CS3219

Software Engineering Principles and Patterns

PeerPrep

Group 40 Project Report

Source code:
https://github.com/CS3219-AY2223S1/cs3219-project-ay2223s1-g40

Deployed web app:
http://peerprepfe.s3-website-ap-southeast-1.amazonaws.com/

Alexander Charles Er Zhenwei	A0214562L
Chong Kok Leong	A0214844E
John Alec Mendoza Branzuela	A0201504B
Jungbae Kim	A0194438U

Table of Contents

1. Background and Purpose of the Project

1.1 Background

In the current age of Big Tech Companies (MANGA), High-Frequency Trading Firms (HFTs) and Start-up Unicorns, difficult and demanding technical interviews have been increasingly employed to aid in the candidate selection process. Many students and graduates often find it challenging to handle the technical complexity of these interviews, and report finding it difficult to demonstrate their thought processes and considerations to the interviewer, while focusing on cracking the highly complex problem sets.

As they say, practice makes perfect. However, technical interview preparation can be a soul-crushing process. Generally, students are advised to tackle as many “Leetcode” style questions as often as possible, with the intent to hone their problem-solving abilities from encountering as many diverse questions and problems as possible, as well as the “muscle” memory” from the sheer number of repetitions. This is often highly tedious and dreary, and many find it hard to feel excited about and commit to.

As students ourselves who have faced this issue first-hand, we have been tasked to create an interview preparation platform and peer matching system called PeerPrep as part of our CS3219 coursework. PeerPrep is a platform that allows students who are looking to practise their technical interview skills to be matched with a like-minded peer, and aid in this process through peer learning and collaboration.

After creating an account with the platform and logging in, a student will be able to select the desired question difficulty, either easy medium or hard to practise. Then the platform will match him or her with a fellow student that is looking to tackle the same difficulty level. Should there not be an available peer, the system will be timed out after 30 seconds and prompt the student to retry to find a peer.

When a peer is successfully found, both students are provided with the same question and a collaborative text field in which they are expected to type their solution. This free text field is like Google Docs as it is updated in near-real time, thus allowing both students to collaborate on the provided question.

When students have finished working on the question and have determined that they are ready to submit their work, they can click the submit button, which will bring up a prompt and confirm their intention to submit. Upon submission, they will be returned to the main page where they are able to attempt to look for another peer and tackle a new question or log out to end their session. The peer that they were matched with in the previous session will get a notification that their partner has ended their session and can either continue to work on the problem or submit it.

1.2 Purpose of PeerPrep

As mentioned in the background information, many students and graduates often find it challenging to handle and adequately prepare for the complexity of technical interviews. They also report finding it difficult to develop and train the ability to demonstrate their thought process and considerations to the interviewer, while focusing on cracking the highly complex problem sets.

Hence, the purpose of PeerPrep is to create a web application that seeks to help students better prepare themselves for these highly challenging technical interviews. This is achieved by providing a solution that enables peer learning and support. Instead of jumping on the technical interview question grind alone, why not do it with a fellow student and not only help each other learn, but also keep each other engaged and feel supported.

PeerPrep helps users to find a fellow peer who wants to work on a problem of similar difficulty and allows them to work together and communicate with each other on the problem to achieve the purpose stated above.

2. Individual Contributions to the Project

Alexander Er	Non-Code ● Documented the Frontend ● Documented the Collaboration Service ● Documented the Chat Service
	Code ● Implemented the Difficulty page ● Implemented the Countdown page ● Implemented the Room page ● Implemented the Question style and formatting ● Configured the web sockets for the Frontend ● Implemented web socket Testing
Chong Kok Leong	Non-Code ● Documented the Background information ● Documented the Question service ● Documented the Application Requirements and Specifications ● Documented the Reflections ● Project Report and Documentation formatting
	Code ● Configured and set-up the MongoDB Cluster ● Implemented the Question Service ● Implemented the Chat Service
John Alec Mendoza Branzuela	Non-Code ● Documented the User Service ● Documented the Development Processes ● Documented the Requirements Prioritisation.
	Code ● Implemented the User Service ● Implemented the User Store with Zustand ● Implemented the Authentication Page ● Implemented the Profile page ● Implemented the Routing ● Implemented the Layout ● Implemented the Navbar ● Implemented the 404 Not found page ● Contributed to the Room page ● Contributed to the Difficulty page ● Implemented unit testing for User Service Controller
Jungbae Kim	Non-Code ● Took charge of Project management. ● Led weekly project meetings ● Defined Sprint deliverables

	• Maintained list of FRs and NFRs • Documented Architectural design • Documented Matching Service
	Code • Implemented the Matching Service • Implemented the Collaboration Service • Configured the Socket.io client-server interactions for the Chat Service. • Managed interactions between the frontend and micro services on the room page. • Developed CD script for the micro services and frontend

3. Application Requirements and Specifications

3.1 Function Requirements

3.1.1 User Service

ID	Description	Priority
FR-U-1	The system should allow users to create an account with username and password.	High
FR-U-2	The system should ensure that every account created has a unique username.	High
FR-U-3	The system should allow users to log into their accounts by entering their username and password.	High
FR-U-4	The system should allow users to log out of their account.	High
FR-U-5	The system should allow users to delete their account.	High
FR-U-6	The system should allow users to change their password.	Medium
FR-U-7	Upon user authentication, the system must redirect the user to the home screen.	High
FR-U-8	If the authentication fails, the login page will provide feedback prompting the user to attempt to login again.	Low

3.1.2 Matching Service

ID	Description	Priority
FR-M-1	The system should allow users to select the difficulty level of the questions they wish to attempt.	High
FR-M-2	The system should be able to match two waiting users with the same difficulty levels and put them in the same room.	High
FR-M-3	If there is a valid match, the system should match the users within 30s.	High
FR-M-4	The system should inform the users that no match is available if a match cannot be found within 30 seconds.	High

FR-M-5	The system should provide a means for the user to leave a room once matched.	Medium
FR-M-6	The system will stop looking for a match when a user exits from the page or chooses to cancel the matching.	High
FR-M-7	Once a user cancels the matching request and/or exits from the page, the database should accordingly remove the associated match model for that user.	Medium
FR-M-8	The system should indicate to the other user if their peer has ended their session after submitting the question.	Low

3.1.3 Question Service

ID	**Description**	**Priority**
FR-Q-1	The system should display and provide a question upon a successful match.	High
FR-Q-2	The system should provide a random question that corresponds to the difficulty level the user has chosen for the match.	High

3.1.4 Collaboration Service

ID	**Description**	**Priority**
FR-CO-1	The system should display a blank document upon a successful match to the users.	High
FR-CO-2	The system should allow users to edit the contents of the document, as well as to configure the style of the document.	High
FR-CO-3	The system should be able to update the state of the document with the changes of the users during the session in real-time.	High

3.1.5 Chat Service

ID	Description	Priority
FR-CH-1	The system should display an empty chat box upon a successful match.	High
FR-CH-2	The system should allow users to send a text message.	High
FR-CH-3	The system should be able to display the messages sent by the two users, sorted by their sending time.	High

3.2 Non Functional Requirements

3.2.1 Integrity Requirements

ID	Description	Priority
NFR-I-1	The system can only discard the contents written to the document after both users in the session choose to leave.	Medium

3.2.2 Robustness Requirements

ID	Description	Priority
NFR-R-1	The question that is fetched and displayed to the room page should persist in case of an intermittent network drop.	High

3.2.3 Performance Requirements

ID	Description	Priority
NFR-P-1	The system should be able to fetch a question from a database within one second.	Medium
NFR-P-2	The system should be able to update and reflect the changes on the collaboration document made by both users within 0.5 seconds.	Medium
NFR-P-3	The system should be able to update and reflect the chats sent by both users within 0.5 seconds.	Low
NFR-P-4	The frontend website should be served within 2 seconds of a request.	Medium

3.2.4 Security Requirements

ID	Description	Priority
NFR-S-1	Users' passwords should be hashed and salted before storing in the DB.	High
NFR-S-2	The system should only allow an admin to create and delete a question inside the database.	Medium
NFR-S-3	Should not share any personal information of the user with any 3rd party organisation.	High

3.2.5 Usability Requirements

ID	Description	Priority
NFR-U-1	The website should be usable from all modern browsers such as Chrome, Firefox and Safari.	High

3.3 Requirements Prioritisation

After looking at the requirements table, we prioritised the requirements according to 2 criteria. The first criterion is whether the requirements are must-have or nice-to-have features. The second is deciding if the requirements are critical to the user's experience on the application by putting ourselves in the perspective of the user.

From this, we highly prioritised authentication and Create-Read-Update-Delete (CRUD) functionalities as they were the most practical to the user and had high overlaps with the must-have features. Furthermore, we also prioritised the User Service and Matching Service first when deciding which microservice to work on as our team believed that it was the core of our minimum viable product (MVP).

4. Developer Documentation

This section discusses our development processes and models, design principles and various decisions and trade-offs we had to make.
If you are looking specifically for developer documentation to work on the project itself, please refer to our GitHub README, which contains the comprehensive technical details of how to install and run the project.

4.1 Development processes

4.1.1 Agile Software Development Cycle

For the software development model, we adopted Agile SDLC model, with regular stand-up meetings to plan and review the Sprints and to groom our backlog items. We had a total of 6 Sprints and used Notion to keep track and plan our Sprints and backlog items.

4.1.2 Branching workflow

Since this project was worked on by multiple developers concurrently, we have decided to adopt the Git branching workflow. As such, direct commits to the master branch are not allowed and every commit to master must be done via a pull request which must then first be approved by at least one other developer. We believe the benefits of having an additional developer spot potential bugs or propose alternative solutions will ultimately outweigh the disadvantages of time lost and the overhead of setting up additional PRs.

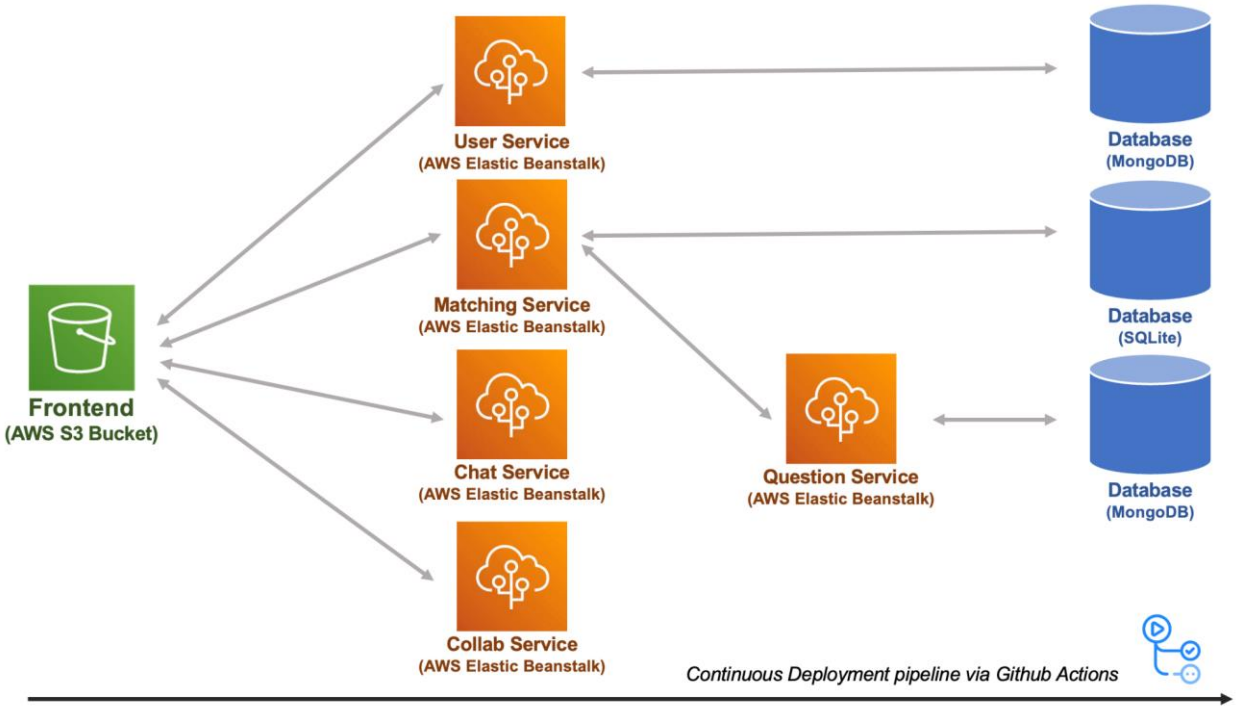
4.1.3 Automated deployment

To improve the speed of the development and to provide the most up-to-date, high-quality product to the end-users, continuous deployment was used, with the help of GitHub Actions. Once pushed to the release branch, the script analyses the frontend and each of the microservices and deploy them in appropriate destinations (mainly AWS S3 Bucket and Elastic Beanstalk).

4.2 Technology Stacks used:

Frontend	React.js
Backend	Node.js, Nest.js, Express.js
Database	MongoDB, SQLite
Deployment	AWS Elastic Beanstalk
Pub-Sub Messaging	Socket.io
Cloud Providers	AWS
CI/CD	GitHub Actions
Project Management Tools	Notion

4.3 Software Architecture



The above diagram illustrates the entire software architecture for PeerPrep. For the frontend, which consists of client-side scripts serving static contents, AWS S3 Bucket is deemed to be an applicable choice for deployment due to its high availability, deployability, and security.

PeerPrep follows micro-services architectural design, which possesses numerous benefits. First, it allows members to work independently on different services at the same time. Second, independent services enable the members to use different technology stacks of their confidence, hence leveraging on the team members' technical expertise. Finally, each service, as opposed to monolithic architecture, can be scaled independently from one another, which prevents unnecessary services from being scaled up. Each micro-service is deployed in AWS Elastic Beanstalk, which reduces configuration overhead, and handles the auto-scaling of resources for the developers.

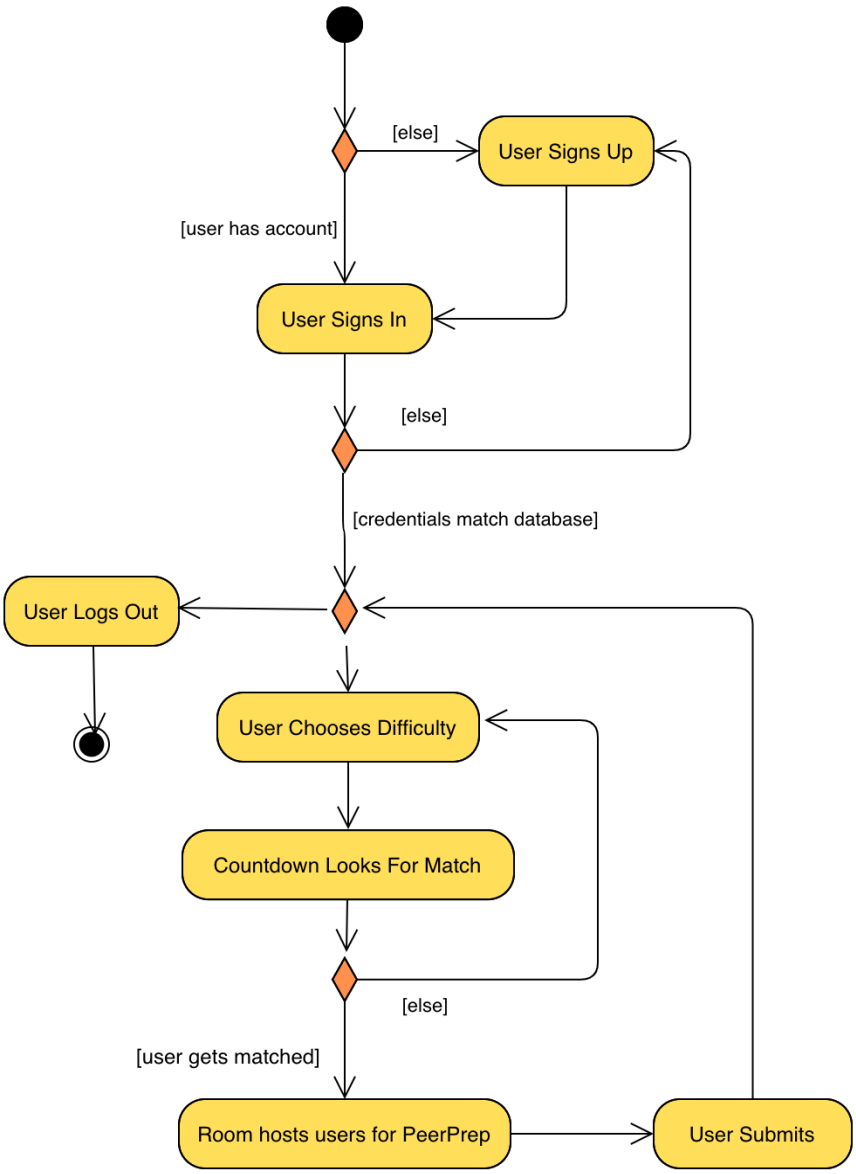
The question service is not directly called by the frontend; instead, the matching service calls the question service via HTTP GET request and distributes the question to the users instead. The rationale and the detailed interaction are given in the matching service section.

4.4 Frontend

4.4.1 Introduction

The frontend provides the main interface users will interact with to utilise PeerPrep. The frontend was developed using React, with the assistance of Material UI (MUI) and Chakra. We decided to use React as it was high in flexibility, and it had the ability to easily reuse components. MUI and Chakra were used as they are comprehensive, and they complement React as they are well supported UI frameworks.

4.4.2 User Journey



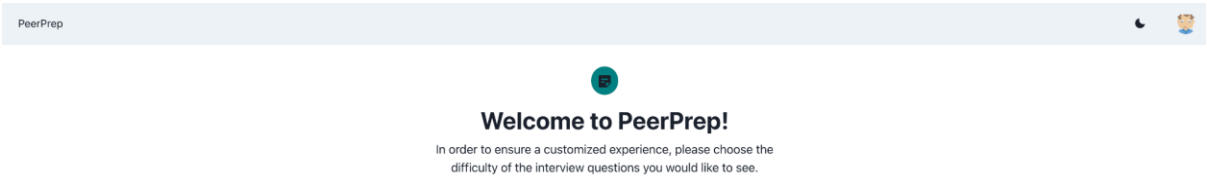
In general, a new user will go through the following journey to access PeerPrep. Firstly, they will encounter the Sign In page that will request for their username and password.

If the user has not registered yet, they will be redirected to a Sign-Up page to register a new account. Afterwards, they will be redirected to the Sign In page again. After signing in, the user will be able to access the Difficulty Page. After choosing a difficulty, they will be automatically redirected to the Countdown Page to match with a user. Upon a successful match, the user will be navigated to the Room Page and can begin using PeerPrep's main services.

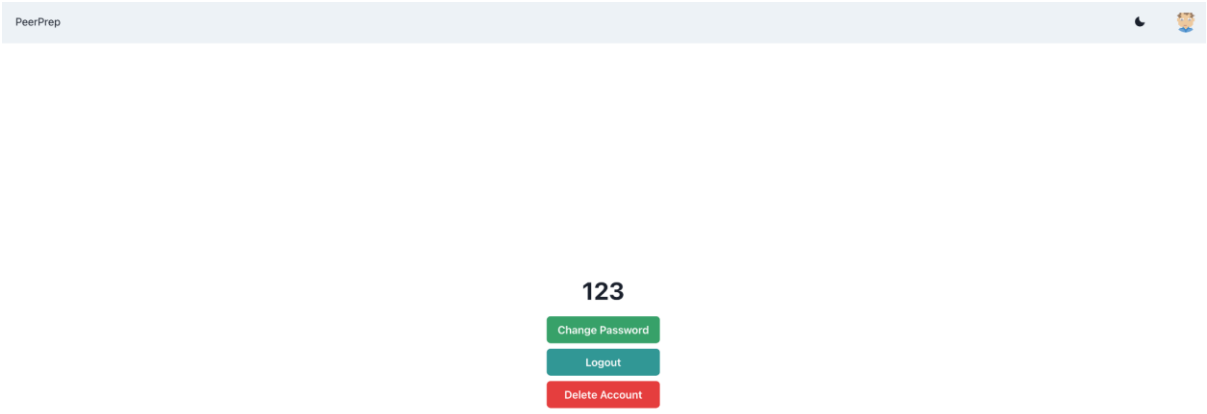
4.4.3 Components

Apart from the main pages, we have also implemented a Navigation Bar and Private Routing. The Navigation Bar acts as a mini hub that allows the user to access certain commands. For example, the user can use the Navigation Bar to logout of his account. The Navigation Bar is designed as a modular component, which allows us to easily import it across several pages, abiding by the DRY (Don't Repeat Yourself) principle. For example, the Navigation Bar can persist as shown in the pictures below. This flexibility allows us to easily reuse it throughout the application.

In addition, we have also implemented Private Routing. Private Routing ensures that the user cannot access restricted parts of the application if he is not authorised. For example, if the user attempts to directly enter the room page (via URL manipulation), the Private Router will detect this and immediately redirect the user to the Sign In page. This will enhance the security of the application.



Navigation Bar in Difficulty Page



Navigation Bar in Profile Page

4.5 User Service

4.5.1 User Authentication

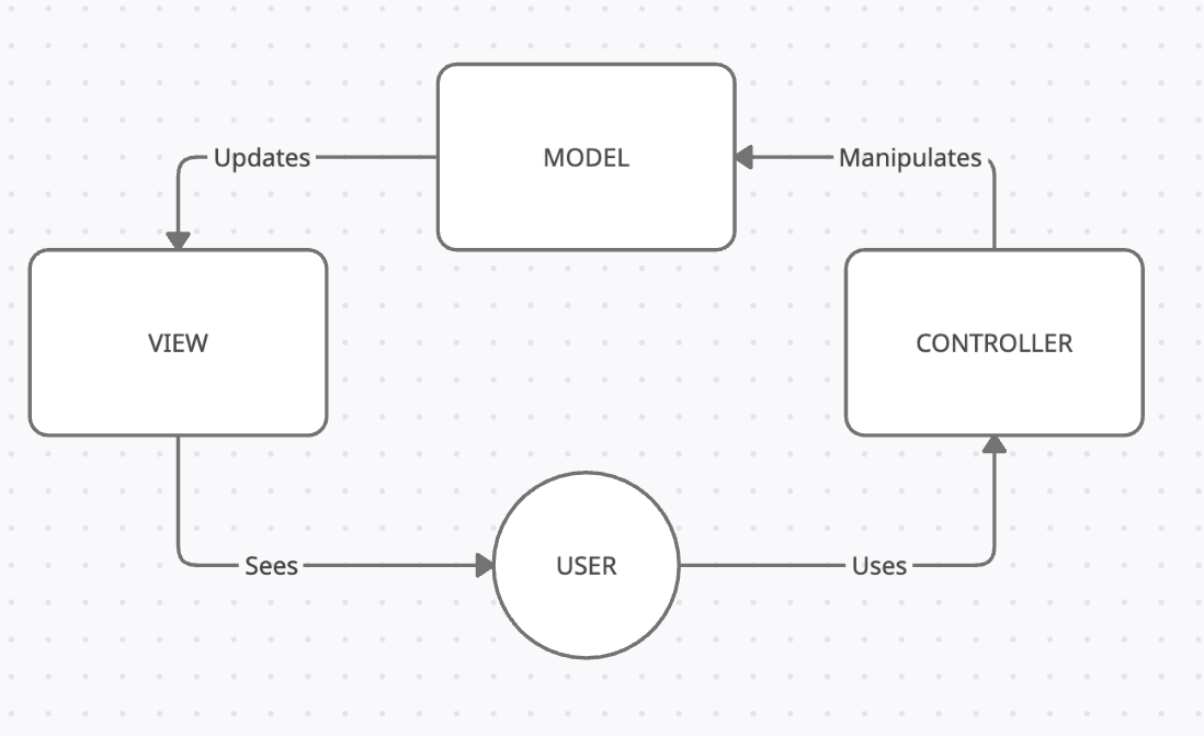
The user service is responsible for implementing user authentication and authorization. Once the user creates an account, we make use of the Bcrypt library to hash the password and store it in the database in our backend.

The backend is built on Express.JS with MongoDB as our choice of database. The reason why we chose MongoDB came down to two criteria: developer experience and cost. Our team was more comfortable with NoSQL databases since we have more industry experience working with it. Furthermore, it was much faster to prototype with a NoSQL database compared to a SQL database as in the initial phases of development, the schema will change quite frequently and using a NoSQL database allows us to be flexible with our schema. Secondly, MongoDB Atlas provides a generous free tier from which we can easily deploy our prototype to.

The user service follows a Model View Controller Pattern (MVC) which is used predominantly to support the storage, retrieval, and display of information to end users. We decided to adopt the MVC pattern for our user service due to its many benefits. The table below lists down the benefits and justifications for using the MVC pattern.

Benefits	**Justifications**
Better modularity and reusability	The model view controller (MVC) pattern enforces the **separation of concerns principle.**
Accelerates developmental processes	As coupling between the Model, View and Controller components are reduced, each team member will be able to work on each of the 3 components concurrently. This paves the way for parallel development and potentially allows us to complete development 3 times faster.
Potentially reduces code duplication	As web applications are constantly evolving, we may plan to create multiple Views for our model to serve different devices. In light of this possible project enhancement, MVC reduces the need for code duplication as it separates our display from our business logic and data components.

Enhances UI modification capabilities	Since components are more loosely coupled, changes in the View components will less likely require changes in the Model component. This is essential as UI changes are usually frequent for web applications.

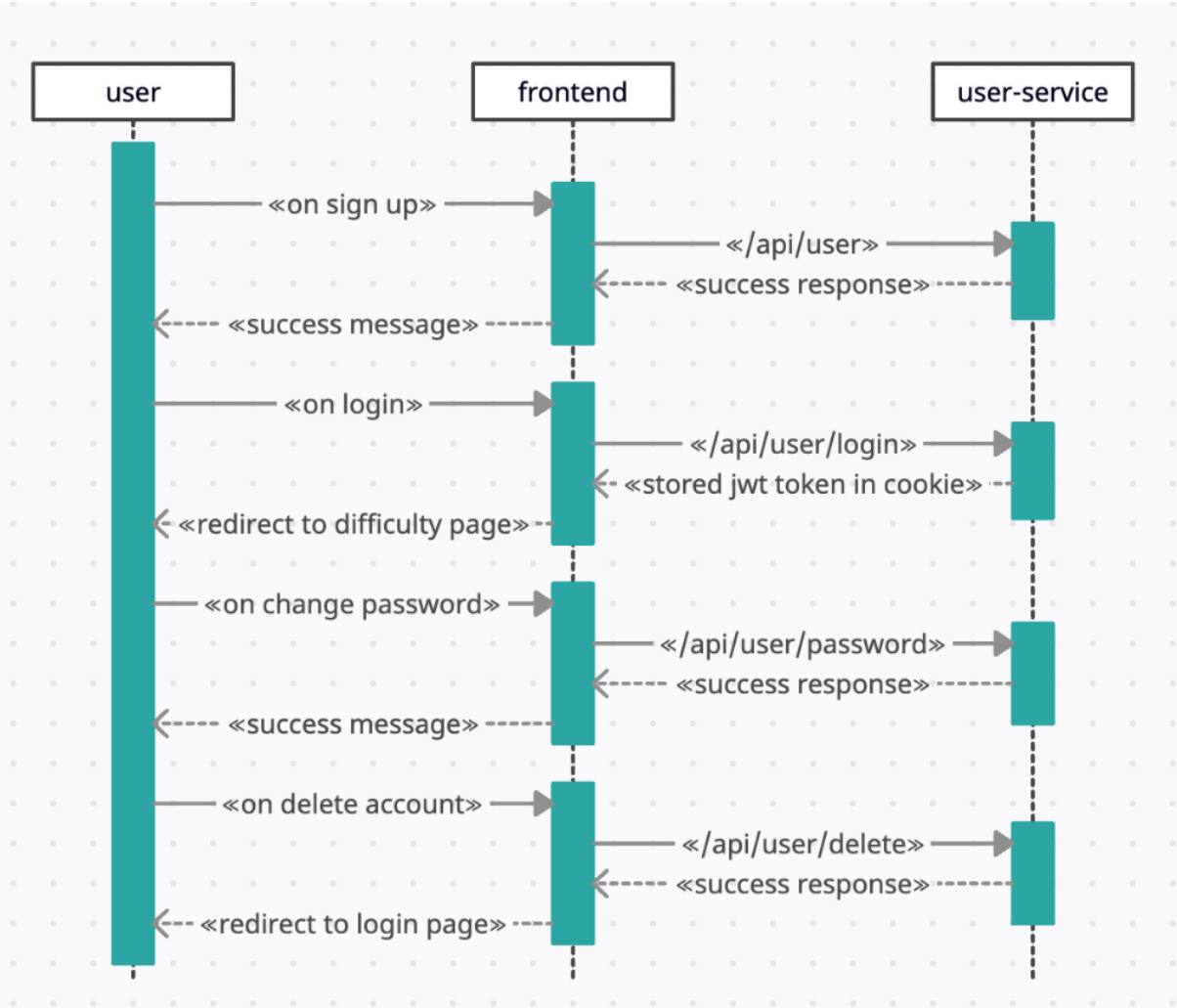


MVC Diagram

4.5.2 User Authentication

We have implemented the user authentication using JWT tokens. When a user logs in, a JWT token containing username is encrypted by signing with a secret. We attach a HttpOnly cookie containing the encrypted JWT token of the user to the response as well. By doing this, we can prevent client-side scripts from accessing the cookie which makes the website less susceptible to cross-site scripting.

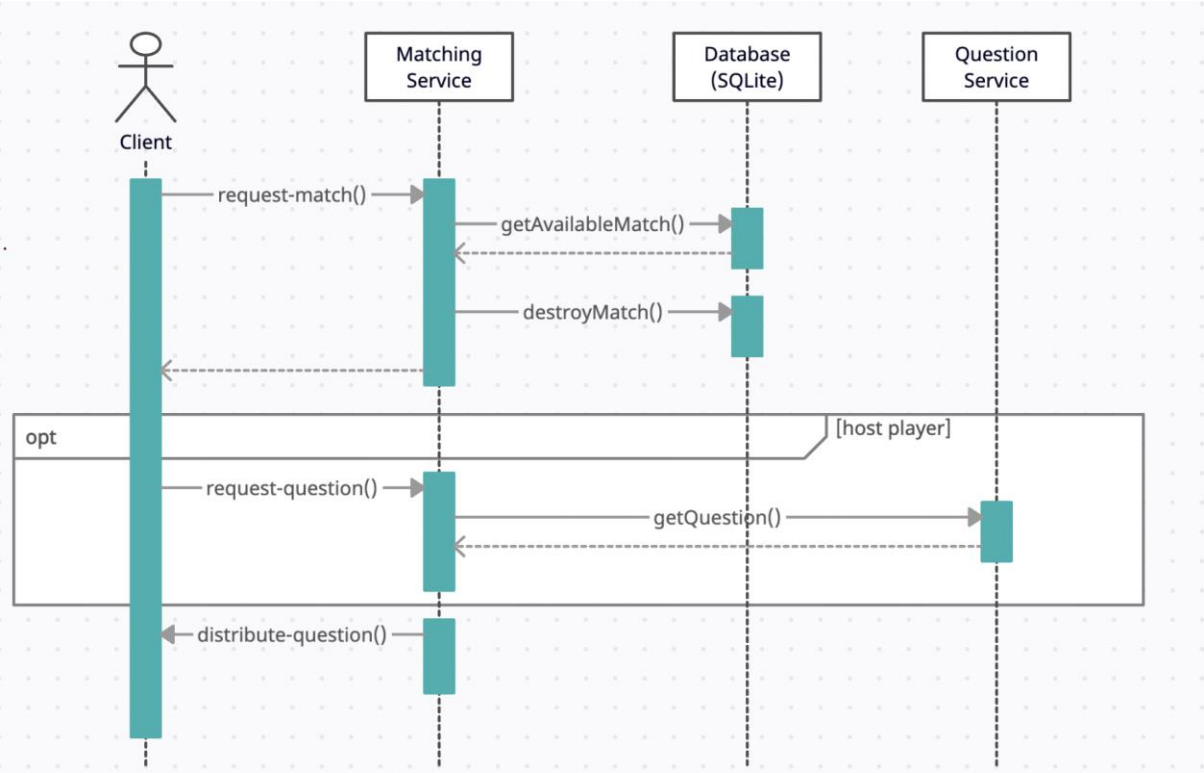
For security reasons, once the user logs out, we blacklist the token to ensure that the user is not able to log in with the same token again. Persistent login is implemented by storing the token of the user in the Zustand store which manages the state.


user
frontend
user-service
«on sign up»
«/api/user»
«success response»
«success message»
«on login»
«/api/user/login»
«stored jwt token in cookie»
«redirect to difficulty page»
«on change password»
«/api/user/password»
«success response»
«success message»
«on delete account»
«/api/user/delete»
«success response»
«redirect to login page»

4.6 Matching Service

4.6.1 Component Interactions

The matching service is responsible for matching up the two distinct users according to their chosen difficulty for the question. After a user submits a matching request, that user's username, along with the difficulty chosen is recorded in the database. Below sequence diagram illustrates the interaction when another user requests a match of the same difficulty as the first (host) user.



After a second user requests a match, then the matching service is going to access the database and looks for whether there is a match request submitted previously. If there is one, then the matching service is going to match the user from the database and another user who just requested the match into the same room. Of course, the match request in the database will have to be deleted to avoid duplicate matching.

After being directed to the room, the host user will request a question to the matching service, reason being that after the matching service retrieves the question from the question service, the matching service needs to distribute the exact same question to the two users in the same room. Hence, inter-service communication is used between the matching and the question service, in the form of synchronous HTTP request.

4.6.2 Pub-sub mechanism using Socket.io

What enables the two users to be in the same room and get the exact same question is Socket.io, a library that supports event-driven communication between the client

and the server, implementing a pub-sub mechanism. Our publisher is the server socket instance, and subscribers are the client socket instances. Two client socket instances are subscribed through the channel named the host user's id, which is also the id of the room.

```
44        const question = await response.json();
45        socket.nsp.to(roomID).emit("distribute-question", question);
```

Like the above code snippet, the server socket instance emits an event through the roomID channel.

```
178       socket.on("distribute-question", (question) => {
179           setQuestionTitle(question.existingQuestion.title);
180           setQuestionDescription(question.existingQuestion.body);
181       })
```

This is then received by the two subscribers, after which the frontend updates the room page by filling in the question title and description for the session. The pub-sub mechanism loosely supports the **Open-closed principle**. Whenever we have a need to support more users in the room (due to a changing business requirement), we can easily extend the system by making additional users join the same room. No internal modifications to the system are required.

4.7 Question Service

The question service is a RESTful API that uses Nest.js for the backend and MongoDB for the database to serve questions for PeerPrep. The questions are classified by difficulty, and each has a unique ID that facilitates the identification and fetching process. As explained in the matching service section, it will provide a random question of chosen difficulty whenever the matching service requests a question to be distributed to the two matched users.

Nest.js was employed for the backend as it helps build lightweight, well-structured, and amazing microservices, it also supports TypeScript which improves reliability and helps reduce bugs in the development process.

MongoDB was employed for the database as it is an open-source NoSQL database. It has a flexible schema-less approach and allows for quick development of the application as lesser time can be spent on configuring the database.

A RESTful API architecture was taken as it takes advantage of HTTP. This means that we do not need to install additional software or libraries when creating a RESTful API.

4.8 Collaboration Service

The Collaboration Service is one of the main features of PeerPrep. We took inspiration from services like Google Docs when we implemented this feature. Since PeerPrep was a service intended to help students prepare for interviews, the Collaboration Service allows them to type and collaborate on any document they desire. The service enables a shared space for the students to share ideas they brainstorm.

The Collaboration Service runs on top of web sockets, like that of the Matching Service. The Socket.io library allows for bi-directional communications via the channels (rooms), which allows changes to be updated quickly and efficiently between the two users.

4.9 Chat Service

The Chat Service allows for quick communication between the two students in PeerPrep. Unlike the Collaboration Service, this service takes up less space in the page and is intended for short communication that should not be in the collaboration document. The service runs on top of web sockets, similar to that of Collaboration Service.

5. Suggestions for improvements and enhancements

5.1 Asynchronous Inter-service communication between services

As explained in the matching service section, there is synchronous inter-service communication between the matching and question service; however, due to the nature of synchronous request/reply, the matching service is blocked until it receives a reply from the question service. This could pose problems to the users because even after a matching is made, the users wouldn't be directed to the room page, hence becoming suspicious of whether a match is made or not.

Asynchronous communication can be used in this scenario, such as using Kafka.js. While requesting a question, the matching service can direct the users to the room page with the collaboration document and chat-box and load up the question instantaneously after.

5.2 Deploy frontend to AWS CloudFront

In addition to AWS S3 Bucket, we can configure to use AWS CloudFront as our deployment point. As AWS CloudFront delivers the static contents through a worldwide network of data centres called edge locations, if our target user is the global audience, this would greatly reduce the latency associated with loading up the website with the best possible performance.

5.3 Prevent URL manipulation in countdown page

As of now, the matching service utilises the difficulty data passed via URL params of the countdown page to match two students. For example, if one were to choose 'Beginner' in the difficulty page, this will be passed to the countdown page's URL as '/countdown?difficulty=beginner'. This poses an issue as users may be able to manipulate the URL to subvert the matching service. For example, even though a user may have chosen a beginner difficulty, the user can manually change the URL to '/countdown?difficulty=advanced'. This would signal to the matching service that the user has chosen the advanced difficulty instead. Hence, an improvement could be made to ensure the integrity of the URL and data passage.

5.4 Implementation of email and forget password for user service

The email address of the user can be included in the sign-up process and this can be taken and stored by the database to be used for future use cases such as a reset password feature where an email receipt is sent to the user's registered email address in order to reset the password for their account in case they forget it.

6. Reflections

Throughout the process of the project, the team certainly has the experience and gained first-hand knowledge of working on a software engineering project that involved many components and various tech stacks to achieve and build the various parts of the platform such as the micro-services as well as a user-friendly front-end.

As a result, we realised that in software engineering, collaboration is essential to the success of the project. As there are so many different technologies that each come with their own complexity, it is neither feasible nor wise for everyone to pick up every technology or tool and divide every task equally. We quickly found that the best way to do things is to identify what are the strengths that each member has, and the unique skills that they bring to the table. With that information, we can better allocate resources and task the expert on the subject matter to tackle those challenges as they will be better equipped to do so.

Another thing that was important was project management. As there was a large amount of work to be done across a myriad of tasks, using a project management tool, in our case Notion, also greatly helped the team to stay focused and in sync. Not only did it serve as a good task and status tracker for the project, it also helped the team to check our progress and ensured that we were tracking well in terms of timeline and adhering to the deadlines of deliverables.

Finally, communication. We set up a recurring meeting on Thursday in the same time slot every week. This greatly assisted the team to plan our work better into sprints and helped everyone get on the same page and up to speed with the developments of each other's contributions. We were able to better adapt on the fly and adjust the development of the project because of the frequent check-ins. During this agreed common timing, questions were raised, and discussions were had on various design and implementation challenges. Constant and clear communication is paramount and critical to the success of any software development team project.