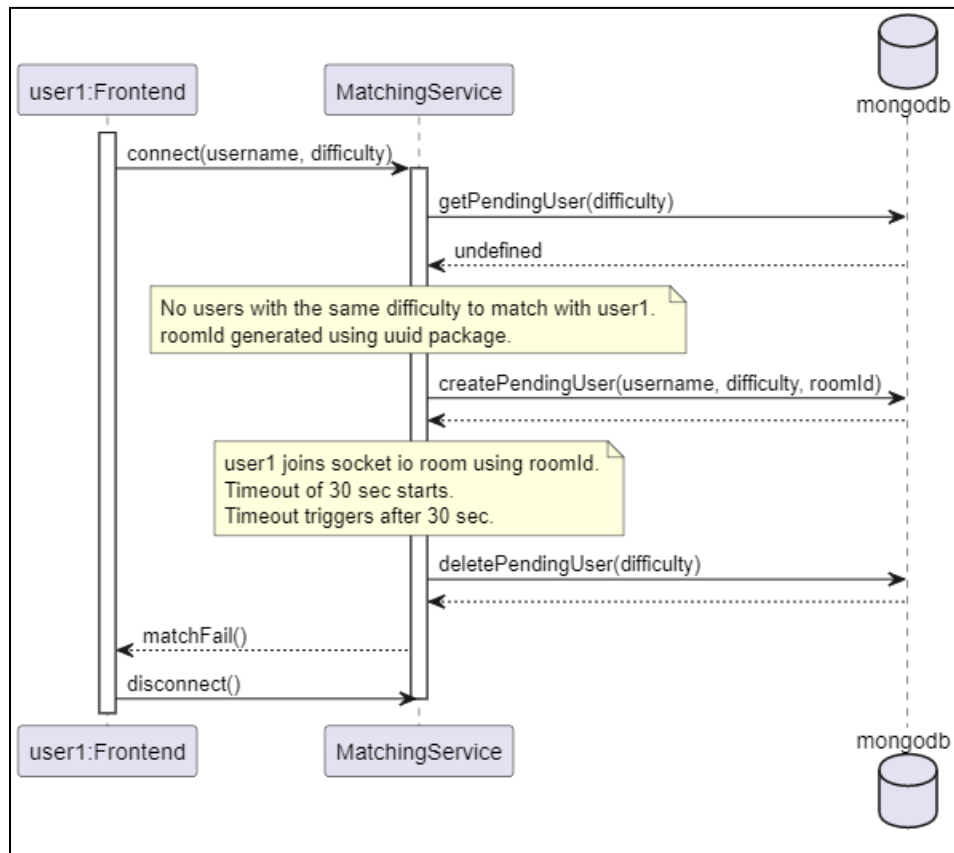
Successful matching of two users



Disconnecting before timer timeouts

Functionality #2: Sending timeout to users who fail to match for 30 seconds

When a user connects to the matching service, a 30 seconds timeout is set for the user to wait for a match. If 30 seconds have passed and the user is not matched, then the user will be sent a “matchFail” message and disconnects from the server.



4.2.2 Rationale

A Pub-Sub mechanism is required to inform users that a matching is made in real time, instead of a request-response mechanism where the client has to continuously send requests to the server to get the matching status. Therefore, Socket.IO is used.

Mutex is also used in order to control the order of asynchronous events such as when accessing the database. For example, when a user connects to the Matching Service and queries the database to find another user who has chosen the same difficulty, this query is an asynchronous event and the server waits for the database to send a response. At the same time, there may be another user who can perform database operations such as adding or deleting entities. Therefore, mutex is used to prevent any race condition.

4.3 Collaboration Service

4.3.1 Design of Collaboration Service

The Collaboration Service enables real-time code editing by multiple users. After being matched by the Matching Service, users connect to the Collaboration Service via Socket.IO connection, and provide their username and unique room id that is generated by the Matching Service. The Collaboration Service is able to match users with the same room id by placing them in the same Socket.IO room. All users in the same room can edit the same document.

Functionality #1: Real-time editing of document

When a user makes a change to a code, the change, which contains information about the position of the change and the characters added or deleted, will be sent to the server. The Collaboration Service assigns a definite order to the changes, using a technique known as operational transformation which allows collaborative editing on a document by handling conflicts in changes made to the document. The server sends the change to all users editing the document, synchronizing the code that matched users see

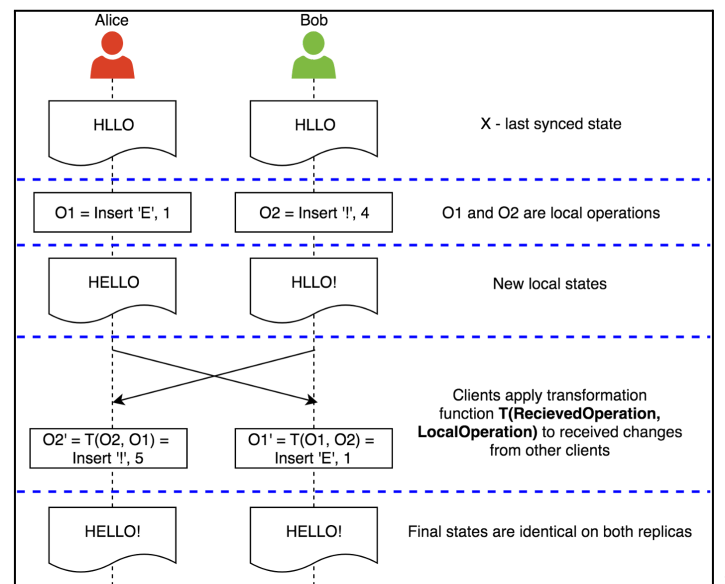


Diagram illustrating Operation Transformation

Functionality #2: Notifying users when a user in the same room leaves

The Collaboration Service notifies users when a user they have been matched to clicks the “Finish” button. Clicking on the button disconnects the Socket.IO connection with the server, and the server will broadcast to the users in the same Socket.IO room that their collaborator has clicked on the “Finish” button.

³<https://medium.com/coinmonks/operational-transformations-as-an-algorithm-for-automatic-conflict-resolution-3bf8920ea447>

4.3.2 Rationale

For the first functionality, we initially implemented collaborative editing by sending the entire text to the server after a change has been made. The server then broadcasts the changed document to all users in the same room. This led to changes made to the text being overwritten by past changes. The reason for this problem is due to the latency between making a change in the document, sending the change to the server, then receiving the change from the server. Therefore, there is a need to use an algorithm such as operational transformation that addresses the latency issue.

For the second functionality, the users have to connect to the Collaboration Service for the entire session, therefore, instead of implementing another service to notify matched users when one of them clicks on the button, the Collaboration Service can implement this functionality.

4.4 Question Service

4.4.1 Design of Question Service

The Question Service is responsible for Questions in PeerPrep. A data model of a Question can be seen below. To clarify, the attempts store an array of the usernames of the users who have attempted the question before.

| Question |
|--------------------|
| title: string |
| question: string |
| difficulty: string |
| attempts: string[] |

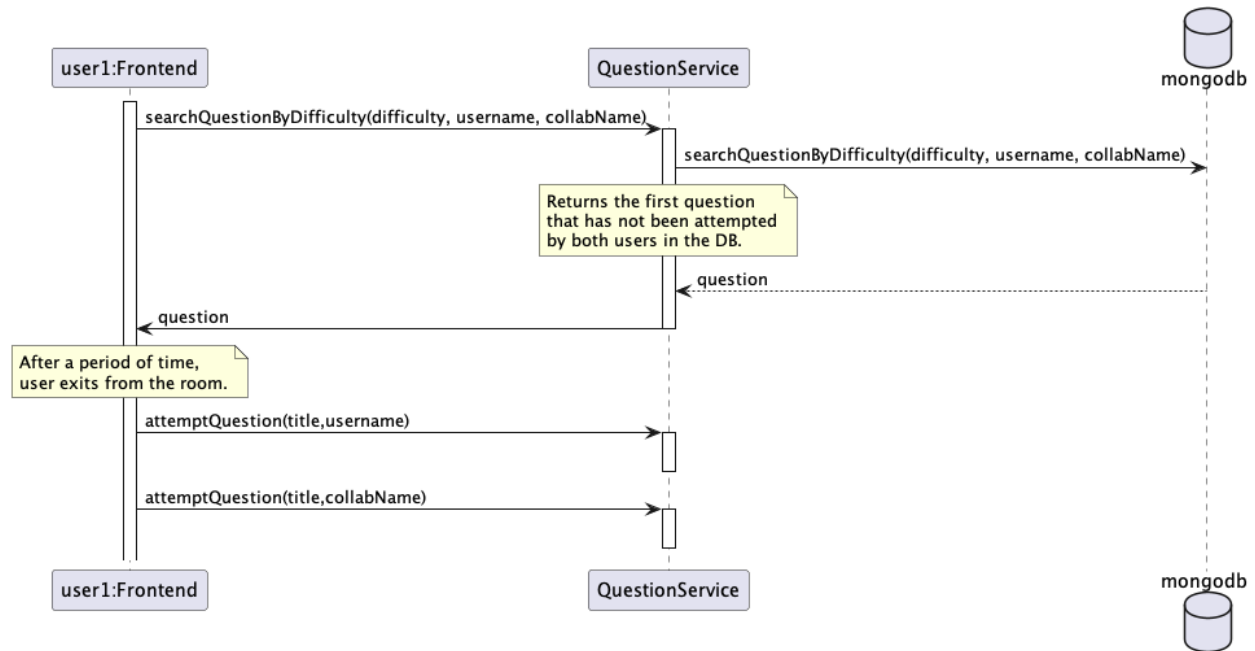
Currently, the Question Service supports the basic CRUD functions. Some of these functions are not utilized in the current iteration of the PeerPrep, but are created to support future possible extensions of PeerPrep.

| API | Purpose |
|--------------------------------|---|
| searchQuestionByDifficulty | Search for a single question by difficulty and 2 usernames. Currently used to retrieve a question that both users have not attempted before. |
| attemptQuestion | Adds the username to the array of users who have attempted the question. Currently used to keep track who attempted the question. |
| searchAllQuestionsAttempted | Search for all the questions attempted by a user. Currently used to retrieve all the questions attempted by a user. |
| createQuestion | Creating a new question. Could be used to support future extensions for user submitted questions. |
| searchAllQuestionsByDifficulty | Searching all questions by difficulty. Could be used to support future extensions for a question bank feature. |
| searchQuestionByTitle | Search for a question by title. Could be used to support future extensions where we want to display more information about a single question. |
| deleteQuestionByTitle | Deleting a question. Could be used to support future extensions for user submitted questions. |

In particular, we will be focusing on how searchQuestionByDifficulty, attemptQuestion, and searchAllQuestionsAttempted supports the functionality of the current application.

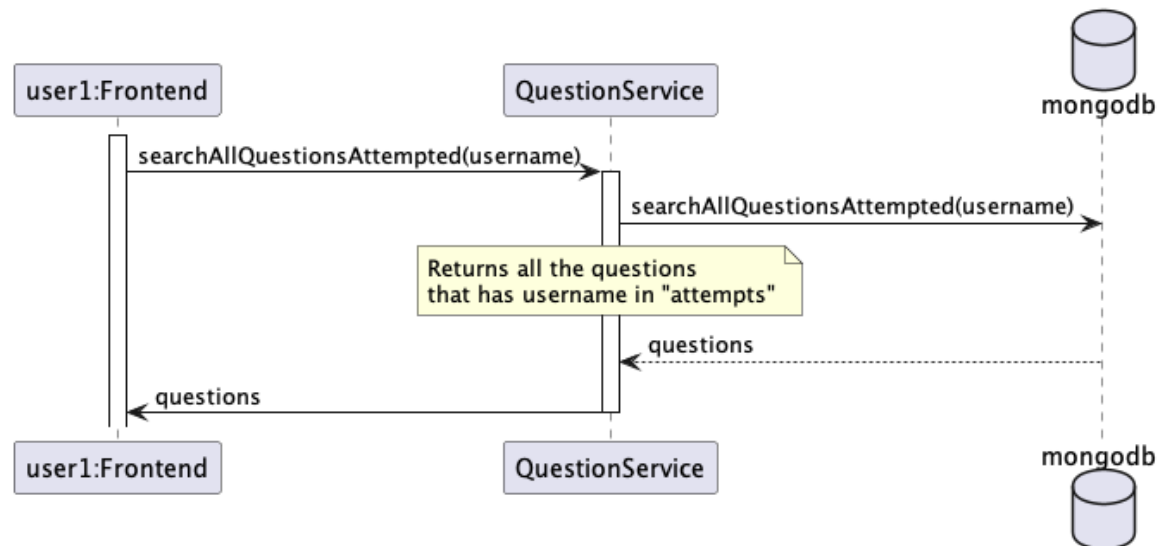
Functionality #1: Displaying a question that has not been attempted by both users

Firstly, PeerPrep needs to be able to display a question that has not been attempted by both users before. Both users will then search the question by difficulty, and this will return the same result for both users. When one of the users exits the room, the exit event will be triggered for that user. 2 api calls will then be made in order to mark the question as attempted by both users.



Functionality #2: Displaying a list of questions attempted by a user

Secondly, PeerPrep needs to be able to display a list of all the questions that a user has attempted before. We have also made use of Redis in order to optimize this operation.



4.4.2 Rationale

For the first functionality of displaying a question that has not been attempted before, one of the key decisions was to decide when to mark a question as attempted. We decided to only mark a question as attempted only when the user exits in order to prevent any race condition that may occur as a result.

We also designed the attemptQuestion api to add one user at a time in consideration of the event whereby there are no collaborators.

For the second functionality, we have also used Redis in order to optimize the query time. This is due to the fact that filtering based on an element in an array could be a costly operation, coupled with the fact that there could be users with many questions attempted. This would result in a long query time. We felt that using Redis to cache the query is suitable in this scenario.

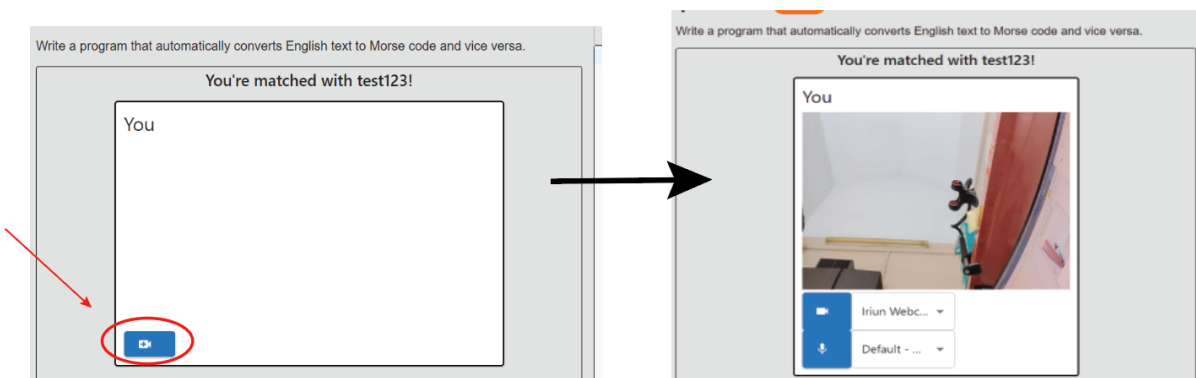
4.5 Webcam Service

4.5.1 Design of Webcam Service

The Webcam Service is responsible for establishing a WebRTC connection between the 2 users matched up in the Question Room. This service allows users to perform video and audio communication with each other, similar to that of Zoom. Besides allowing for better discussion of the coding question, the service also allows users to practice their interview skills with each other, such as tone and posture when answering technical questions.

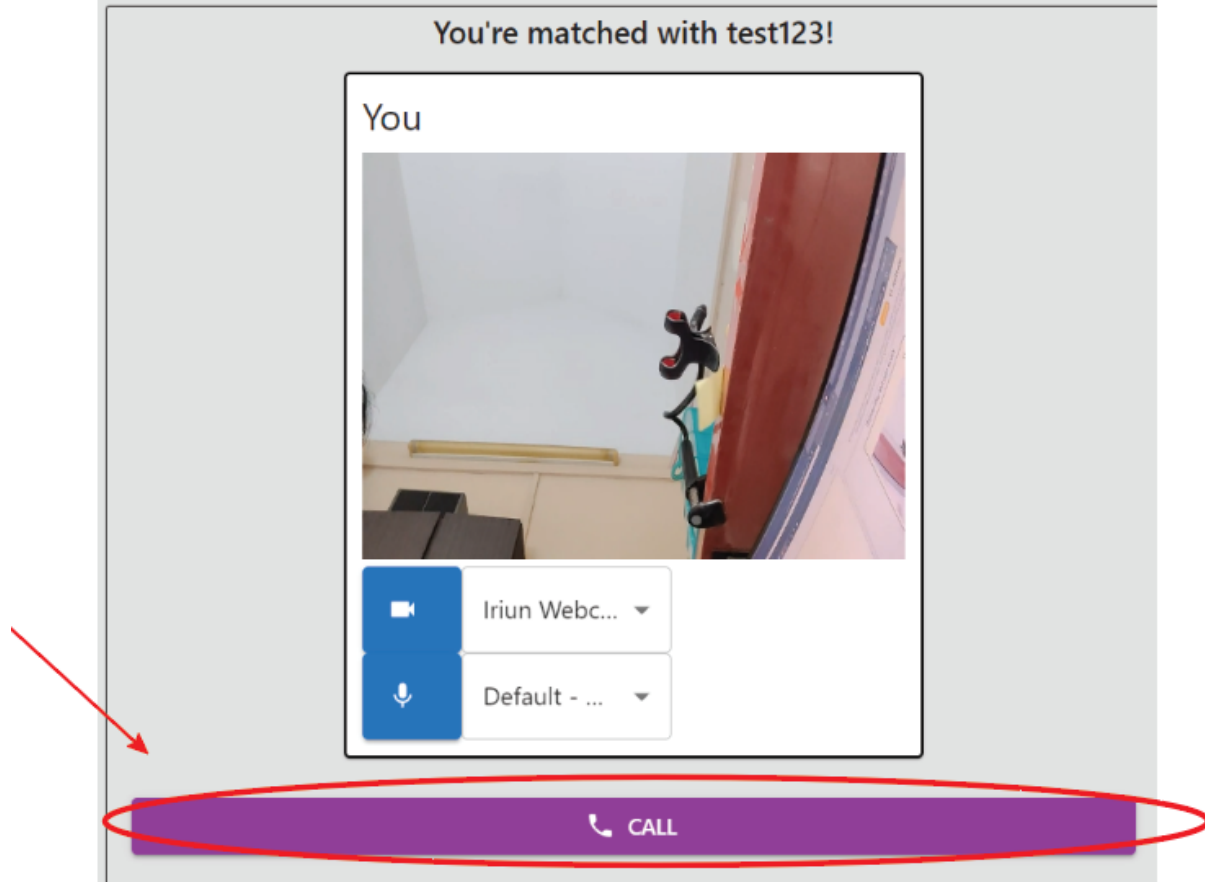
The service utilizes Socket.IO to communicate to both users. The Webcam Service emits messages using the same room id as the Collaboration Service.

The user first turns on their video feed, before starting a video call using the “Call” button.



Turning on video feed

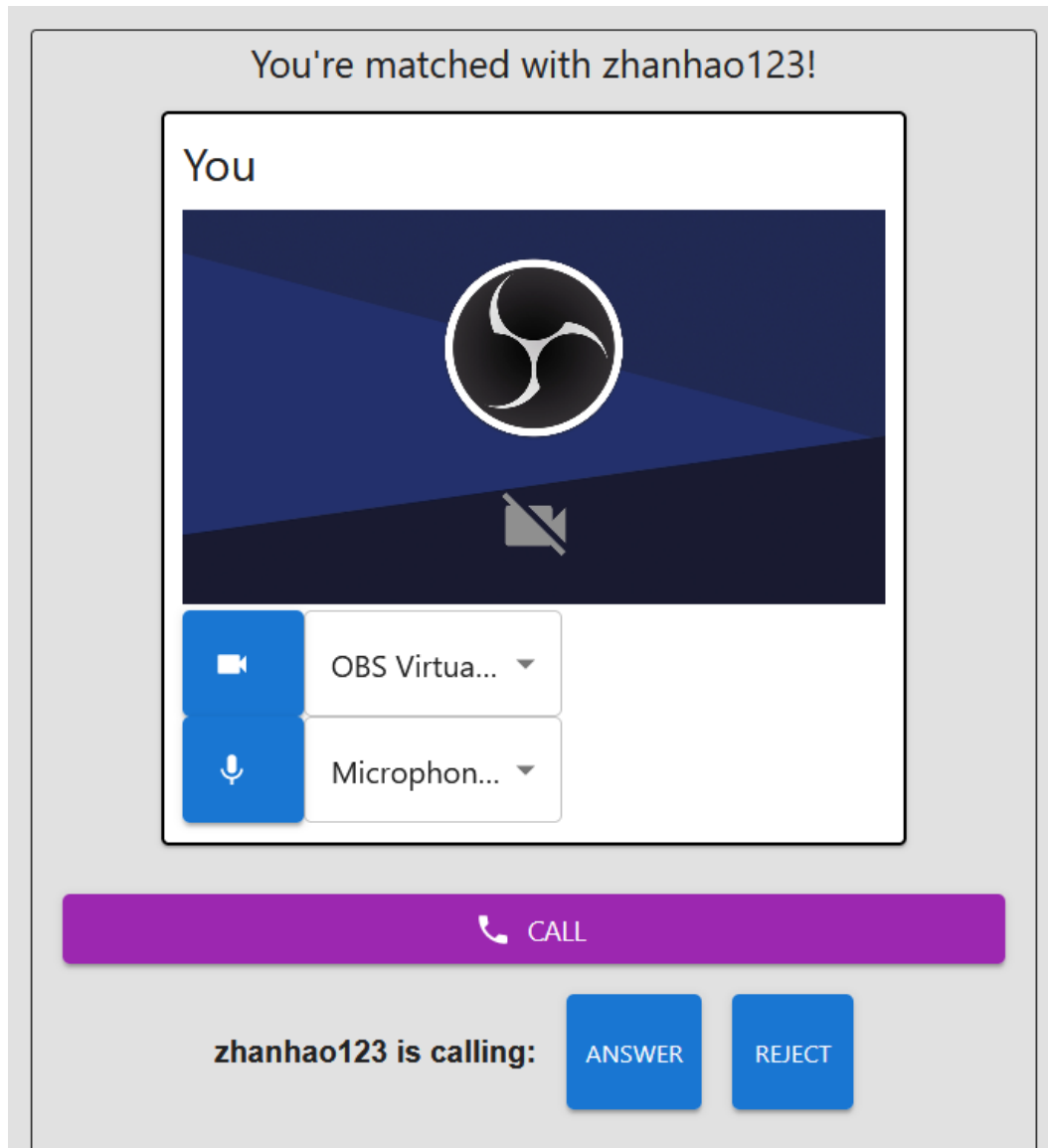
Implement a binary heap. Once using a list as the base data structure and once by implementing a pointer-linked binary tree. Use it for implementing heap-sort.



Starting Call

This emits a call signal to the Webcam Service along with the WebRTC connection signal data. The Webcam Service then emits this signal to the other user.

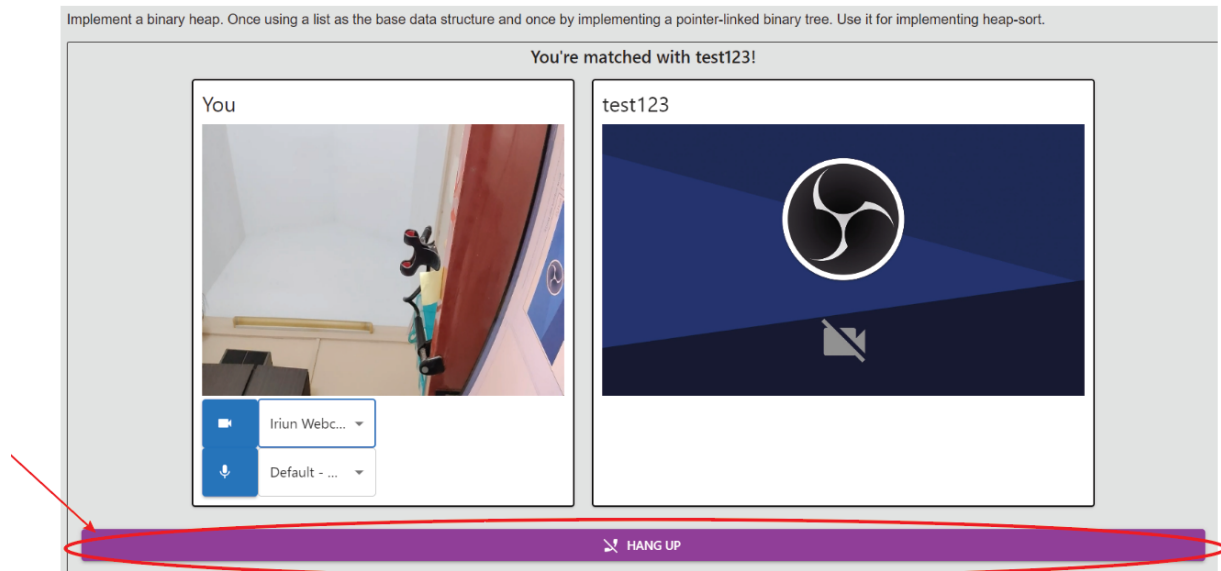
The other user in the room would then get this signal and have the option to accept or reject the call.



Receiving call

The option to reject provides privacy for users who might not wish to be in a video call.

After accepting, an answer call signal is emitted to the Webcam service along with their own WebRTC connection signal data, which emits a call accepted signal to the other user. The user who initiated the call then accepts the signal. WebRTC connection would then be finished and the users can then engage each other in the video call. After the connection is done, any user can choose to hang up which terminates the RTC connection. This is similar to the process above, with the Webcam Service emitting the end call signal to the other user.



Ending call

Functionality #1: Having real-time video and audio connection between users

The service should aim to provide a video call which has low latency to simulate live communication. In this regard, WebRTC was chosen as it performs well. It is also open-source and free, hence would not add extra costs to the application. A possible improvement in the future would be testing the connection for all types of internet speeds and having a variable video and audio quality matching the internet speed so that the service can work the best for all types of users.

Functionality #2: Switching of audio and video streams with muting and hiding functionality

The Webcam Service allows users to change their audio and video streams outside of and during the call, which is done by switching the track of the connection. This allows users who have multiple webcams or microphones to be able to seamlessly switch between them to find the best one for the video call. The service also allows users to mute and hide their audio and video streams. This provides extra privacy to users.

4.5.2 Rationale

For the call to be started or answered, users have to first turn on their video feed. This makes it slightly inconvenient, as an extra step for users to do before making the call. For users who do not wish to share their audio or video stream, they will also have to turn on their video feed before muting and hiding their video.

The reason behind this is to reduce the complexity of the service, since the Webcam service would then have to handle the adding in of new signals after the call is established. Given the possible bugs that could arise, it was decided to make it mandatory for the video feed to be on before the call.

When changing video streams, instead of renegotiating a new stream, the same stream is used, with the track being changed to the new desired video track. This, similar to the above, reduces complexity for the Webcam service, not having to handle renegotiations. The inconvenience over here is that the user will not be able to use a new device that was added after the call was established, for example plugging in a new microphone in the middle of the call. Although this would inconvenience some users, we feel that this is a drawback rarely encountered by most users hence we decided not to implement it.

4.6 UI/UX enhancement

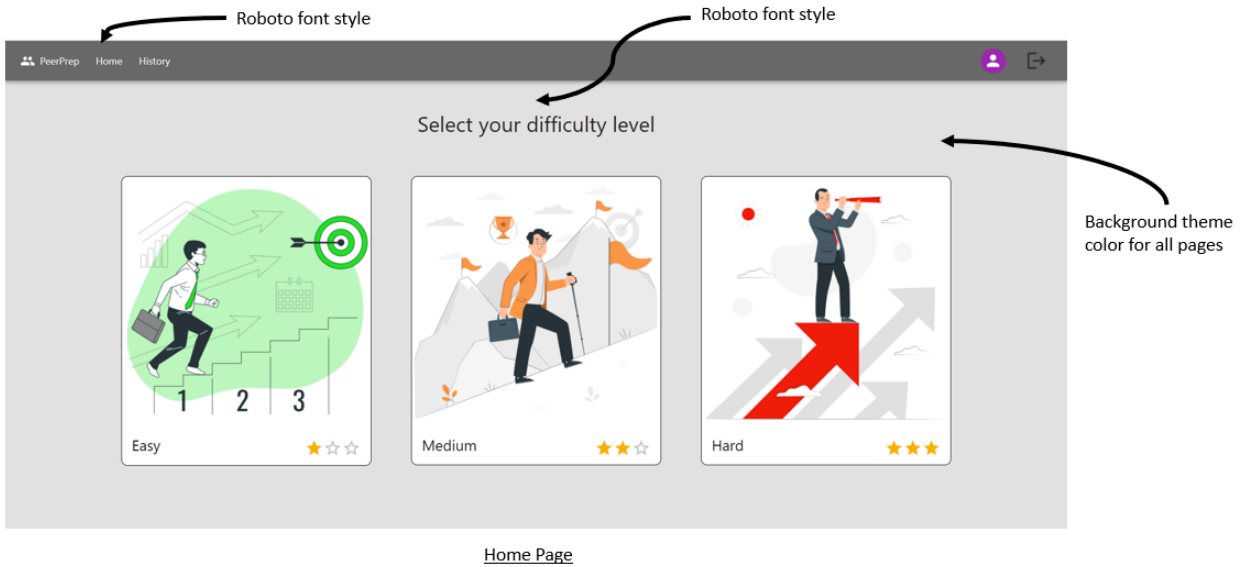
4.6.1 Design of UI/UX enhancement

User interaction and user experience (UI/UX) enhancements were implemented to make the application more intuitive and attractive to the users. Through designing the frontend, Nielsen's Usability Heuristics for User Interface Design was adopted, namely:

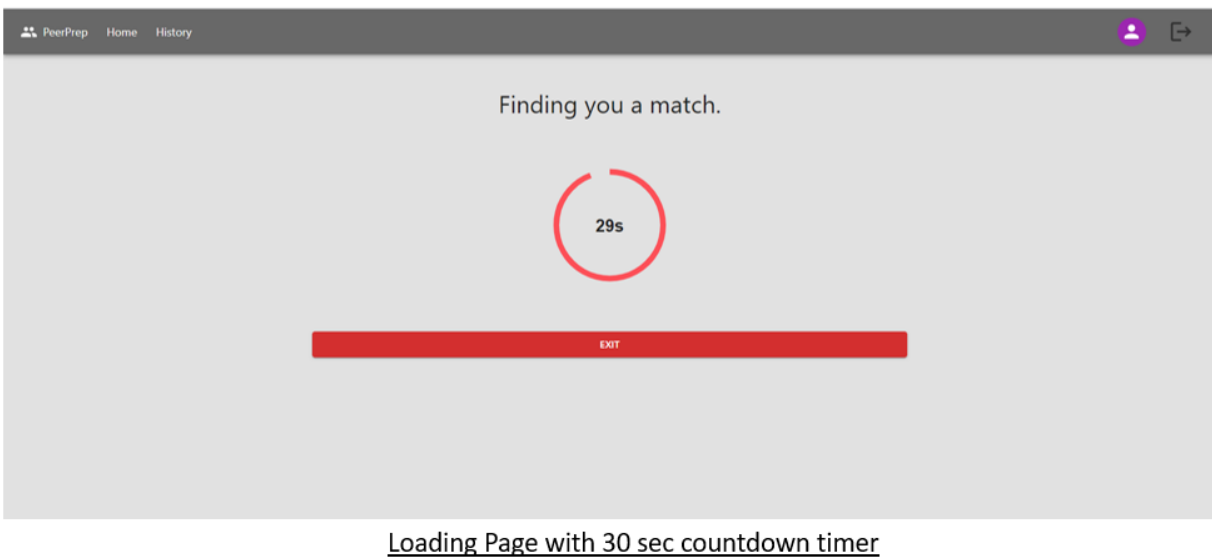
- H1 Visibility of System Status
- H4 Consistency and Standards
- H5 Error Prevention
- H7 Flexibility and Efficiency of Use
- H8 Aesthetics And Minimalist Design

The graphic elements and styling are consistent throughout the different pages as the user navigates in the app. The consistency and standards (H4) are also followed by choosing color palettes, themes, PeerPrep icon, layouts, alignments and Roboto, the long running #1 font⁴, for our display.

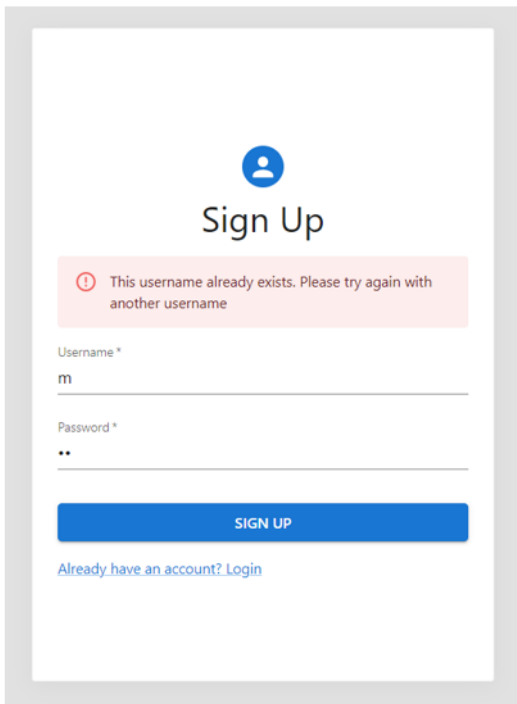
⁴<https://kinsta.com/blog/best-google-fonts/#:~:text=Roboto,appearances%20on%20Google%20Fonts'%20analytics.>



The system keeps the users informed about the current state and actions. Through the implementation of countdown timer, the users know that the system is looking for a match, aligning with Visibility of System Status (H1).

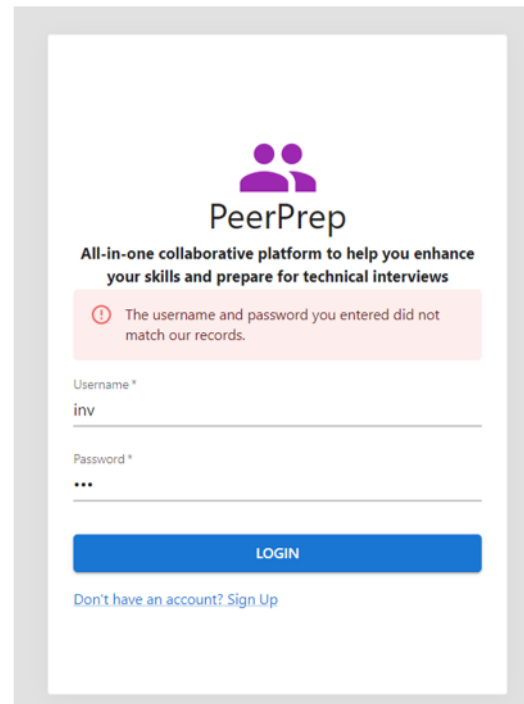


Error messages are stated clearly to the users when the users attempt to create an account with duplicate username or log in with an invalid username/password



The image shows a 'Sign Up' form with a blue user icon at the top. Below the icon is the text 'Sign Up'. A red error message box states: 'This username already exists. Please try again with another username'. The form has two input fields: 'Username *' with the value 'm' and 'Password *' with two dots. A blue 'SIGN UP' button is at the bottom, with a link 'Already have an account? Login' below it.

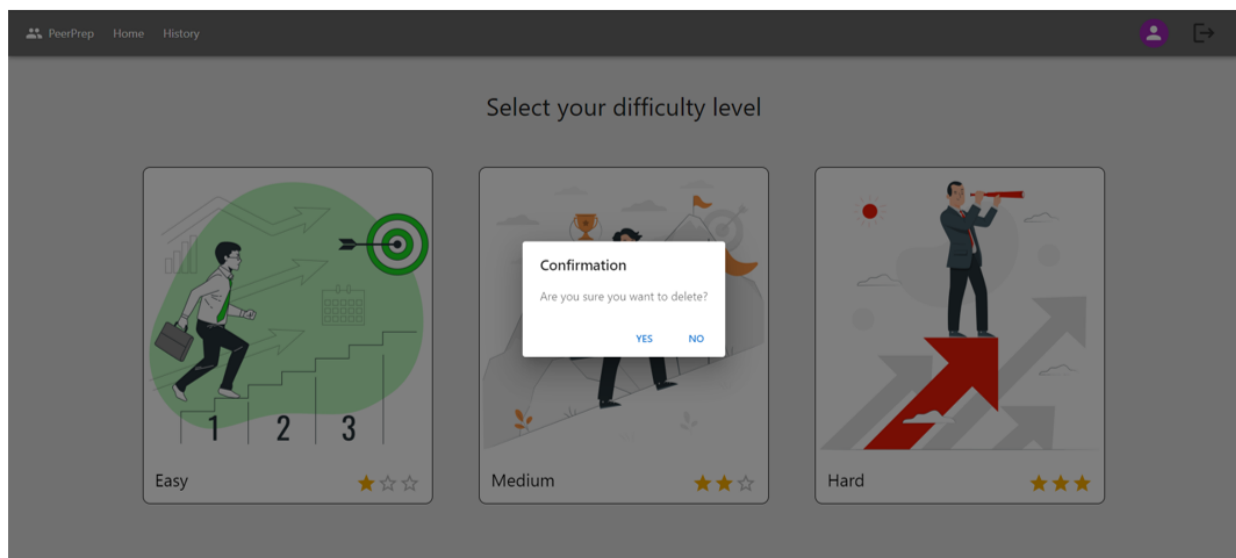
Error Messages upon duplicate username



The image shows a 'PeerPrep' login form with a purple user icon at the top. Below the icon is the text 'PeerPrep' and a tagline: 'All-in-one collaborative platform to help you enhance your skills and prepare for technical interviews'. A red error message box states: 'The username and password you entered did not match our records.'. The form has two input fields: 'Username *' with the value 'inv' and 'Password *' with three dots. A blue 'LOGIN' button is at the bottom, with a link 'Don't have an account? Sign Up' below it.

Error Messages upon invalid credentials

Confirmation messages are presented to the users in case the user accidentally makes a mistake. Deleting the user account is a critical action. Showing the user with a confirmation dialog can help to further ensure user's approval before they commit to the action, aligning with Error Prevention (H5).



The image shows a 'Select your difficulty level' screen with three options: 'Easy', 'Medium', and 'Hard'. Each option has an illustration and a star rating. A 'Confirmation' dialog box is overlaid on the 'Medium' option, asking 'Are you sure you want to delete?' with 'YES' and 'NO' buttons. The dialog box is white with a black border and a black title bar.

Delete User Confirmation

4.6.2 Design Decision and Implementation

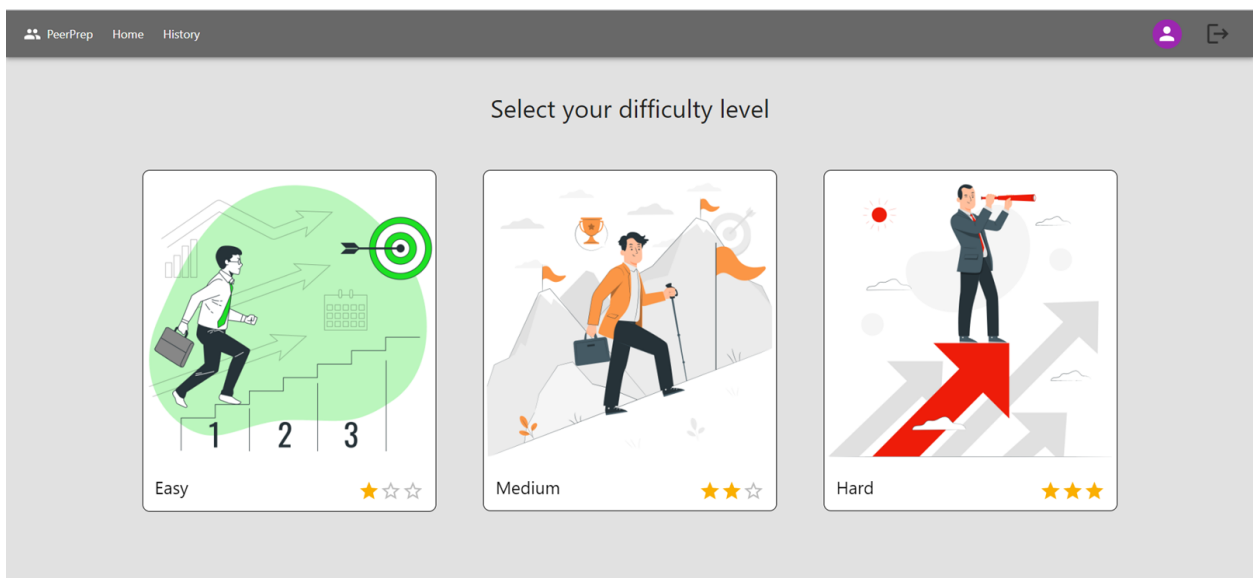
4.6.2.1 Design of Difficulty Selection

Initially, there are 3 buttons, EASY, MEDIUM and HARD. Clicking on the buttons will bring the user to the loading page.

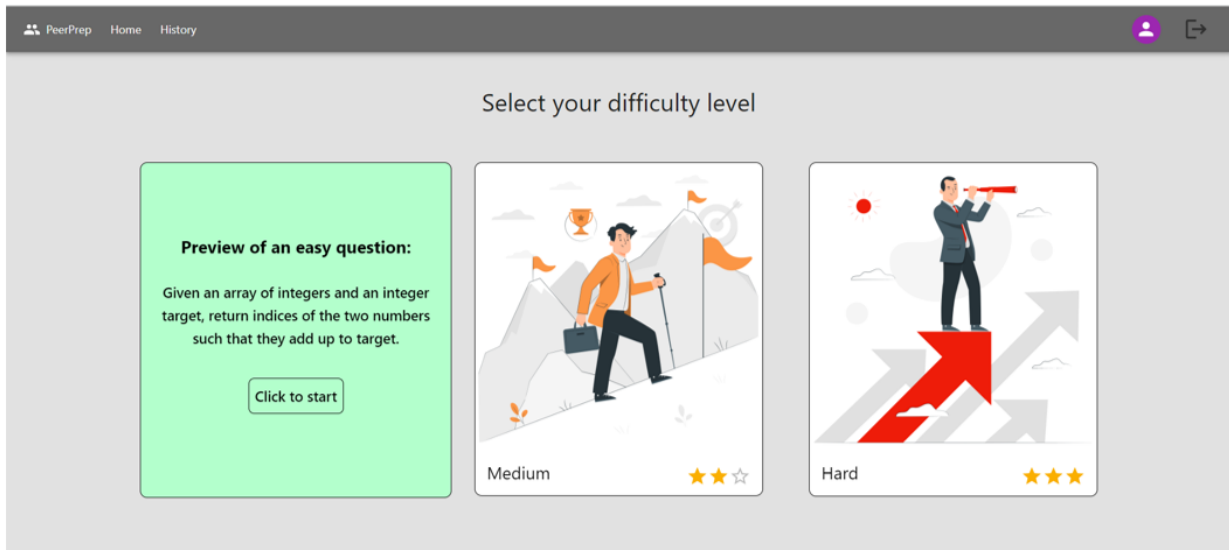


Old Design

The design is revised and optimized, changing from buttons to flippable cards with images. On the front of the card, images and stars are displayed to depict each difficulty level. This improves the aesthetics, making it more beautiful and attractive. Hovering over each difficulty level, the card will flip around and a sample question for that specific difficulty level is displayed. This allows the users to immediately gauge the difficulty and select the difficulty of his/her choice, improving user-friendliness.



New Design



Hovering over Easy difficulty

4.6.2.2 Rationale

Based on the initial design, the user will have to go through the process of waiting for a successful match before finding out if the question is too easy or just right or too hard.

Although the difficulty is shown on the buttons, a sample question will be clearer as different users have different perspectives of difficulty.

Allowing the users to view a sample question first can help to improve the flexibility and efficiency of use(H7), improving the usability of the system.

4.6.2.3 Feature Implementation Design Consideration

A particular design consideration was made during the implementation of this card flipping animation. Initially, react-flippy was used as the library to implement this animation and it worked fine. However, testing upon the newer npm versions, there was an error upon running: npm install.

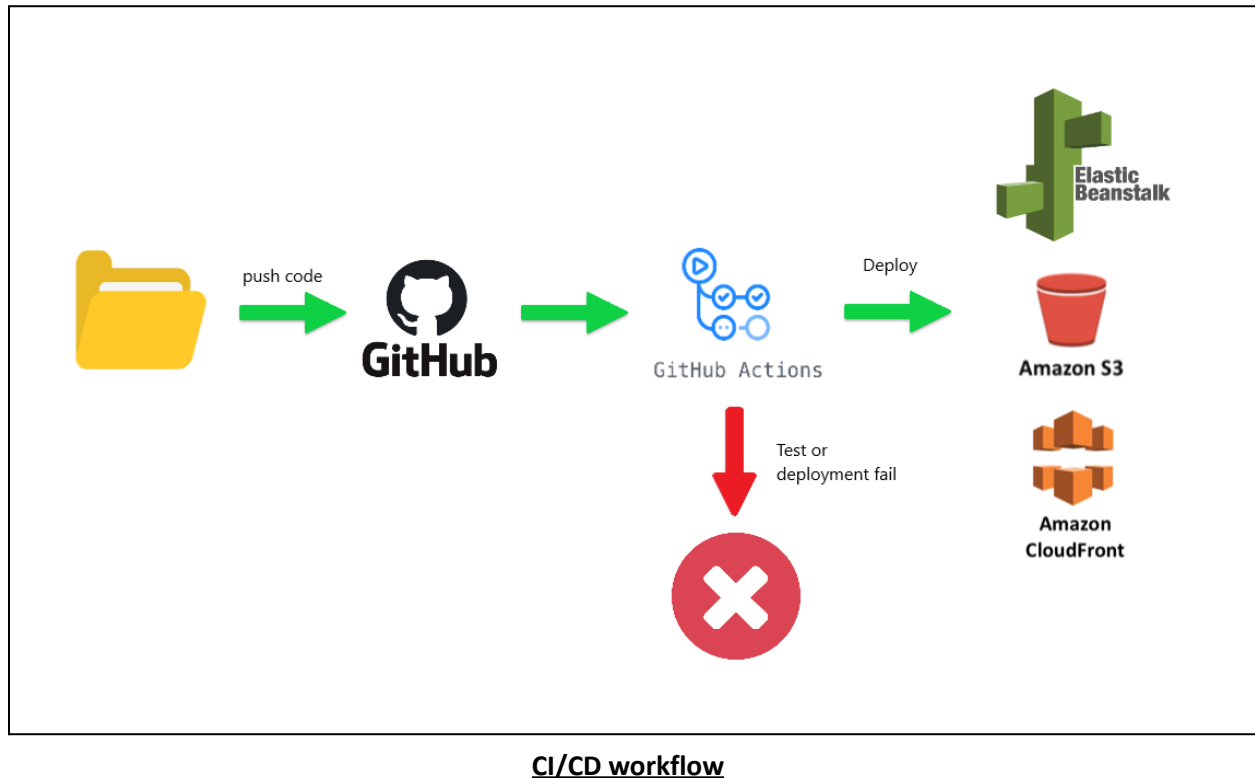
Upon further investigation, this was because react-flippy was compatible with React v17.0.2 but the current React version for our application is v18. There are a few ways that we thought of fixing this issue:

1. Downgrade the npm version. This works because for older npm versions, running npm install will show a warning, while for newer npm versions, it will be an error.
2. Run npm i - --legacy-peer-deps. This works by bypassing peerDependency autoinstallation.
3. Change to another library that is compatible

We decided to adopt solution 3. This is because while option 1 and option 2 are viable solutions, flexibility (the ease with which the system can be modified) is a trade-off as new software engineers will have to modify to an older npm version or run another command instead of the default npm install command. Following option 3, react-transition-group, a much newer compatible library, is used for the implementation which works well for running with old and new npm versions.

5. DevOps Practices

Github Actions is used for CI/CD for the project.



5.1 Continuous Integration (CI)

The CI workflow is configured to run when new code is pushed to any branch in the Github repository. A MongoDB server and Redis server are started in the local test environment. As Node.js is used for implementing the microservices, each service has its own dependencies defined in its respective package.json file. Tests are run using Mocha⁵, which is a test framework running on Node.js. Github actions builds and runs tests for each service. When all tests have passed, the CI workflow has completed successfully.

5.2 Continuous Deployment (CD)

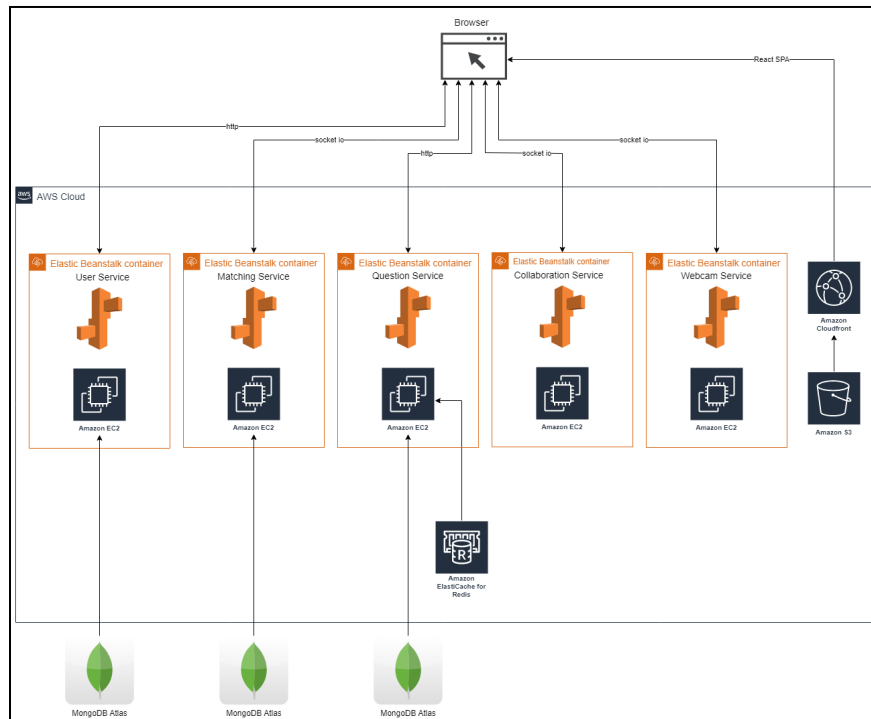
The CD workflow is configured to run only when new code is pushed to the main branch in the Github repository and after the CI workflow has completed successfully. Each microservice is deployed to AWS Elastic Beanstalk⁶. Elastic Beanstalk offers built-in environment monitoring for the application deployed,

⁵ <https://mochajs.org/>

⁶ <https://aws.amazon.com/elasticbeanstalk/>

and Github Actions monitors the status of the Elastic Beanstalk environment for one minute after deployment to determine the success of the deployment. Continuous Deployment will stop when there is a failure in deployment. The frontend is built in the Github Actions environment and deployed to Amazon Simple Storage Service⁷ (S3) and Amazon CloudFront⁸. CloudFront is a content delivery network service that delivers web content stored in the S3 bucket.

5.3 Deployment Details



AWS architecture diagram

Each microservice is deployed on AWS Elastic Beanstalk, with the application running in an Amazon Elastic Compute Cloud⁹ (EC2) instance. Each Elastic Beanstalk environment runs various AWS services, such as Elastic Load Balancing¹⁰ (ELB), AWS Auto Scaling¹¹, S3, AWS CloudFormation¹² and Amazon CloudWatch¹³, and these services are automatically configured for users to deploy their applications easily. Elastic Beanstalk uses nginx as a reverse proxy to map the application to the ELB load balancer on

⁷ <https://aws.amazon.com/s3/>

⁸ <https://aws.amazon.com/cloudfront/>

⁹ <https://aws.amazon.com/ec2/>

¹⁰ <https://aws.amazon.com/elasticloadbalancing/>

¹¹ <https://aws.amazon.com/autoscaling/>

¹² <https://aws.amazon.com/cloudformation/>

¹³ <https://aws.amazon.com/cloudwatch/>

port 443, enabling HTTPS for encryption of data transferred between browsers and the microservices. Self-signed SSL certificates are used for the microservices.

For the frontend, the dynamic web content is stored in a S3 bucket and when there is a request for the content, Cloudfront will deliver the content. Cloudfront is configured to redirect HTTP to HTTPS. The reason for using HTTPS is not only to provide secure connections, but also to enable the feature to use webcam and microphone for the browsers, as most browsers require a website to be secure to grant these permissions.

The User Service, Matching Service and Question Service use separate MongoDB Atlas clusters for persistent storage. The Question Service also uses Amazon ElastiCache for Redis¹⁴ for data caching.

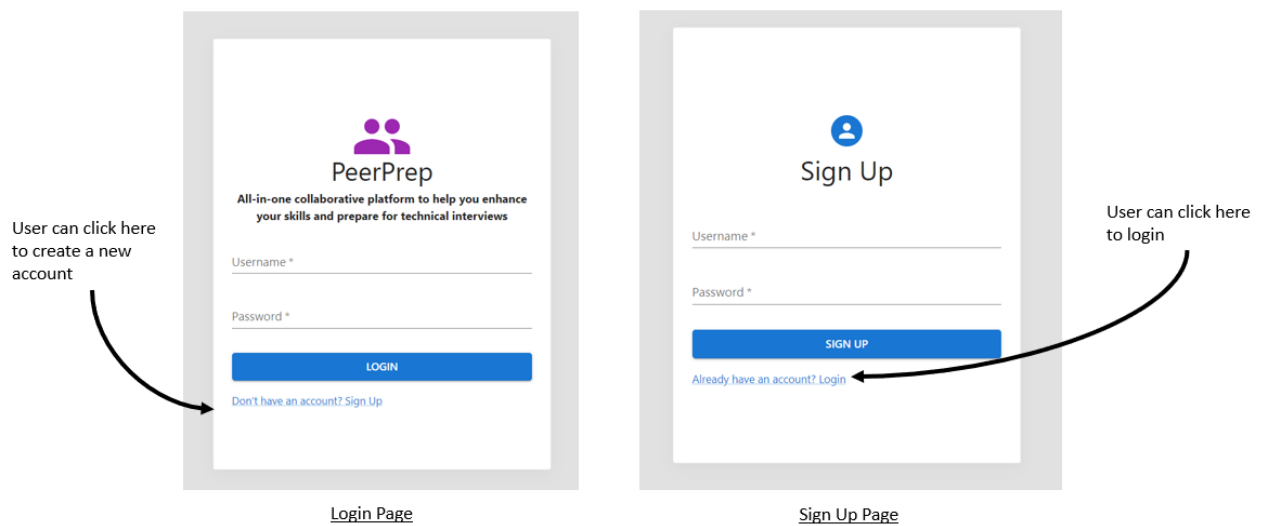
Links to deployed applications can be found at the last page of this report.

¹⁴ <https://aws.amazon.com/elasticache/>

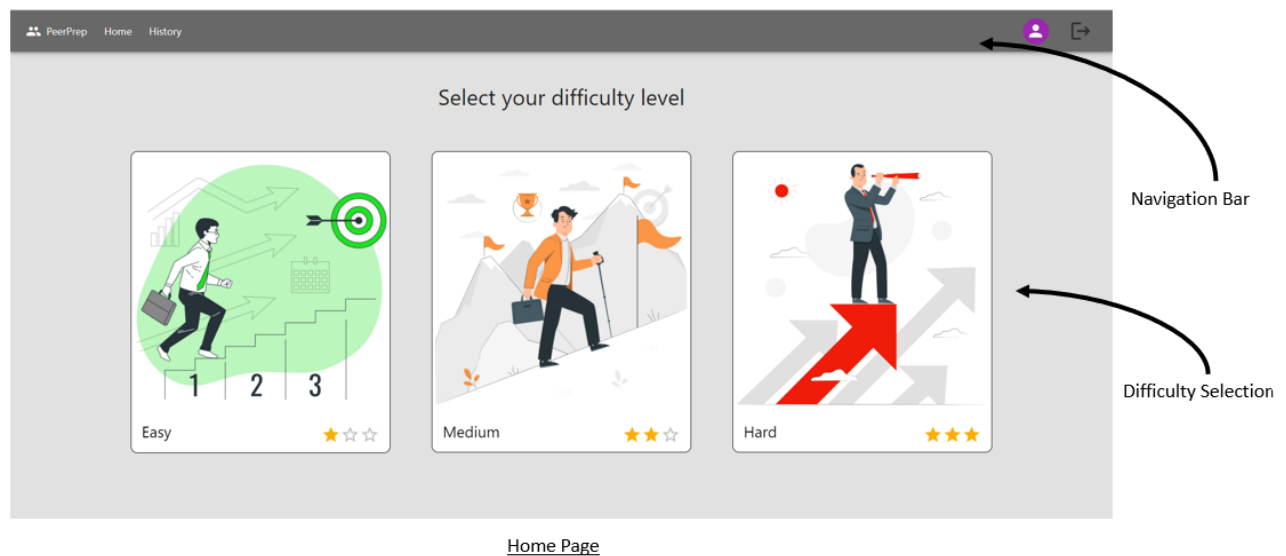
6. User Interface

This section displays the user journey, from the user arriving on PeerPrep to visiting the key touchpoints on the site to eventually logout.

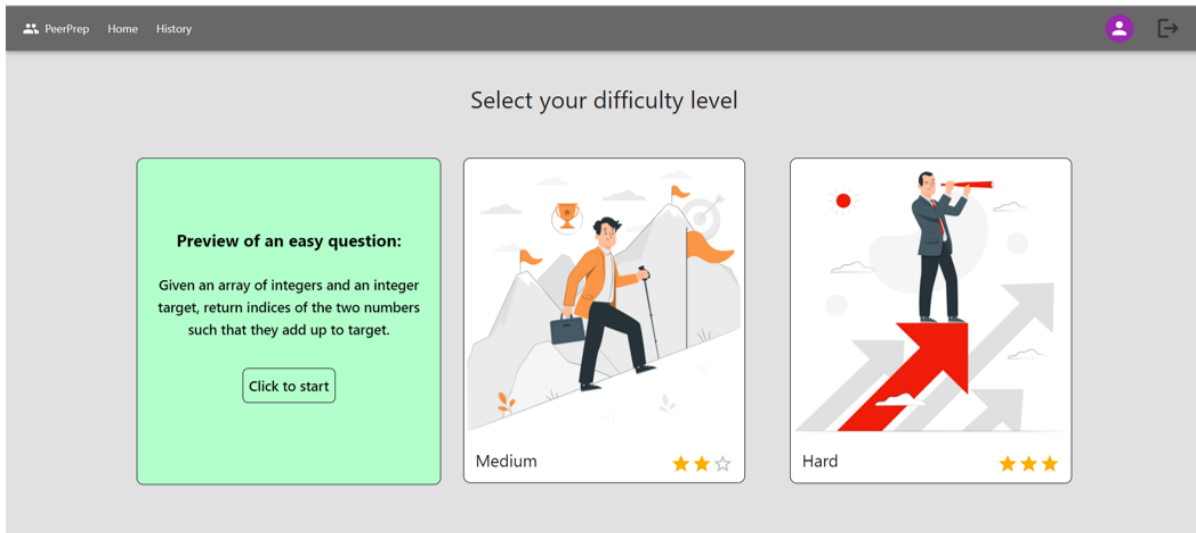
Upon arriving on PeerPrep, the user can 1. Login with an existing account or 2. Sign up to create a new account



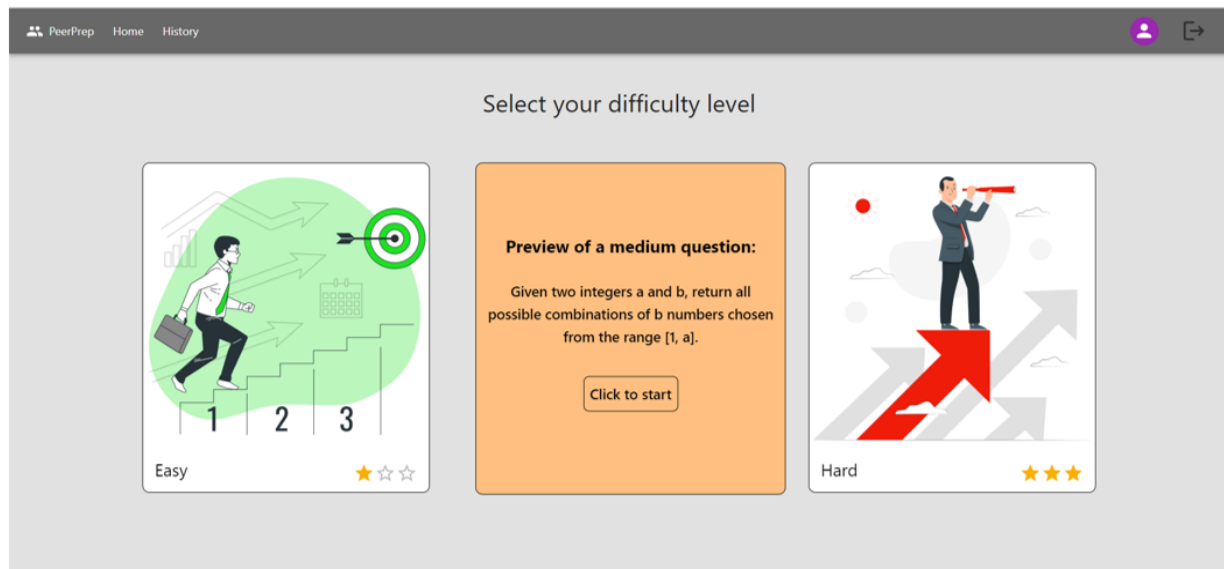
After the user is logged in, the homepage is displayed



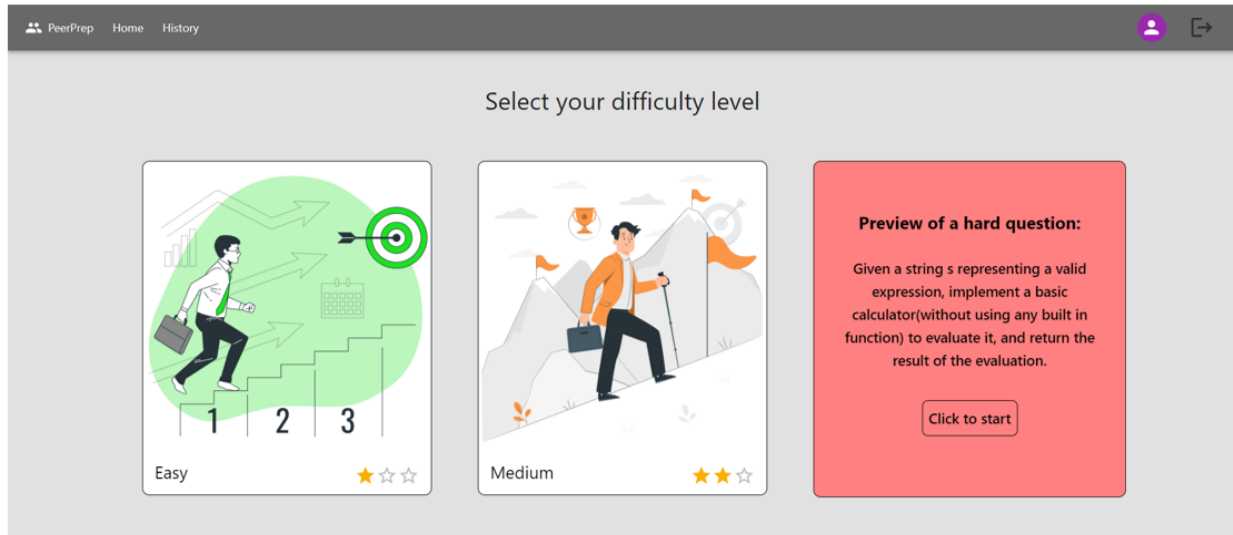
The user can hover over the question difficulty level to get a sample question for that difficulty level.



Hovering over Easy difficulty

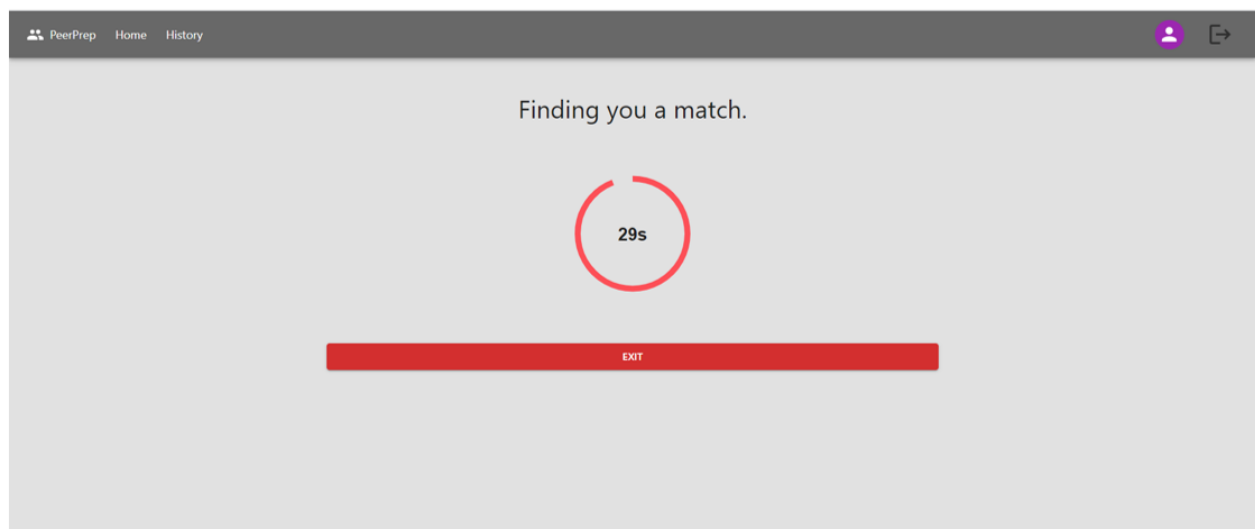


Hovering over Medium difficulty



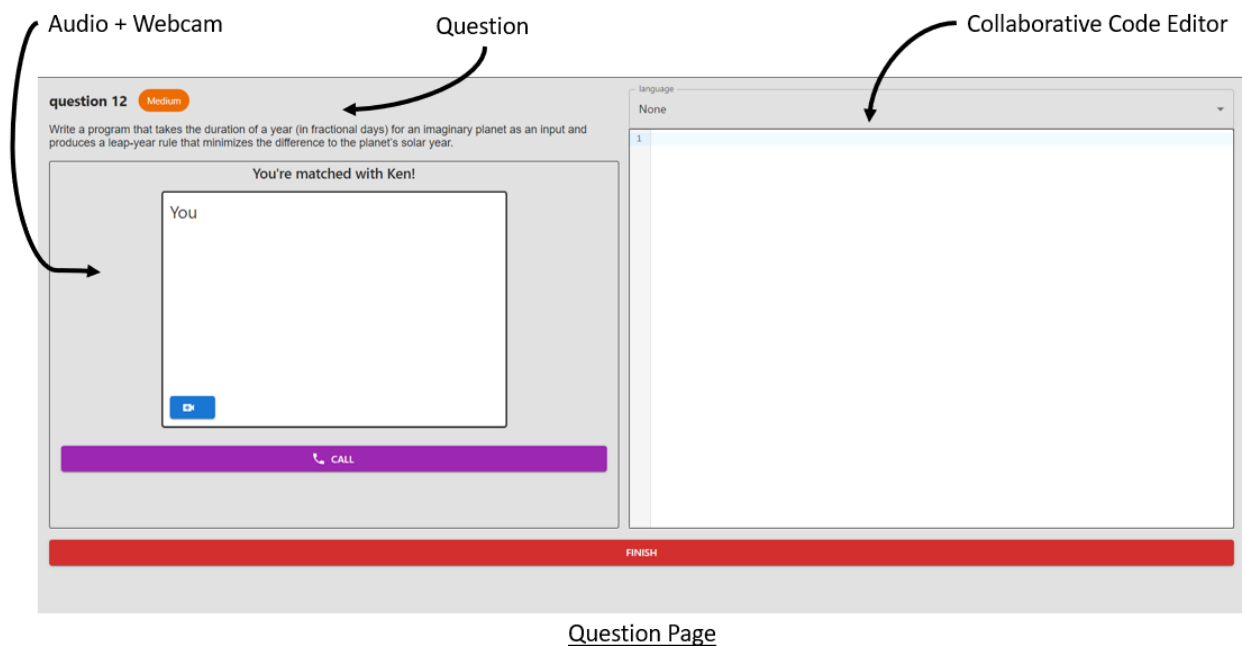
Hovering over Hard difficulty

After deciding which question difficulty level to attempt, the user can click on the question difficulty, bringing the user to a loading page. In this page, there is a countdown timer of 30 sec during which the system will search for a match with the same difficulty level.

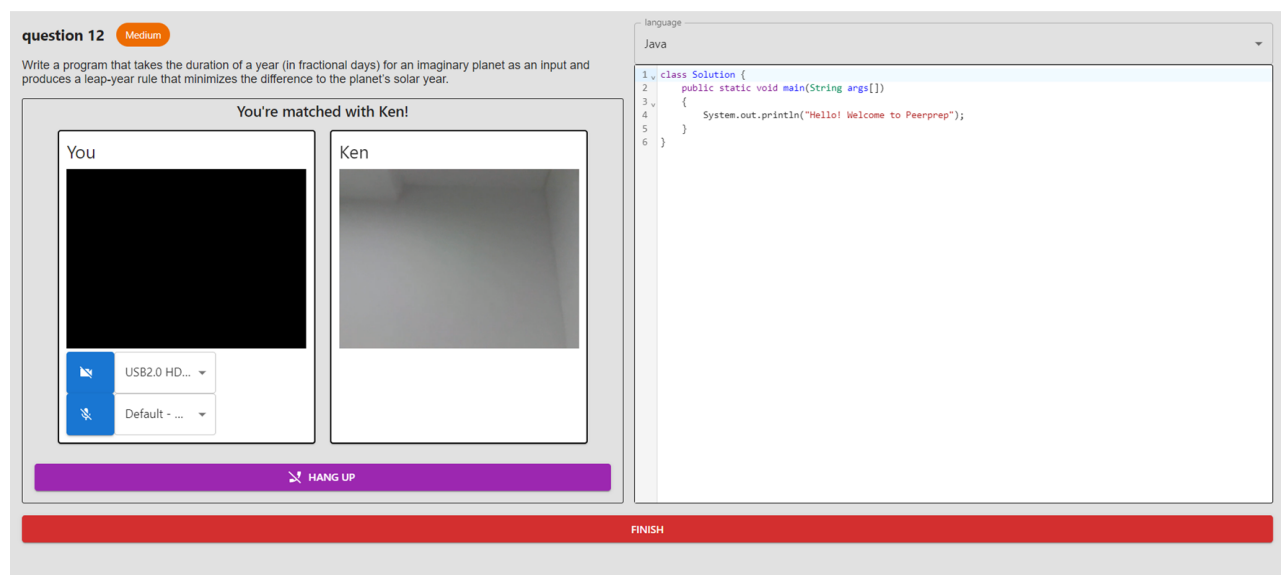


Loading Page

If there is no match after 30 sec, the user will be directed back to the home page. If there is a match, the question page is displayed.

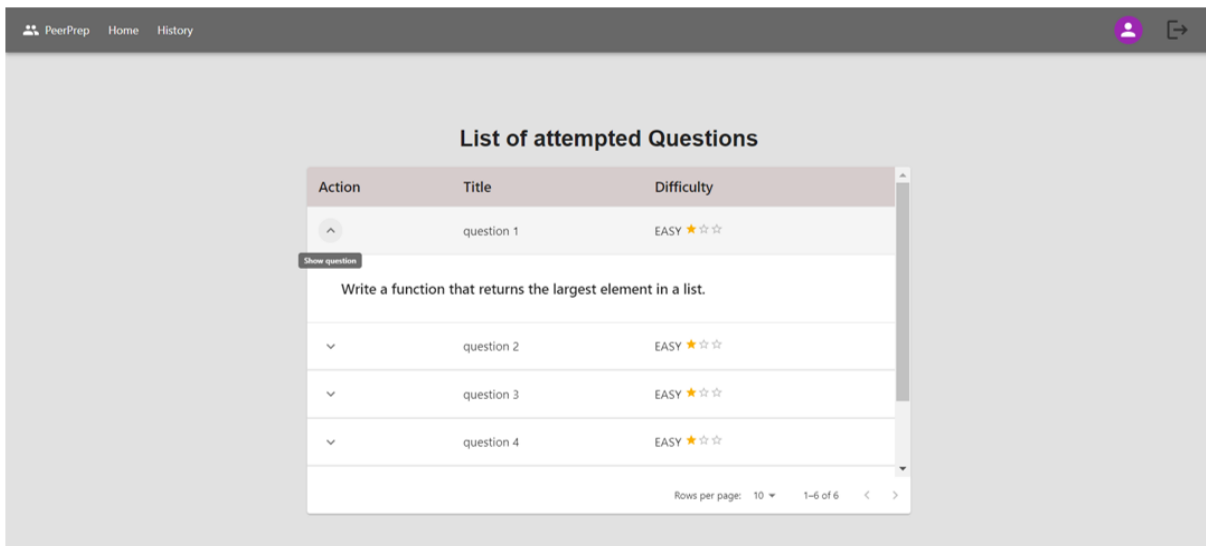


When the user is ready to start, the user can select the coding language and start coding. The user can also share his audio/camera with the other match. Both users (the users and matched peer) can view the code being updated in real time, see each other face-to-face and hear each other.



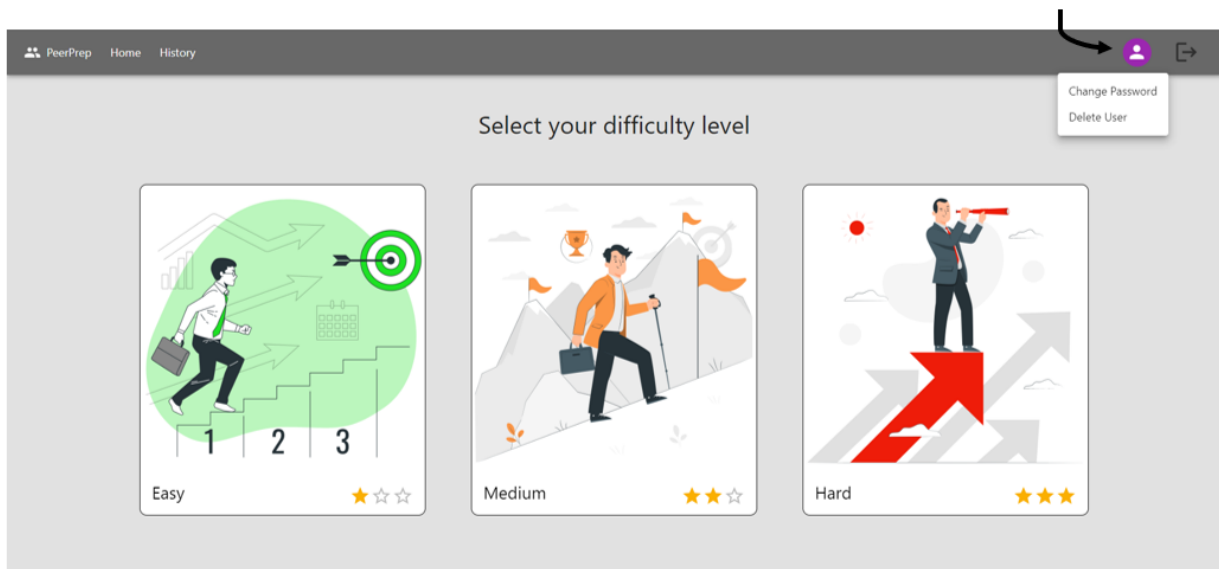
When the user finishes the question, the user can click on the finish button, returning both users back to the home page.

The user can also view the history of attempted questions by clicking on the history tab in the navbar.

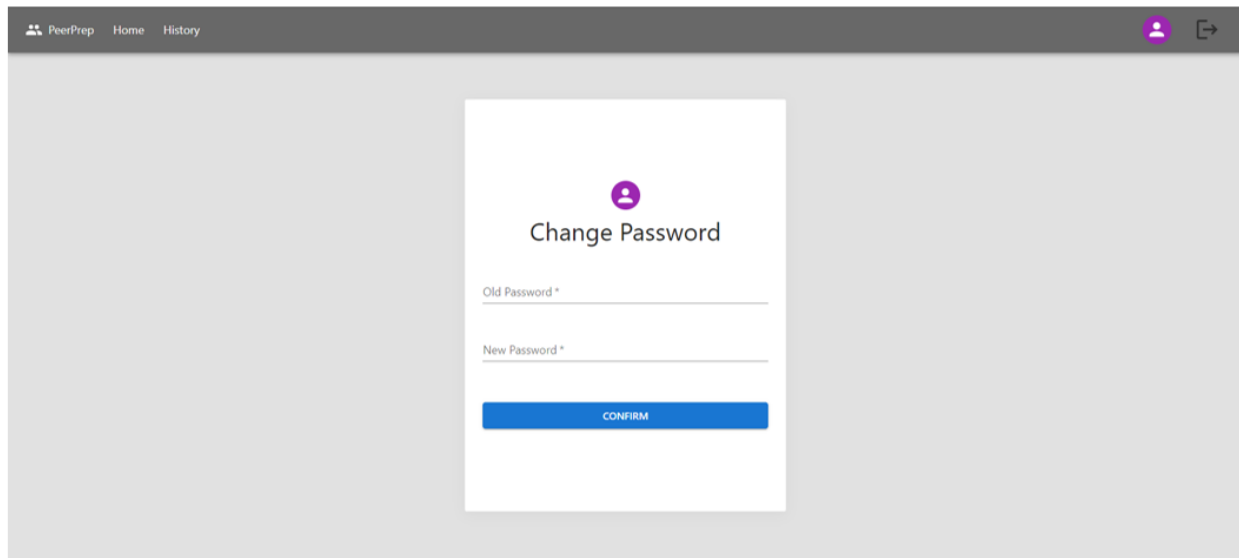


History Page

If the user would like to change password or delete user, the user can click on the user icon

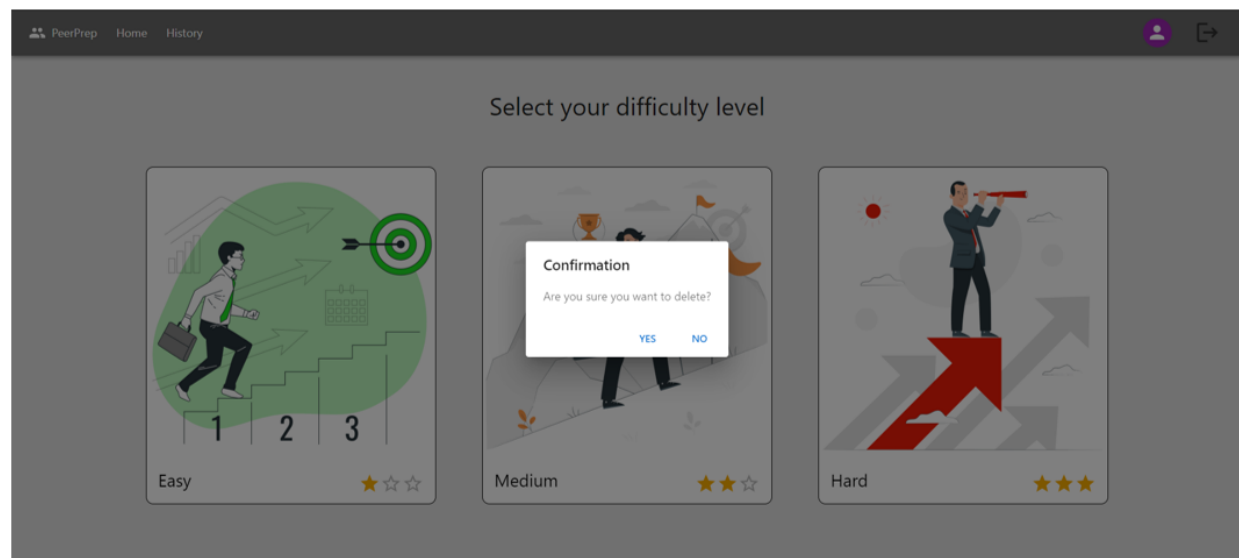


When the user clicks on Change Password, the Change Password page will be displayed. Filling in the correct old password and a new password, the password will be successfully changed.

The screenshot shows a web application interface with a dark grey header bar. On the left of the header are the links 'PeerPrep', 'Home', and 'History'. On the right are a user profile icon and a share icon. The main content area is light grey and features a white modal box titled 'Change Password'. Inside the modal, there is a purple user icon, followed by the title 'Change Password'. Below the title are two input fields: 'Old Password *' and 'New Password *'. At the bottom of the modal is a blue button labeled 'CONFIRM'.

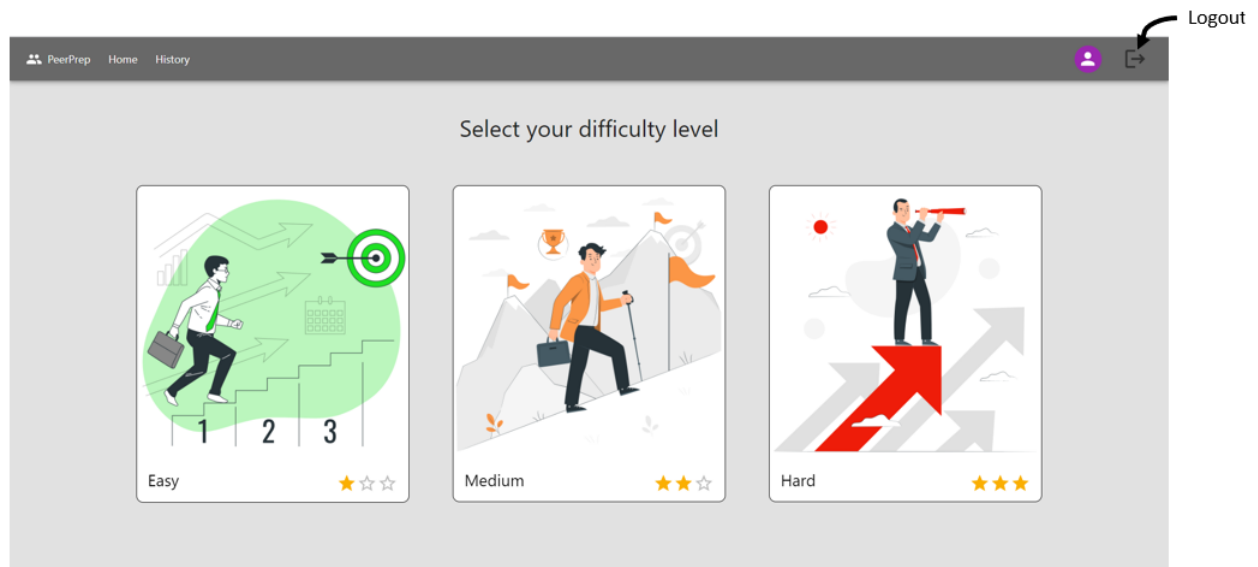
Change Password Page

When the user clicks on Delete User, a confirmation will be prompted to the user. Clicking on yes will delete the user account..

The screenshot shows a web application interface with a dark grey header bar containing 'PeerPrep', 'Home', 'History', a user profile icon, and a share icon. The main content area is dark grey and titled 'Select your difficulty level'. It displays three difficulty level options: 'Easy' (illustrated with a person climbing steps), 'Medium' (illustrated with a person climbing a mountain), and 'Hard' (illustrated with a person standing on a large red arrow). Each option has a star rating below it. A white confirmation dialog box is overlaid on the 'Medium' option, asking 'Are you sure you want to delete?' with 'YES' and 'NO' buttons.

Delete User Confirmation

Finally, the user can log out of PeerPrep.



7. Improvements and enhancements

7.1 Question Service: Increase variety of questions and viewing question bank

One way we can increase the variety of questions is by allowing the submission of questions. Currently, the backend of the question service supports creation and deletion of questions. In addition to these, there needs to be a method to verify the validity of these questions before they are approved for display. In addition, we could allow for the viewing of the questions available in the question bank

7.2 Allow for the compilation of code + Running test cases

In some interview settings, the code has to be compiled and several test cases are run. This is an improvement that we can make to our existing PeerPrep in order to help simulate an interview environment. In order to implement these new changes that we propose, we will need to create a Code Service that will be responsible for compiling the code and test cases.

7.3 Quality of life features for matching service

There might be instances when users find it hard to be matched within 30s. In these instances, there could be two options presented to them.

- (1) Continue doing the question alone
- (2) Continue waiting

Currently, when users are not matched within 30s, they are rerouted back to the question selection page. We could improve on this by presenting them with the above options upon timeout.

In addition, besides selecting questions based on difficulty, the user could also indicate preference for the topic of questions to attempt. This allows the users to practice questions that they are weaker at. To support this change, changes have to be made to the matching service (To match based on topic + difficulty), and the question model (To also include a topic attribute).

7.4 Deployment of Application

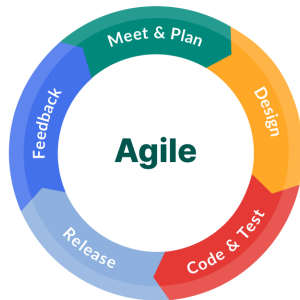
The current way the frontend and microservices are deployed is that they have unique domains. This is not ideal as the authentication cookie should be configured such that the browser will send the cookie to sites with the same domain for better security. One improvement we can make is to use Amazon Route 53¹⁵ to route traffic to each microservice so that they share the same domain.

¹⁵ <https://aws.amazon.com/route53/>

8. Project Management

In general, we have divided PeerPrep into its respective subcomponents, such that each of us takes ownership of the feature. In addition, we all take active steps to ensure that the deadlines are met for each milestone. We adopted the Agile software development methodology in order to respond to changes quickly.

GANTTPRO



We have broken down the deliverables into the 3 milestones, each corresponding to a sprint cycle, as shown in the table below. Within the milestones, we conduct weekly group check-ins to help remove blockers and ensure that the features are delivered timely. We have prioritized requirements that we have marked as high.

| Milestone 1 | Milestone 2 | Milestone 3 |
|--|---|--|
| <p>Key Deliverable: MVP of the product with a basic front-end</p> <p>Deliver the essential services of the 2 Must-Haves.</p> <ul style="list-style-type: none">- User-Service- Matching Service <p>Deliver a basic UI to demonstrate the concept.</p> | <p>Enhancements to the UI</p> <p>Enhance the User-Service to support greater functionality for account management (Changing password, deleting account)</p> <p>Enhance the Matching-Service to handle edge cases</p> <p>Develop the Collaboration Service.</p> <p>Develop the Question Service.</p> <p>Develop Webcam Service</p> | <p>Enhancements to the UI</p> <p>Enhance the Question Service to support getting a history of questions.</p> <p>Testing, bug fixes</p> <p>Deploying the product.</p> |

9. Reflection

Our team is grateful for the opportunity to work on the PeerPrep project as it has provided us with exposure to modern software engineering practices. Through the project, we are able to work and experiment with different tech stacks. We also learn the importance of good, deliberate design in order to meet the requirements of the product. The requirements of the product are interesting, and being able to deliver the features is also satisfying. In the process, we have also recognised that each one of us has different strengths and weaknesses. Coming together as a group to work on the project helped tremendously when we are trying to overcome technical challenges and making PeerPrep work.

10. Others

To view the deployed application, visit the following https links and trust the self-signed certificates

<https://matserv-env.eba-ei3vpcg2.ap-southeast-1.elasticbeanstalk.com/>

<https://userservice-env.eba-p3j8behu.ap-southeast-1.elasticbeanstalk.com/>

<https://webserv-env.eba-jwtv3m3g.ap-southeast-1.elasticbeanstalk.com/>

<https://collab-env.eba-jd3df2b9.ap-southeast-1.elasticbeanstalk.com/>

<https://quessvc-env.eba-2agrnkqm.ap-southeast-1.elasticbeanstalk.com/>

Frontend:

<https://dj17yv8d2l9t3.cloudfront.net/>