CS3219 Software Engineering Principles

PeerPrep

Team 44

Jung Hyunseok (Peter) (A0220141E)

Zhou Jiahao (A0218561E)

Andy Lam Wei Jie (A0218309E)

Chiun Jia Qian Nadine (A0221134Y)

TABLE OF CONTENTS

1 INTRODUCTION

Technical assessments are part and parcel of the interview process for computer science (CS) students applying for a software engineering internship position. This arduous process only continues as students search for full-time jobs after graduation; it is thus imperative for those students to have a strong understanding of CS fundamentals as well as complex data structures and algorithms. To ensure such capabilities, many students turn to online platforms such as LeetCode and HackerRank to enhance their technical skills by mindlessly taking on a stream of algorithm problems. Understandably, solving countless problems alone quickly gets boring and even daunting, especially when faced with more challenging types of problems.

PeerPrep aims to tackle the above issue by creating an environment in which computer science students would be encouraged to take on challenging technical assessments; this feat is achieved through three core features provided by the platform: (1) **real-time collaboration**, (2) **saved match history**, and (3) **sizeable collection of problems**. These features on top of a clean and responsive user interface (UI) helps students hone their understanding of CS concepts, and at the same time, learn to effectively communicate their approach to problems via collaboration with another student. All in all, PeerPrep hopes to empower students by allowing them to study and prepare for online technical assessments with their peers, hence its name.

PeepPrep is built by a team of students from the National University of Singapore (NUS), as part of CS3219 — Software Engineering Principles and Patterns. The repository for this project is available here.

The following document is intended for those who wish to continue development of PeerPrep or are interested in tackling the above problem themselves, and also serves as a reference for the CS3219 teaching staff.

2 BUSINESS REQUIREMENTS

This section provides a high-level overview of PeerPrep, detailing the project's vision, scope, and stakeholders.

2.1 Vision

Provide a platform with common technical assessment questions for students to practise and prepare for software engineering interviews. Focus on real-time collaboration between users to encourage fruitful discussion of the problem and thus exploring potential solutions together.

2.2 Project Scope

PeerPrep is developed as a web application, allowing users to attempt technical problems on data structures and algorithms via an intuitive and responsive UI. To emulate a realistic technical interview environment in which multiple users can concurrently read and write to a shared codepad, PeerPrep provides features such as real-time collaboration as well as chat and video call capabilities for enhanced communication.

The scope in which this application operates is therefore one that encompasses the target user group of individuals — primarily computer science students — who wish to hone their knowledge of CS concepts to better prepare themselves for technical interviews. The ease of access to the application coupled with its powerful features is likely to be of great value to potential users.

2.3 Stakeholder Analysis

2.3.1 Users

The end consumer, a.k.a. the user of PeerPrep is the most relevant stakeholder. Seeing as the primary need for a user is to readily improve their coding skills and familiarity with technical interviews, the application provides an experience that most closely resembles that of a real-life interview setting.

Each user may also have different preferences when it comes to solving technical challenges, be it pertaining to the difficulty of the problem, or the language they use to tackle it. PeerPrep therefore grants an extra level of freedom to users who want a more personalised experience, with the ability to choose one of three difficulties — i.e. easy, medium, and hard — as well as a specific language of their choice aided with auto-completion and syntax highlighting.

The team also understands that during coding interviews, the interviewer may ask the interviewee to vocalise their thoughts and walk through their approach in tackling a problem; thus, the application will also have a chat and video call capabilities to allow effective communication to take place.

With respect to PeerPrep's UI, the user is likely to be familiar with pre-existing coding platforms in the market such as LeetCode and HackerRank. For this reason, the application adopts an interface comparable to the likes of said coding platforms to provide a more familiar and comfortable environment to work in, thereby improving overall user experience.

2.3.2 Software engineers

Yet another stakeholder group — although relatively indirect than the last — is software engineers who are interested in the problem of facilitating effective technical interview practice. The application's code is therefore available in a public repository and is open to technical improvements proposed by other software engineers; the repository may also serve as a foundation to those who wish to build their own platform from the ground-up.

To ease onboarding development of PeerPrep — or that of its derivatives — the code must be understandable and thus readily maintainable. The team ensures this by strictly following appropriate coding standards and software engineering practices such that the code is more uniform and efficient. Through its microservice architecture, the code is effectively laid out in a modular fashion to maintain a low barrier of entry for new software engineers joining the project.

2.4 User Journey

The following steps illustrate the high-level main success scenario of PeerPrep:

1. A user is interested in honing their technical skills before their upcoming interview and registers a new account on PeerPrep.
2. Thereafter, the user logs into the application through their newly created account.
3. The user selects a difficulty level and proceeds to matchmaking from which a partner user is found.
4. The user (and their partner) are redirected to a collaborative coding space in which they can code together on a shared codepad and communicate directly with each other via chat and video call features.
5. A question of the previously selected difficulty is displayed together with the other components in the collaboration space and the user is able to draft out a solution on the shared codepad with a language of their choice. The codepad also features auto-completion and syntax highlighting which greatly benefit the user.
6. After attempting the problem together, the session is officially concluded upon either the user or their opponent leaving the match.
7. The concluded match is saved in the user's match history, whereby the user can review the problem they have solved in the past.

3 INDIVIDUAL CONTRIBUTIONS

Name	Technical Contributions	Non-technical Contributions
Jung Hyunseok (Peter)	• Implemented Matching MS (Redis, matchmaking logic) • Implemented User MS • Implemented Question MS • Implemented Collaboration MS (chat communication, video call communication, collaborative coding) • Designed frontend and implemented UI	• Report • Presentation
Zhou Jiahao	• Implemented Matching MS (matchmaking logic, History MS) • Implemented Question MS • Integrated Matching MS with Question MS and Collaboration MS	• Report • Presentation
Andy Lam Wei Jie	• Implemented Question MS • Implemented Collaboration MS (chat communication) • Implemented Matching MS	• Report • Presentation
Chiun Jia Qian Nadine	• Implemented Question MS • Implemented Collaboration MS (collaborative coding) • Implemented User MS	• Report • Presentation

4 REQUIREMENTS SPECIFICATION

The high-level requirements of PeerPrep are as follows:

- Allow a user to match with another user who has chosen the same difficulty
- Provide a question of the selected difficulty at random from a sizeable question bank
- Enable meaningful real-time collaboration and communication between two users

4.1 Functional Requirements

Below are the functional requirements (FRs) of PeerPrep categorised by each microservice that the application adopts. Each section is accompanied by a description of the feature domain along with a list of concrete FRs that are fulfilled by the application.

4.1.1 User MS

The User MS provides features pertinent to the creation and maintenance of a user account in PeerPrep. The service allows a user to acquire a new account and gain access to user-specific features that are locked behind route-level authentication middleware.

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

4.1.2 Matching MS

The Matching MS matches users based on the difficulty they choose. Users are pooled into one of three groups — separated by difficulty — from which they are paired and forwarded to a collaboration space in which they can tackle a randomly generated technical problem together.

S/N	Functional requirement	Priority
FR2.1	The system should allow the user to select the difficulty level of the questions they wish to attempt.	High
FR2.2	The system should be able to match two waiting users with the same difficulty levels and put them in the same room.	High
FR2.3	If there is a valid match, the system should match the users within 30 seconds.	High
FR2.4	The system should inform the users that no match is available if a match cannot be found within 30 seconds.	High
FR2.5	The system should provide a means for the user to leave a room once matched.	High

4.1.3 Question MS

The Question MS generates and returns a technical assessment problem of a chosen difficulty as indexed with a query parameter. The service queries its database and retrieves a random question that contains relevant information, including, but not limited to, question ID, similar topics, difficulty, and the question content.

S/N	Functional requirement	Priority
FR3.1	The system should have a question bank that contains a list of technical assessment questions.	High
FR3.2	The system should be able to issue a question of the correct specified difficulty level.	High
FR3.3	The system should store the questions based on their difficulty level.	High
FR3.4	The system should randomly generate a question of a specified difficulty level.	Medium

4.1.4 Collaboration MS

The Collaboration MS — which internally hosts the Communication MS as illustrated in Figure 5.1.a — enables the mechanism for real-time collaboration between participants in a room, on top of other useful features such as chat and video call. The service provides real-time collaborative editing laced with a conflict-free replicated data type (CRDT) structure for the shared document to resolve edit conflicts. Sending and receiving of messages as well as live video calls are enabled by WebSockets and WebRTC, respectively.

S/N	Functional requirement	Priority
FR4.1	The system must allow multiple users to interact on a code editor or equivalent, displaying the updates made by each user on both ends.	High
FR4.2	The system should allow up to two users into the collaboration room.	High
FR4.3	The system must help users to reconnect back to the collaboration room upon unexpected disconnect.	Medium
FR4.4	The system should allow video chat between users in the same session.	Low
FR4.5	The system should support auto-completion and syntax highlighting.	Low

4.2 Non-Functional Requirements

Below are the non-functional requirements (NFRs) of PeerPrep categorised under each type of NFR that is to be met in the design and implementation of PeerPrep.

4.2.1 Constraints

S/N	Description	Priority
NFR1.1	PeerPrep must be compatible with the majority of modern browsers, namely: Chrome, FireFox, and Microsoft Edge.	High
NFR1.2	PeerPrep must be responsive and thus accessible on devices of varying screenports, namely: desktop monitor, laptop, tablet, and mobile phone.	Medium
NFR1.3	PeerPrep must not use more than 1 GB of memory from the browser.	Medium
NFR1.4	PeerPrep can only be accessed via a stable internet connection.	High
NFR1.5	PeerPrep must implement conflict-free solutions for collaborative features; namely, the shared document between two users must account for concurrent edits.	High

4.2.2 List of quality attributes

4.2.2.1 Performance

S/N	Description
NFR2.1	The first contentful paint of PeerPrep's frontend must occur in one second or less.
NFR2.2	Chat messages must be sent and received between two users in one second or less.
NFR2.3	Video calls must be stable and have less than three seconds of delay in both video and audio feedback.
NFR2.4	PeerPrep should match users who have chosen the same difficulty in less than 0.5 seconds if there is a possible match.
NFR2.5	The system should be able to support at least 50 pairs of concurrent users in their respective collaboration spaces

4.2.2.2 Security

S/N	Description
NFR3.1	Users' passwords should be hashed and salted before storing in the database to guard against malicious attacks.
NFR3.2	The questions as stored in the Question MS database must be sanitised before being fetched by the client as to prevent cross-site scripting (XSS) attacks.
NFR3.3	Routes for microservices should be protected with authentication middleware to ensure that only users with the right level of access can query user-specific endpoints.
NFR3.4	Frontend input forms should sanitise the data before forwarding to a microservice as to prevent potential attacks, including, but not limited to, SQL/NoSQL injections and XSS attacks.
NFR3.5	PeerPrep must blacklist access tokens of users who have logged out or deleted their accounts to prevent attackers from accessing user-specific features via said users' access tokens.

4.2.2.3 Usability

S/N	Description
NFR4.1	A first-time user must be able to understand how to register and log into an account without explicit onboarding
NFR4.2	A user must be able to navigate the matchmaking menu and initiate the matchmaking process without explicit onboarding
NFR4.3	A user must be able to traverse down the list of previous matches and review their progress without explicit onboarding
NFR4.4	Visual feedback via a modal or toast must be provided upon each user-initiated action
NFR4.5	A user must be able to readily navigate to the account settings page and make necessary account-related changes without explicit onboarding

4.2.2.4 Correctness

S/N	Description
NFR5.1	PeerPrep must display correct user-specific information, namely, match history and account settings, to the current user.
NFR5.2	Upon successful matchmaking, the system must direct the user to a collaboration space containing a technical assessment problem with the correct chosen difficulty.
NFR5.3	The collaboration space should facilitate correct communication by exchanging chat messages as well as video and audio feedback to and from the correct partner.

4.2.2.5 Robustness

S/N	Description
NFR6.1	All invalid requests to microservices should be handled gracefully without incurring a server crash.
NFR6.2	An abrupt disconnect from a user in the collaboration space should be handled properly by prompting the partner user.

NFR6.3	Unsuccessful interactions between microservices and their respective databases should be handled gracefully and must notify the user.

4.2.2.6 Reliability

S/N	Description
NFR7.1	Requests and responses between (1) the frontend and a microservice and (2) a microservice and its database must be exchanged consistently with a 95% chance of success.
NFR7.2	The frontend must attempt to reconnect an abruptly disconnected user to their respective collaboration space within five seconds.
NFR7.3	Upon failure, a microservice must reliably be restored to its original state within 30 minutes.

4.2.3 Quality attributes prioritisation

Attribute	Score	Performance	Security	Usability	Correctness	Robustness	Reliability
Performance	3		<	<	^	^	<
Security	5			^	^	<	<
Usability	2				^	^	^
Correctness	5					^	<
Robustness	4						^
Reliability	4						

With reference to the above quality attributes prioritisation matrix, it is evident that, concerning the development of PeerPrep, there are a number of qualities that must be fulfilled by the application, namely, performance, correctness, and robustness.

The prioritisation of performance effectively stipulates that the application can run smoothly, efficiently, and quickly. In the context of PeerPrep's development, the team must ensure that the application can be experienced by the user with little to no delay, whilst performing potentially taxing operations such as concurrent editing and video call communication. To further emphasise the feeling of strong and reliable performance, the application must display helpful feedback to the user whenever and wherever possible; this would allow the user to experience an immediate cause and effect of their actions and bolster the idea that the application is highly performant.

The team's focus on correctness entails an application that is not only fast and efficient, but performs operations predictably and correctly. This would ensure that the right information is displayed to a user at the right time, i.e. the application "works as expected". For instance, directing a user to a collaboration space that contains a technical problem that is not of the difficulty that the user had selected would negatively affect the user's experience. To prevent such circumstances, correctness must actively be accounted for throughout development to guarantee a predictable and thus positive user experience.

Robustness is yet another quality that should not be overlooked during development of PeerPrep, as it promotes graceful handling of unexpected scenarios that may occur during the lifetime of an application. Should unexpected network issues or invalid user inputs potentially cause failure in one of the application's components, the failure should be handled gracefully without significantly impacting the user experience. As PeerPrep adopts socket programming in which network issues could cause detrimental damage to the user experience, the team must take great care in ensuring that potential errors are minimally caught and handled gracefully, without causing collateral damage to other components of the system.

4.2.4 System requirements

- PeerPrep should work on mainstream modern browsers such as Google Chrome, Safari, Edge.
- PeerPrep should be compatible on major device viewports such as that of desktop PCs, laptops, tablets, and mobile phones; i.e. the application should be responsive.

4.2.5 External interface requirements

Quoting Richard Thayer (2002), “External interface requirements specify hardware, software, or database elements with which a system or component must interface.” [4] The sections to follow detail the requirements that must be met to ensure effective interfacing and communication between a system and external components.

4.2.5.1 User interfaces

PeerPrep must adopt a familiar and thus approachable UI to ease onboarding the application’s various features. Though there is no specific style guide on which the UI is based, a high-fidelity prototype has been created to emulate a user’s experience on the platform and to guide development of the actual UI.

To support usage on multiple devices, screen layout and resolution constraints are also taken into account; the application will need to provide a responsive layout that changes the position, sizing, and general look of its on-screen components depending on the size of the device viewport. This would encourage more users to board the application as a positive user experience can be maintained on most if not all compatible devices.

To accommodate users who may be visually impaired, the application will adopt accessibility standards such as the Accessible Rich Internet Applications (ARIA) specification to aid use with screen readers; as the standard suggests (2022), it allows websites to be “more accessible to people with disabilities” [1]. With enhanced accessibility, the visually impaired would be able to experience all of PeerPrep’s features with little to no difficulty.

4.2.5.2 Hardware interfaces

PeerPrep will need to be able to run on most modern computing devices, including fully-fledged desktop PCs, laptops, tablets, and mobile phones. As PeerPrep is a web application, the web libraries and frameworks that are used for the application must be carefully considered to ensure proper rendering and usage on the browser.

The video call communication feature of the application effectively necessitates that the device has a front-facing camera. Though this feature is integral to the ideal PeerPrep experience, the application should still take into account the case where a camera feed may not be active or available, making appropriate adjustments to the feature as needed.

4.2.5.3 Software interfaces

Between PeerPrep and the OS on which the application runs, yet another level of interfacing is required; the application is developed for modern versions of most operating systems (OSes), namely MacOS, Windows, Android, and iOS. By supporting usage on major OSes, PeerPrep will be accessible to a wider group of potential users.

Database interfacing will also need to be considered during development of PeerPrep, as the microservices of the application access their respective databases to retrieve and store data. For example, to save and retrieve the match history of a user, previous matches would need to be recorded in a database — be it NoSQL or SQL — to support persistence of data between different sessions.

4.2.5.4 Communication interfaces

A number of internet protocols must be adopted to enable different types of communication between the application and the microservices; interfacing with the browser, HTTP/S to securely retrieve HTML data for the frontend, as well as the WebSocket and WebRTC protocols to enable real-time collaborative editing and communication between two clients.

4.2.6 Data requirements

The crux of the data requirements lie in services that employ their own database, for example, the Question MS. A question object must minimally contain useful information that can be formatted and displayed to the end user, including, but not limited to, the difficulty, title, content, and similar topics.

As more questions may be added into the Question MS database as the application grows bigger, extending the number of keys of a question object may not be viable as the change must be applied to all objects that exist in the database. Therefore, a small yet meaningful schema must be defined first to mitigate the need to extend the schema.

5 DESIGN AND IMPLEMENTATION

This section details PeerPrep's technical implementation as well as software design patterns that are adopted in the application.

5.1 Architectural Design

The microservice architecture design pattern was primarily adopted to compartmentalise and segregate different services, ultimately giving way for potential horizontal scaling in the future. This separation of concerns allows for ready implementation of additional services due to loose inter-service coupling, mitigating the need for developers to integrate new services with existing ones. That is, each service can be treated in a modular fashion that builds up the individual functionalities of a whole system.

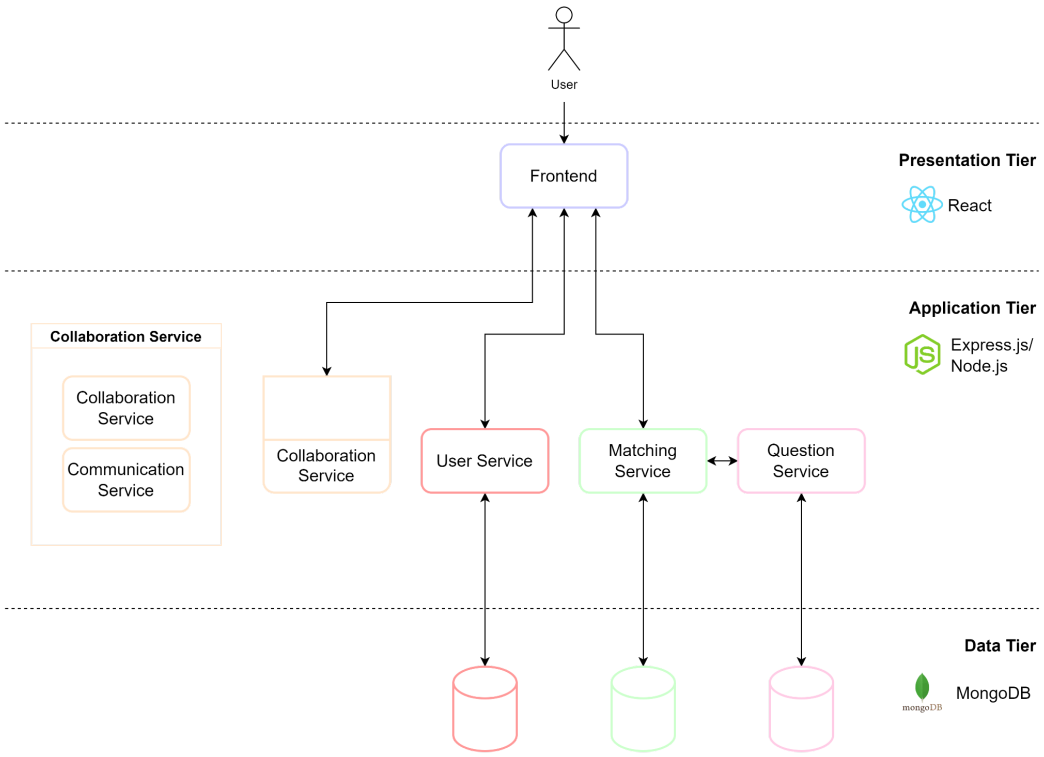

Figure 5.1.a: PeerPrep Microservice Architecture

The services may communicate with clients — be they the frontend or even other microservices — through an API that designates endpoints for different queries and/or operations. For example, in the context of PeerPrep, the Question MS communicates directly with the Matching MS as the role of the former is to supply a randomised question to a user in the collaboration space, which is efficiently achieved by directly querying the latter. Each MS

also encompasses its own database separate from that of others to prevent large volumes of data transfer between multiple services and a single shared database, further isolating the services from one another.

As for how the microservice architecture work with respect to user interaction, (1) the frontend receives a user event through its controller, (2) the frontend forwards this event to an endpoint of a microservice from which said microservice runs a specified operation, and (3) if needed, the MS queries its database to retrieve persistent data for use with the operation.

A brief summary of the benefits of adopting a microservice architecture for PeerPrep is as follows:

- **Faster development**
  The separation between services allows the team to split up and concurrently develop different services. The division of work enables one or two members of the team to work on each individual service, giving rise to faster development. In cases where a service cannot operate properly without another service — e.g. Matching MS and Question MS — a stub can be used during development to simulate the functionality of a service.

- **Scalability**
  In a microservice architecture, the loose coupling of services allows the team to readily add or remove services without significant impact to the usability of other services and thus the overall product. This also implies that maintaining and updating the features of existing services is also more efficient due to abstraction of concerns between individual services of the system.

- **Bug traceability**
  Though a more technical benefit, the microservice architecture makes it easy to trace and squash bugs as they are encountered. When a bug is found during development, the developer may easily trace down the root cause by identifying from which service the bug occurred, making necessary updates to the code thereon.

5.2 Technology Stack

The team resorted to four core technologies — MongoDB, ExpressJS, ReactJS, NodeJS (MERN) — for PeerPrep's frontend and backend implementation. Below is a brief description and rationale for employing the aforementioned technologies in PeerPrep's development with respect to the system's microservice architecture as illustrated in Figure 5.1.a.

5.2.1 Presentation tier

React is used in the frontend web server to render web page content that is to be displayed to the end user. React is also responsible for handling user events and modifying the application state, triggering updates to the view accordingly.

5.2.2 Application tier

NodeJS/ExpressJS act as the bridge of communication between the client tier and the database tier, allowing the client to query persistent data from storage. With strong community support and access to a myriad of third-party packages, NodeJS/ExpressJS enables fast full-stack application development.

5.2.3 Database tier

MongoDB is the database of choice for PeerPrep, and acts as a document store that persists data such as user account information, match history, or the question bank between different user sessions. Compared with SQL databases, including, but not limited to, PostgreSQL, MySQL, and MSSQL, MongoDB's inherent NoSQL nature allows for more flexible schemas without key constraints and potential for horizontal scaling via sharding.

5.3 Design Patterns

5.3.1 Database per Service Pattern

The Database per Service Pattern is a direct consequence of adopting the microservice architecture. The pattern entails the segregation of databases across multiple microservices, with each individual service accompanied by its own database. In the context of PeerPrep, each service executes commands and queries on their respective database, e.g. the User MS queries its database of user accounts, and the Matching MS queries its database of previous user matches. This separation allows the services to be more loosely coupled such that they can be developed and maintained more efficiently without affecting other services; that is, the system is rendered more scalable.

Although this pattern does have a number of benefits, it also comes with its drawbacks. For one, managing multiple databases across different services can become a challenge as the services grow in number. The system is scalable in that it is easy to add, change, or remove services, but this also comes at the price of a proportional increase in the number of databases to maintain. Should the team at some point decide to convert the services' database from MongoDB to a SQL alternative, performing SQL operations could be quite difficult with the Database per service Pattern as the tables are spread across different services.

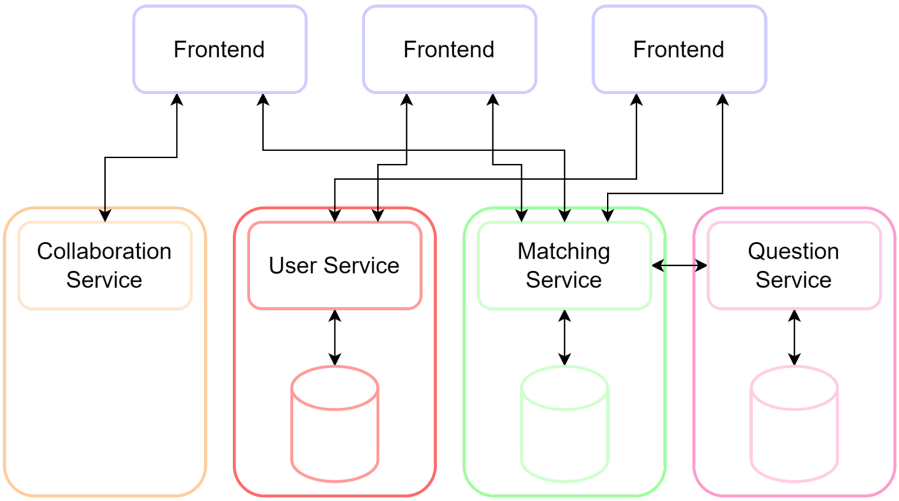

Figure 5.3.1.a: Database per Service Pattern

5.3.2 Repository Pattern & Chain of Responsibility Pattern

The Repository Pattern employs an abstraction of repositories that, according to Microsoft (2022), are defined as “classes or components that encapsulate the logic required to access data sources.” [3] That is, a repository (1) abstracts away common data access functionality and (2) centralises the handling of domain objects, consequently decoupling business logic with the underlying persistence technology.

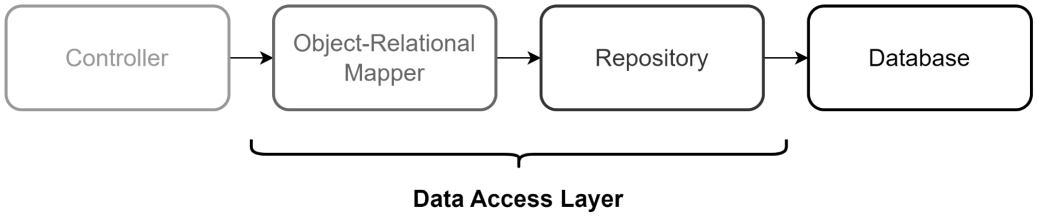

Figure 5.3.2.a: Repository Pattern

PeerPrep adopts the repository pattern in its microservices to abstract away the logic in fetching documents from the services’ respective database, effectively establishing a data access layer through which business logic can be translated to data access operations.

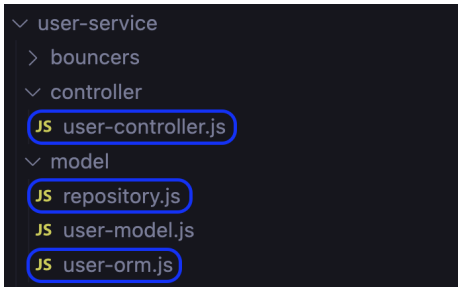

Figure 5.3.2.b: PeerPrep Repository Pattern file structure

The pattern is realised via a “chain of responsibility” between three major components: the controller, Object-Relational Mapper (ORM), and repository; each component works to transform the business logic to a data access operation.

5.3.3 Pub-Sub Pattern

The Pub-Sub pattern is dominantly present in microservices that handle functionalities related to real-time communication, namely, the Matching and Collaboration MSes, as the services necessitate the use of WebSockets that give rise to the live broadcasting of updates between multiple connected clients — in the context of the pub-sub pattern, subscribers. When a client conversely forwards a message to the microservice, the roles are effectively reversed and the clients and services behave as publishers and subscribers, respectively.

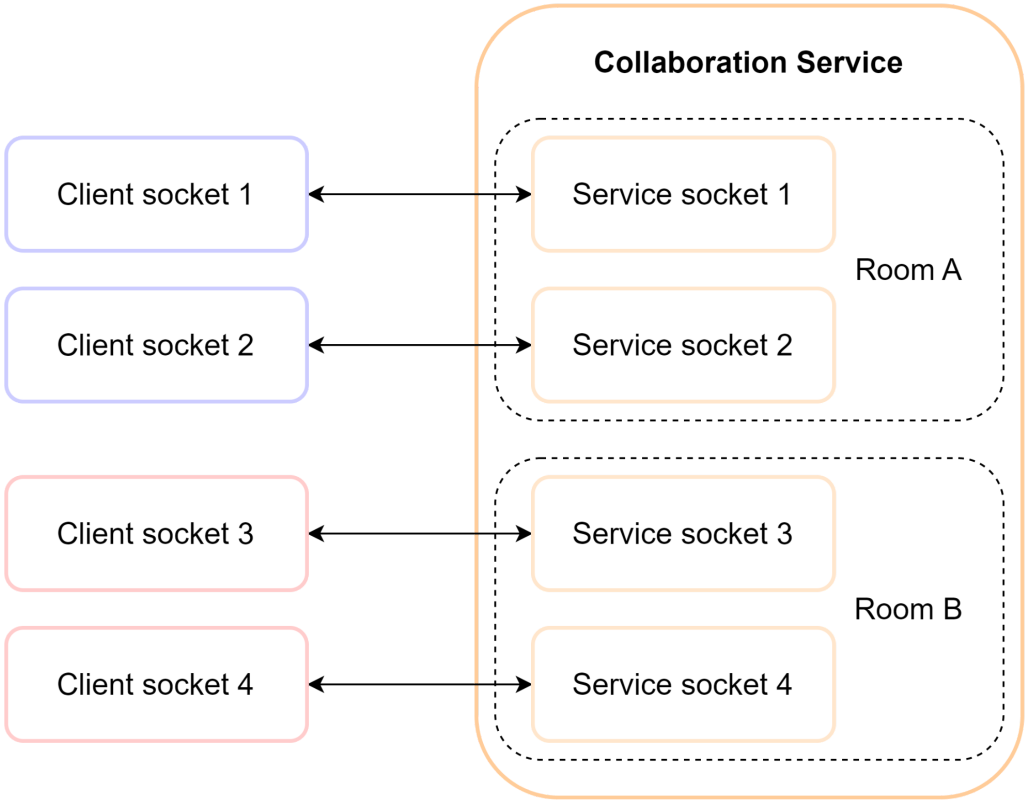

Figure 5.3.3.a: PeerPrep Collaboration MS Pub-Sub Pattern

On a more technical level, PeerPrep makes use of Socket.io that facilitates real-time, bi-directional communication between WebSockets. In particular, the Collaboration MS employs Socket.io rooms to selectively broadcast messages to clients, further concretising the notion of the Pub-Sub Pattern. The following table details the roles of the client and service during the execution of a specified WebSocket event:

Event	**Role of client**	**Role of service**
The client socket fires an event that is received in real-time by the service. e.g. the client edits the code in the code editor.	Publisher	Subscriber
The service socket fires an event that is received in real-time by the client. e.g. the service forwards code changes that occurred from the edit.	Subscriber	Publisher

5.3.4 Adapter Pattern

The Adapter Pattern introduces the adapter entity that allows otherwise incompatible interfaces to communicate and collaborate with each other. An adapter essentially acts as a middleman that sits between two interfaces, transforming operations from one interface to the other to join the functionalities of two independent interfaces.

The Adapter Pattern is used extensively in microservices that access its database, namely the User, Matching, and Question MSes. Considering that the services run on the Node.JS runtime environment, and that MongoDB is adopted as the services' database technology, there exist no means for the service to directly perform data access operations without some sort of translation between commands executed in Node.JS and objects in MongoDB.

This is where Mongoose plays the role of the adapter to provide compatibility between the interfaces, allowing for query building and schema validation in MongoDB directly from the Node.JS runtime environment. This compatibility effectively adds another layer of abstraction between the business logic and the data layer as all data access operations are abstracted away within the adapter which effectively operates as a translator between the two interfaces.

```
const MatchHistorySchema = new Schema({
    matchId: {
        type: String,
        required: true,
        unique: true,
    },
    playerOneUsername: {
        type: String,
        required: true,
    },
    playerTwoUsername: {
        type: String,
        required: true,
    },
    date: {
        type: Date,
        default: Date.now,
    },
    question: {
        type: QuestionSchema,
        required: true,
    }
});
```

Figure 5.3.4.a: PeerPrep Adapter Pattern file structure

```
export async function findMatchHistory(params: {
    username: string;
}) {
    const result = MatchModel.find({
        $or: [
            {playerOneUsername: params.username},
            {playerTwoUsername: params.username}
        ]
    })
    return result;
}
```

Figure 5.3.4.b: PeerPrep Adapter Pattern schema validation

5.4 Microservices

5.4.1 User MS

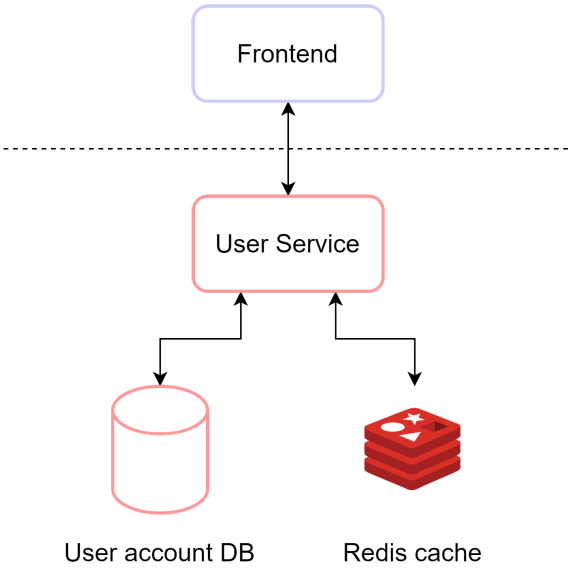

Figure 5.4.1.a: PeerPrep User MS

The User MS handles user related events such as account creation, account deletion, user login and logout, as well as user password update. Furthermore, it is also responsible for user authentication by providing and verifying access tokens for features that require authentication.

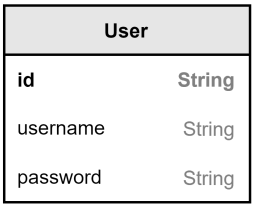

Figure 5.4.1.b: PeerPrep User MS schema

To facilitate persistent storage of users in MongoDB, PeerPrep adopts the above user model. The user model consists of three key elements: ID, username, and (encrypted) password. Note that although the ID itself serves as a unique identifier for users, the username is used instead throughout the system as they are also unique in PeerPrep. The implication of this is that should the team decide to implement functionality for users to change their username, this

would not be possible until the system is modified to check for user uniqueness using the user's ID and not the username.

Technologies used

Technology	Purpose
Redis	Caching of blacklisted JWT access tokens upon user logout or account deletion.
JSON Web Token (JWT)	Generation and verification of JWT access tokens to guard against features that require authentication.
MongoDB	Persistent storage of basic user information as illustrated in Figure 5.4.1.b.
bcrypt	Encryption and subsequent decryption of user passwords.

5.4.1.1 Signup

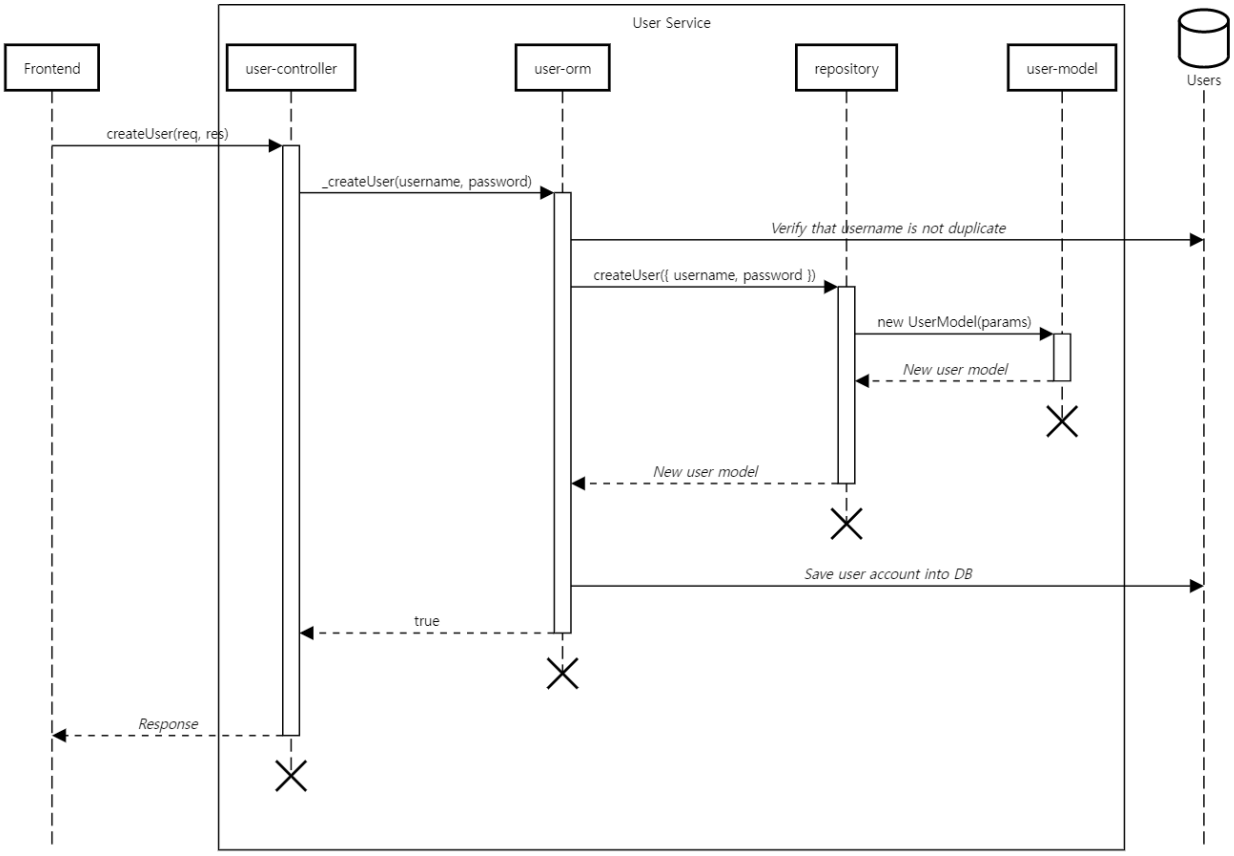

Figure 5.4.1.1.a: PeerPrep User MS signup sequence

To create a new account, the user will first initiate the event from the frontend. The repository pattern takes place in which the command for user creation is modified by each component in the chain until it reaches the data layer. Upon successful creation of a new account, the account information will be saved in the User MS database, and a success response will be returned to the frontend.

With reference to the above sequence diagram, before `_createUser` is called on user-orm by the user-controller, the User MS will first confirm that the username is not already taken. If it has been taken by another user, the User Service will return an error response; else, `_createUser` will be successfully executed and a successful response returned to the user thereafter, assuming no internal server errors occur.

Note:

- For security purposes, PeerPrep employs the bcrypt library to encrypt user passwords before submitting their information. The encryption occurs at the user-model where the user password is hashed before creating a new instance of a user model.

5.4.1.2 Login

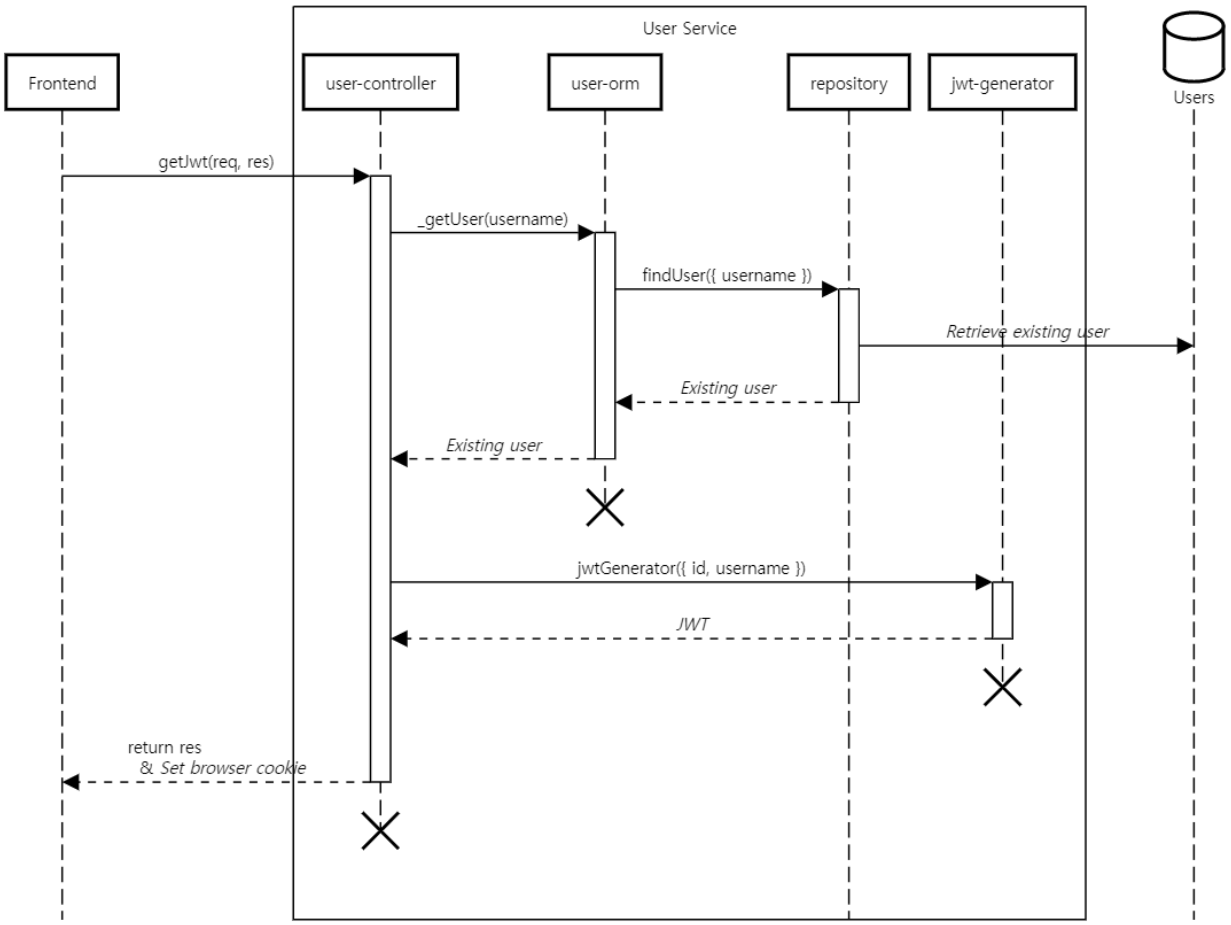

Figure 5.4.1.2.a: PeerPrep User MS login sequence

Upon user login, if the username and password matches one in the User MS database, the user-controller will generate a JWT token that serves as an access token for all actions that require authentication. The token will be set as a cookie in the user's browser and retrieved for verification thereon.

With reference to the above diagram, the jwt-generator returns a JWT token that contains encoded information about the user's ID and username. Although PeerPrep at its current state does not provide any features that require authorisation, should the team decide to implement a new role and subsequently role-specific actions, the JWT token may be encoded differently to encompass information about the user's role as well.

Note:

- The bcrypt library is used during this sequence of events to decrypt the encrypted password from user account creation.

5.4.1.3 Update password

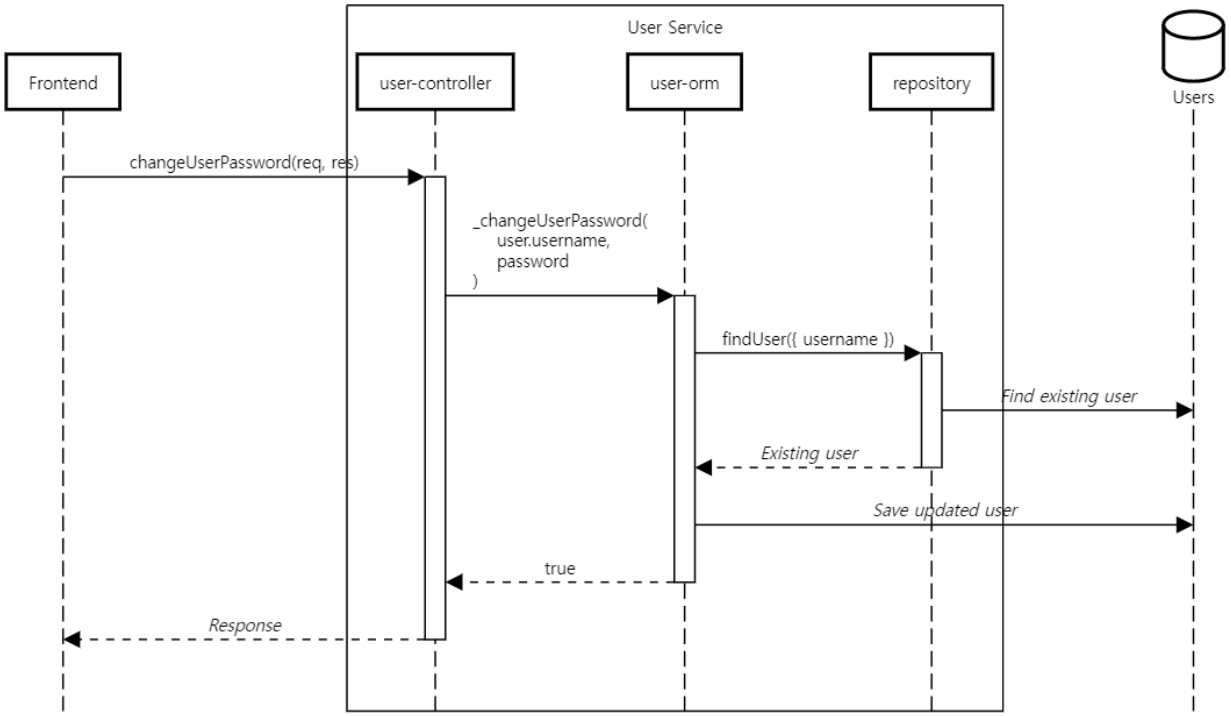

Figure 5.4.1.3.a: PeerPrep User MS update password sequence

To update the user password, the User MS first finds the account details of the user that triggered the event using their username. Once an account is successfully found, the User MS updates the password details and saves the new password back into the database. A successful execution of a user password update is accompanied by the service returning a success response to the user.

Note:

- Before the password update event is initiated, a middleware that sits between the endpoint and the password update function fetches the user's JWT token from the request, only proceeding with the password update if the token is valid.
- Similar to account creation, to ensure security, PeerPrep employs the bcrypt library to encrypt user passwords before submitting their information. The encryption occurs at the user-model where the user password is hashed before returning the update instance of a user model.

5.4.1.4 Logout

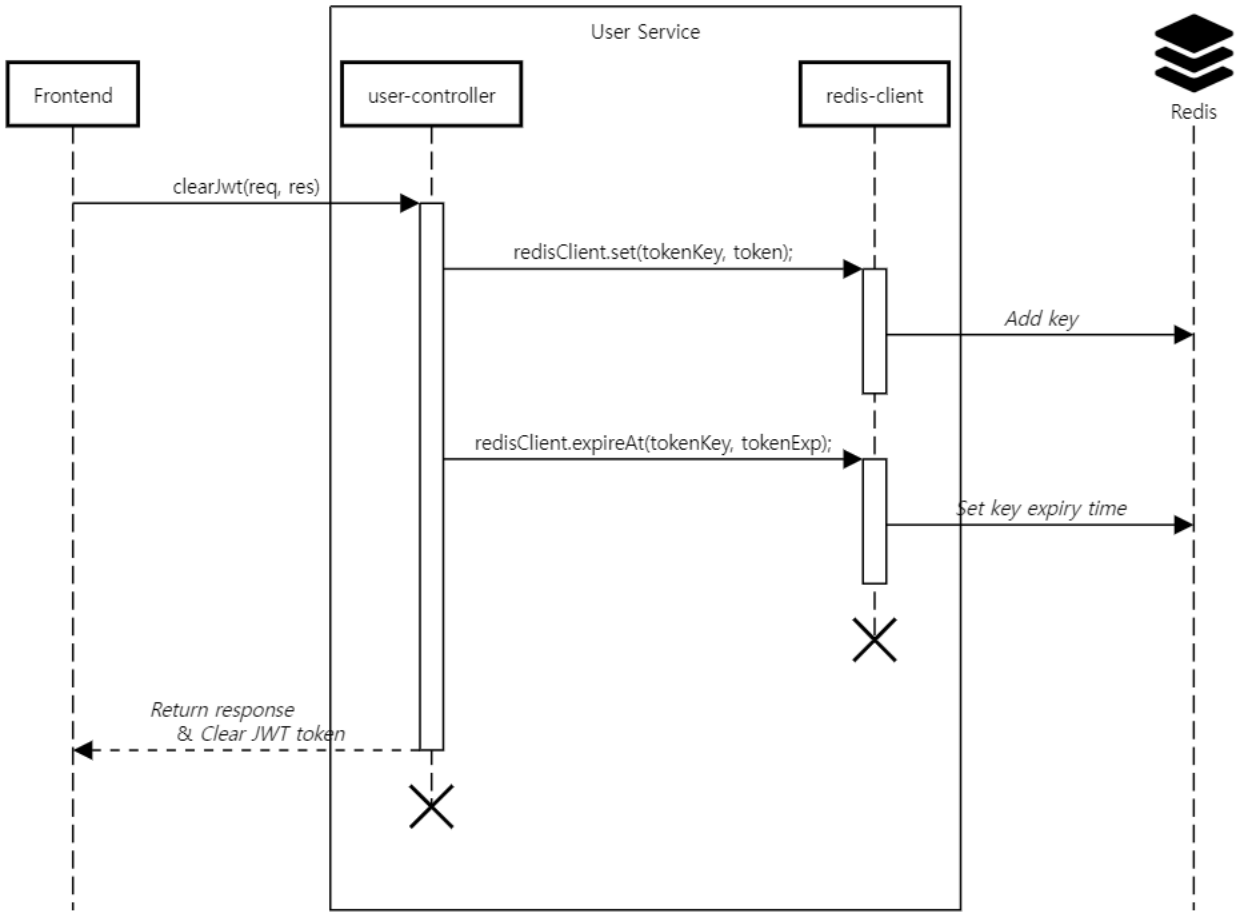

Figure 5.4.1.4.a: PeerPrep User MS logout sequence

To execute user account logout, a method call to the user-controller invoked the initiation of the logout event. The user's JWT token as retrieved from the request is blacklisted in the service's Redis cache so as to prevent a malicious attacker from reusing the token to access any features that require user authentication. After blacklisting the token, the controller returns a success response to the frontend from which the token cookie is removed from the browser at the same time.

Note:

- Before the user logout event is initiated, a middleware that sits between the endpoint and the password update function fetches the user's JWT token from the request, only proceeding with the logout if the token is valid. The sequence of events taken for authentication is detailed in Section 5.4.1.6.

5.4.1.5 Delete account

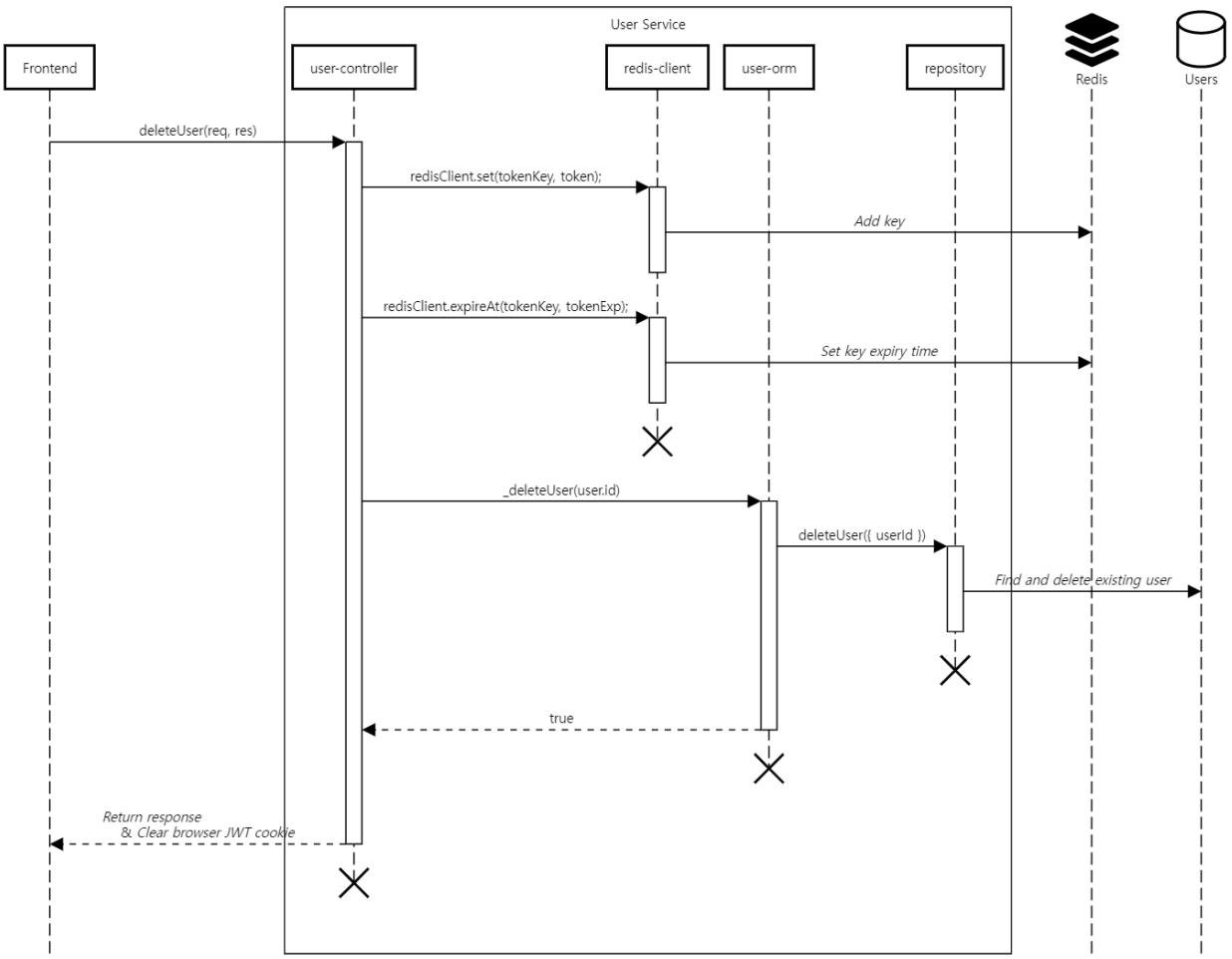

Figure 5.4.1.5.a: PeerPrep User MS delete account sequence

When account deletion is initiated, the JWT token is blacklisted in Redis cache as with the logout event. Successful token blacklisting leads to removal of user information from the User Service database and a success response being returned to the user.

Note:

- Before the account deletion event is initiated, a middleware that sits between the endpoint and the password update function fetches the user's JWT token from the request, only proceeding with the deletion if the token is valid. The sequence of events taken for authentication is detailed in Section 5.4.1.6.

5.4.1.6 Authentication

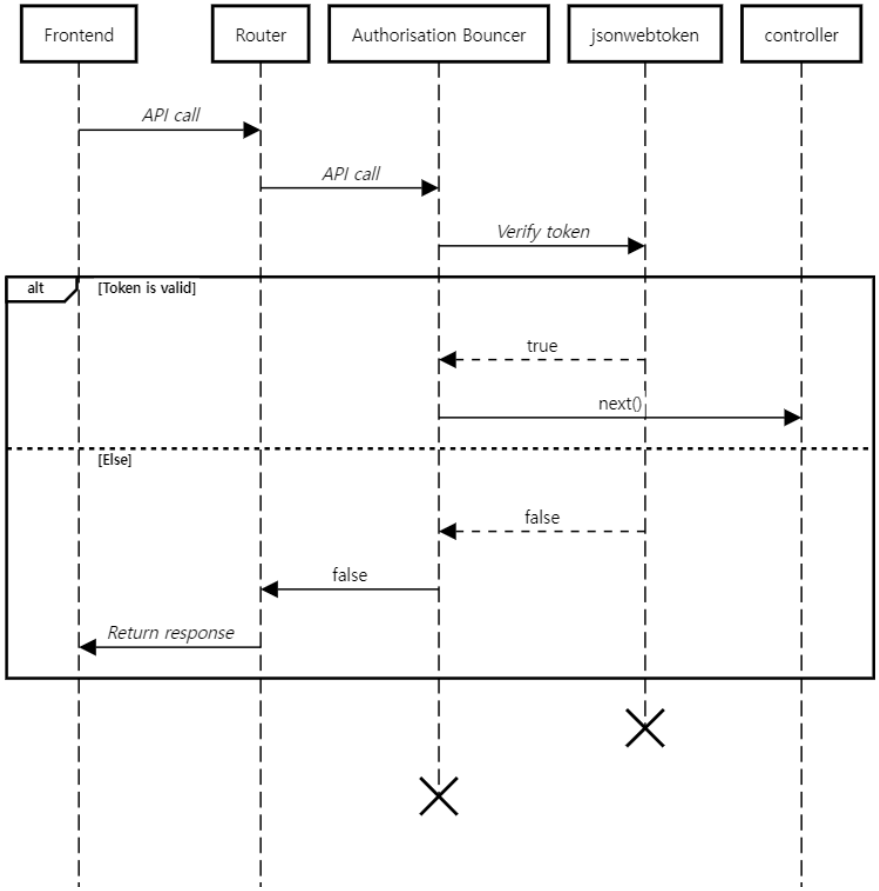

Figure 5.4.1.6.a: PeerPrep User MS authentication sequence

For actions that require user authentication, namely account password change, account logout, and account deletion, authentication middleware is put in place to verify user authentication status. This is implemented via a “bouncer” approach, in which the middleware acts as a bouncer that prevents the execution of certain events if the user is not authenticated. The middleware checks that the token is indeed valid before forwarding the parameters or inputs of the requests to the following controller.

5.4.2 Matching MS

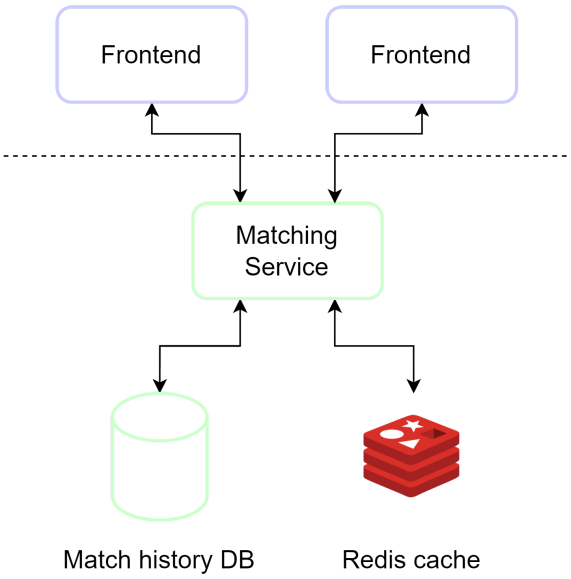

Figure 5.4.2.a: PeerPrep Matching MS

The Matching Service primarily handles the matchmaking of two users as well as persistent storage of previous user matches, i.e. match history.

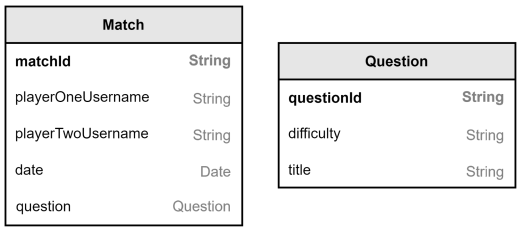

Figure 5.4.2.b: PeerPrep Matching MS schema

To facilitate persistent storage of users in MongoDB, PeerPrep adopts the above match and question models.

Technologies used

Technology	Purpose
Redis	Pool users based on the difficulty chosen, returning matches upon successful matchmaking.
MongoDB	Persistent storage of match history information as illustrated in Figure 5.4.2.b.

Note:

- Redis cache was chosen over MongoDB for storage of pending matches as Redis is an in-memory database and is thus extremely fast and appropriate for matchmaking.

5.4.2.1 Find or create pending match

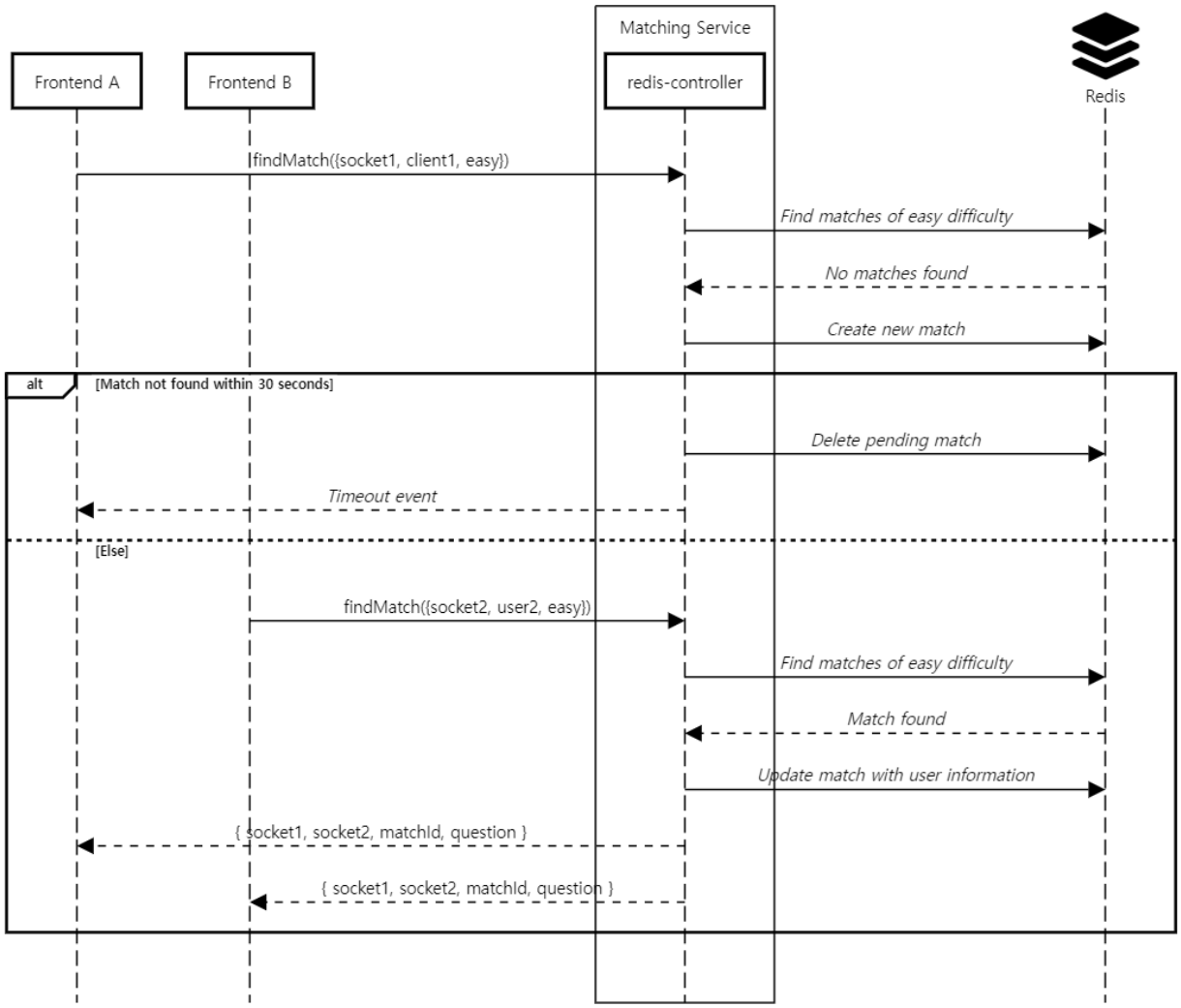

Figure 5.4.2.1.a: PeerPrep Matching MS find or create pending match sequence

The sequence through which a successful match is returned relies on two users concurrently searching for a match using WebSockets. Assume that the first user finds no valid match found in the match pool; the redis-controller proceeds to create a new entry in Redis and waits for incoming match requests. At this point, another user may initiate the matchmaking process in the same pool as that of the first user; the redis-controller finds a pending match and pushes the second user into said match. Since the implementation is WebSocket-based, the service (publisher) simply notifies both users (subscribers) of a successful match from which the frontend directs them to a shared collaboration space; the connotation of publisher-subscriber based approach is discussed in more detail in Section 5.3.3.

5.4.2.2 Find active match

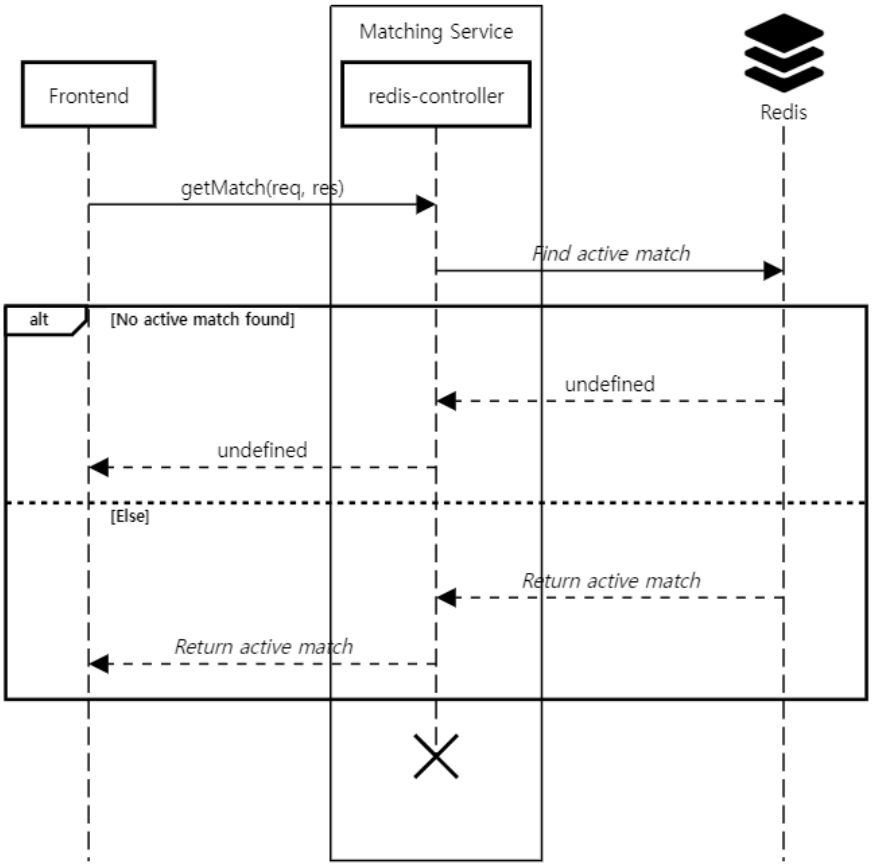

Figure 5.4.2.2.a: PeerPrep Matching MS find active match sequence

To retrieve an active match as created after successful matchmaking, the Matching MS simply queries the Redis cache to find any active matches containing the user. If no active match is found, an empty value is returned. Otherwise, the active match is returned and the frontend redirects the user to the collaboration space.

5.4.2.3 Create match history

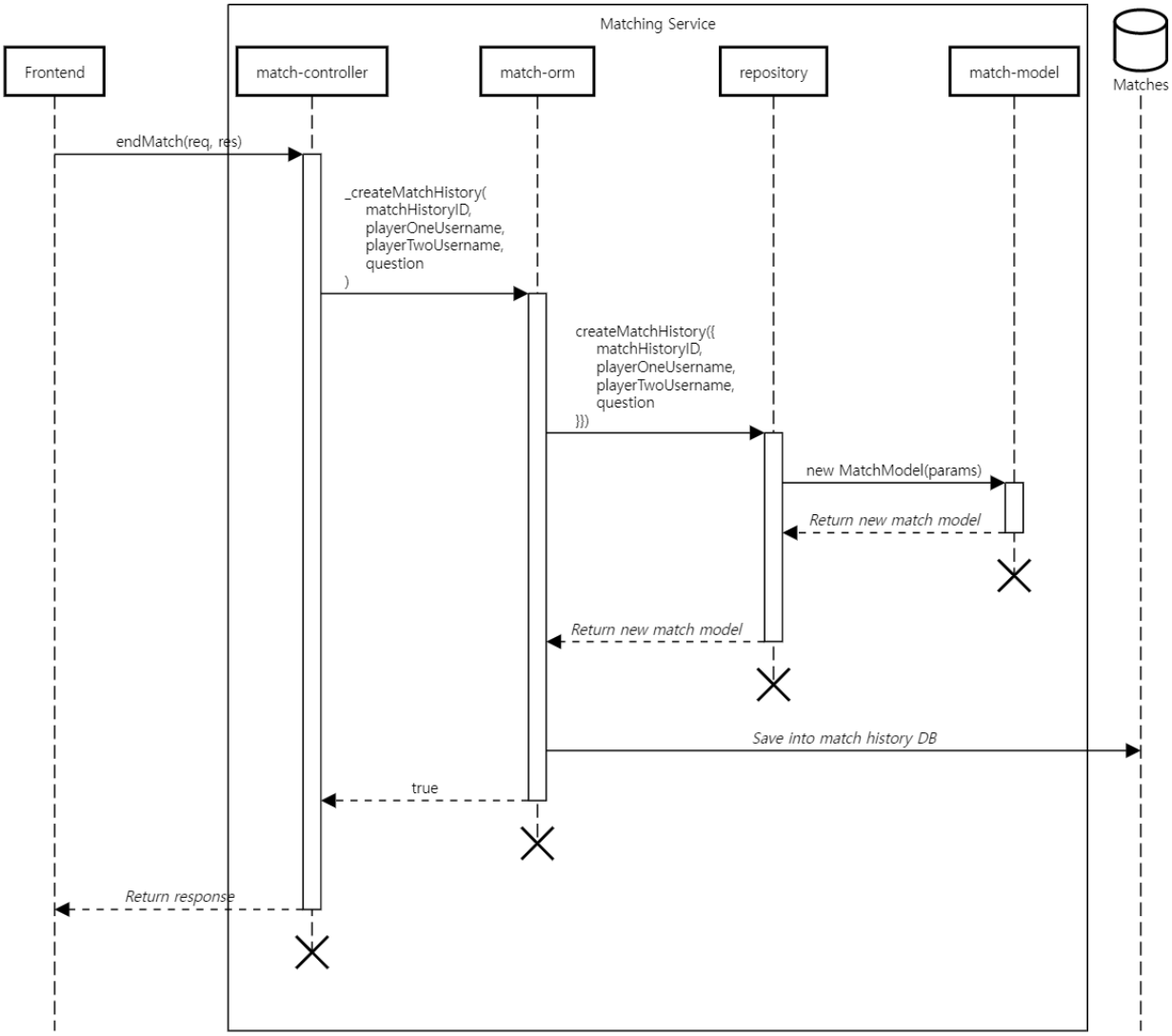

Figure 5.4.2.3.a: PeerPrep Matching MS create match history sequence

Similar to the account creation feature of the User MS, the Matching MS adopts the Repository Pattern to abstract away the creation of a match history object; the Repository pattern is discussed in more detail in Section 5.3.2. Note here that the creation is only triggered when the user ends the current match in which they are participating.

5.4.2.4 Retrieve match history

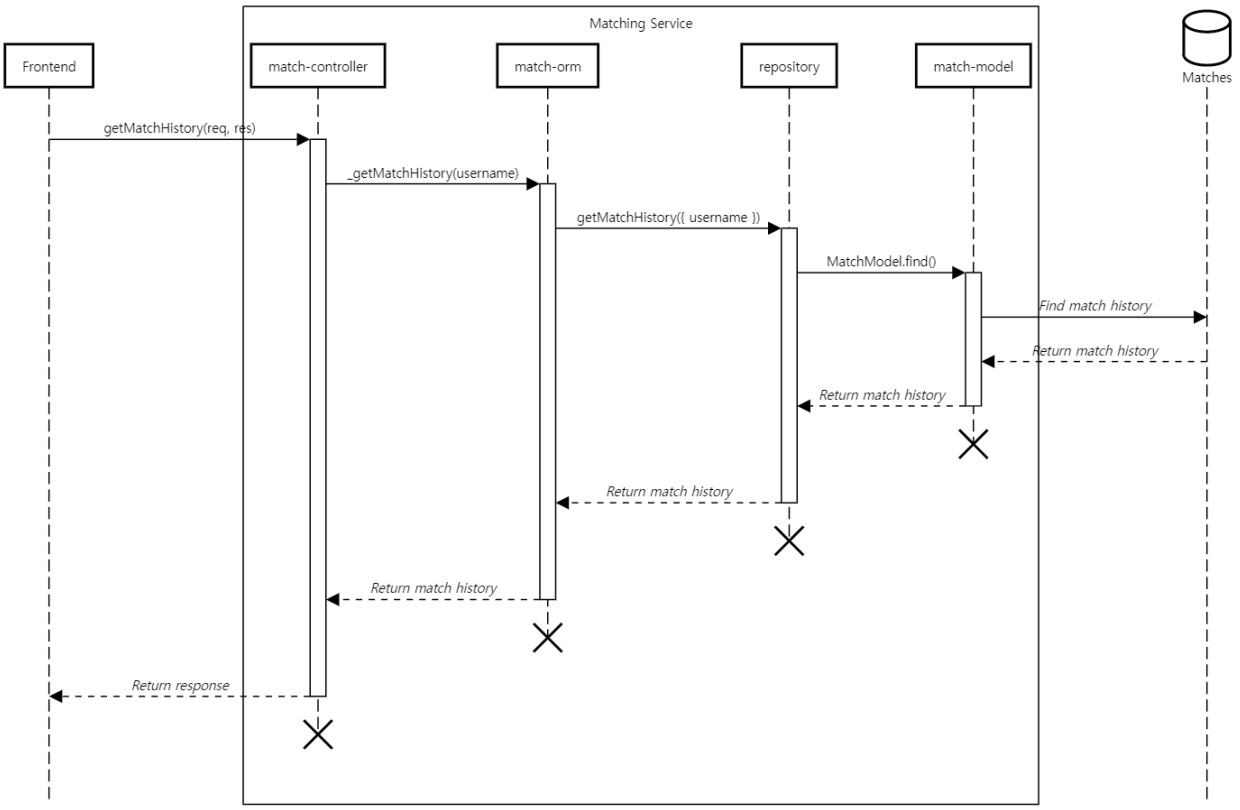

Figure 5.4.2.4.a: PeerPrep Matching MS retrieve match history sequence

A list of match history can be retrieved upon request from the frontend by recursively executing commands on Repository Pattern components — as illustrated in Figure 5.3.2.a — until data is retrieved from the database. An array of match history is returned to the user from which it is styled and displayed to the user in the dashboard page.

5.4.3 Question MS

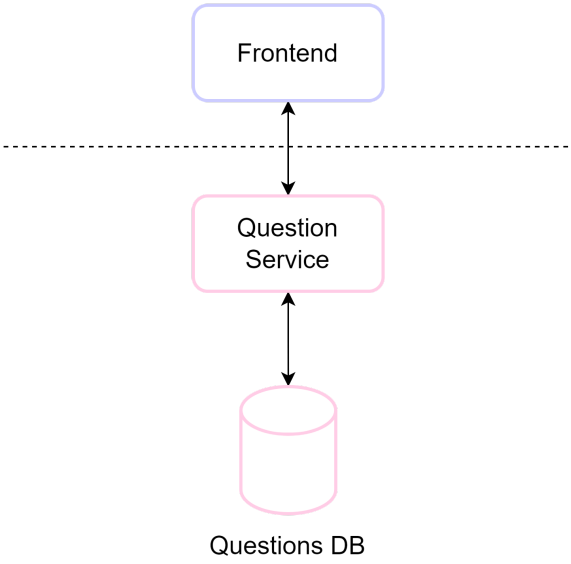

Figure 5.4.3.a: PeerPrep Question MS

The Question MS is rather simplistic, listening for requests at just one endpoint for generating a random question of a specified difficulty.

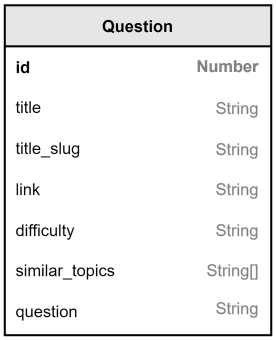

Figure 5.4.3.b: PeerPrep Question MS schema

Observe that the Question schema above is different from that of the Matching MS as illustrated in Figure 5.4.2.b, as the former here is designed to contain the entire question whilst the latter is used as an intermediary for the Match schema.

Technologies used

Technology	Purpose
MongoDB	Persistent storage of questions.

5.4.3.1 Retrieve random question

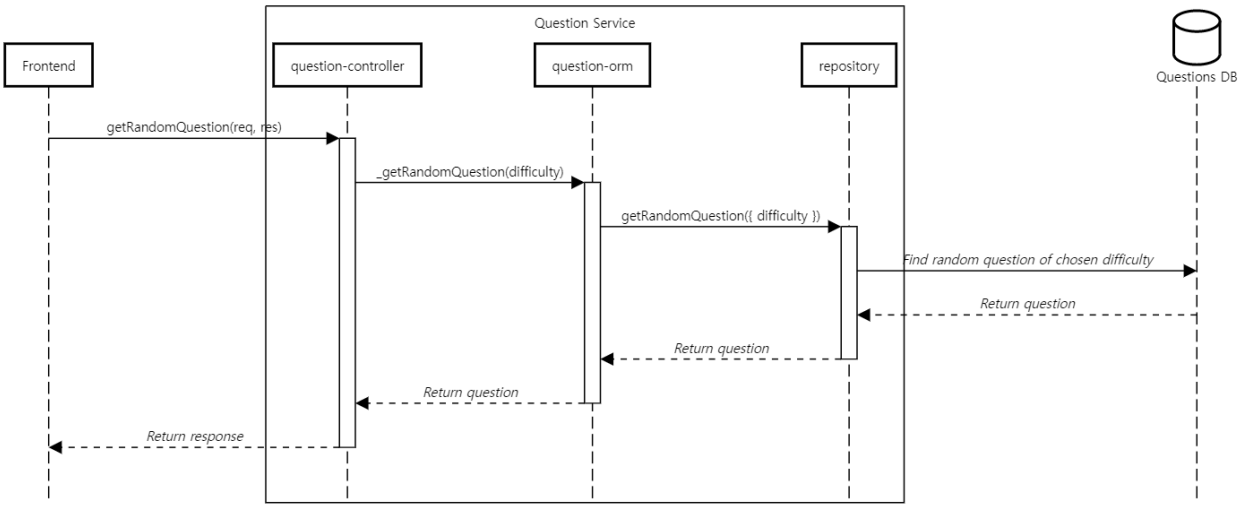

Figure 5.4.3.1.a: PeerPrep Question MS retrieve random question sequence

To retrieve a random question of a specified difficulty after successful matchmaking, the frontend initiates the process and invokes the service to retrieve a random question from the database. This is achieved — similarly to other services that query into their database — by passing control from the controller, to the ORM, to the repository to incrementally convert the operation to one that would be executable within the data layer. To specify the difficulty of the question that is to be retrieved as well as to randomise the selection, the question is retrieved using MongoDB's $match and $sample aggregations.

5.4.4 Collaboration MS

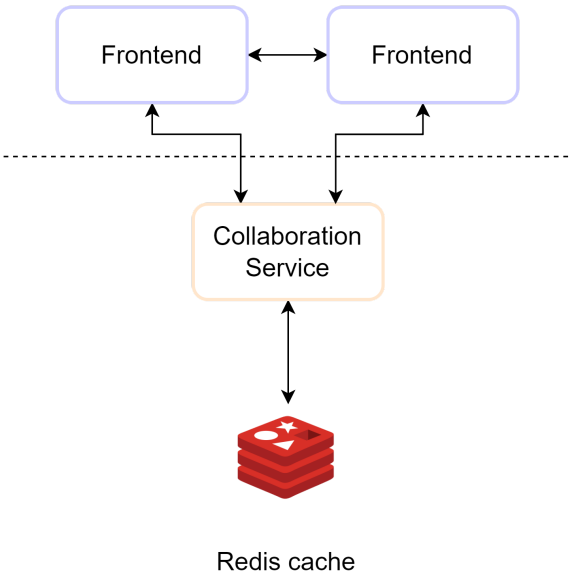

Figure 5.4.4.a: PeerPrep Collaboration MS

When a successful match is found via the Matching MS, the two users that were involved in the matchmaking will be directed to a shared collaboration space, in which they will be able to collaborate and communicate with each other through the Collaboration MS. Though initially designed as two separate services (Collaboration MS and Communication MS), the two services were later merged together during development to facilitate more maintainable development as having two services meant that accounting for double the number of WebSocket connections. The Collaboration MS employs a number of in-memory data structures, namely maps, to establish the relationship between the user sockets and the match in which they are engaging; the mappings are as follows:

From	To	Purpose
Match ID	AutoMerge document	Maintain the shared document on which the users in a given match are collaborating.
Match ID	Chat array	Maintain the chat history between two users in a match.
Match ID	Socket IDs	Maintain a list of socket IDs to which the service can publish messages to.
Socket ID	Match ID	Identify which match the user is taking part in.

Technologies used

Technology	Purpose
Socket.io	Real-time, bi-directional connection between the frontend and service sockets.
Automerge	Handle overwrite conflicts and allow for real-time collaborative editing via a CRDT.
simple-peer	Facilitate peer-to-peer (P2P) connection between two users via WebRTC for video call communication.

5.4.4.1 Collaboration space setup

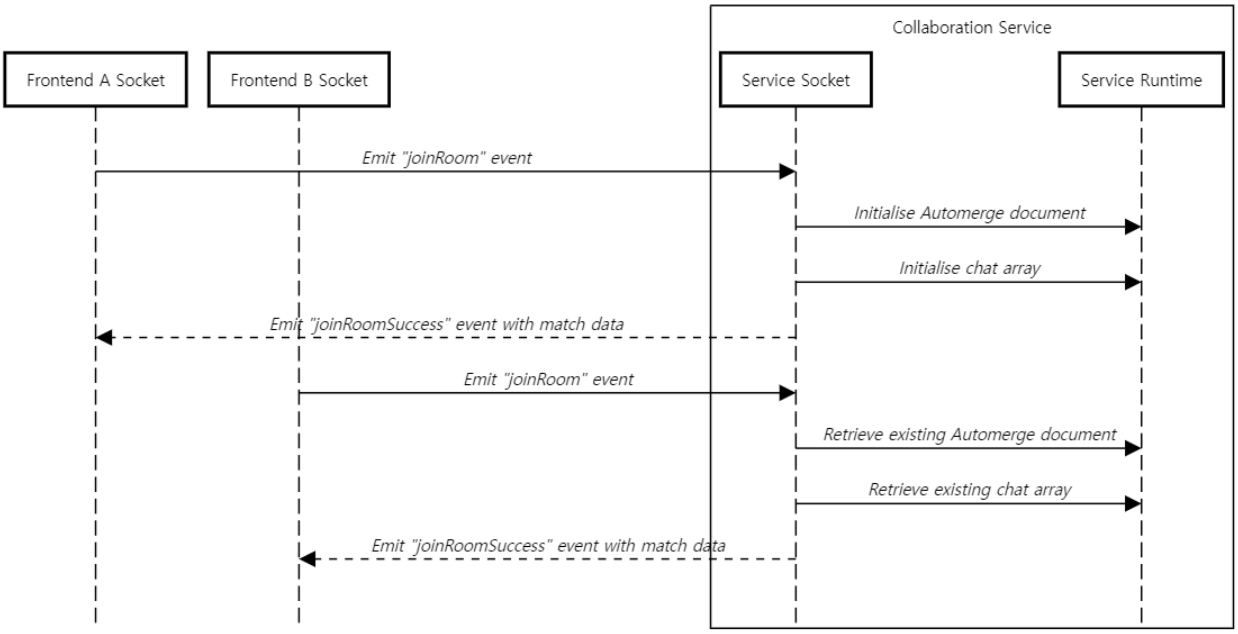

Figure 5.4.4.1.a: PeerPrep Collaboration MS collaboration space setup sequence

When the first user enters the collaboration space, the client socket emits a joinRoom event to the service socket. Upon successful join, the service performs the following key operations to prepare the room for collaboration:

First user

1. Push the client socket into a room with the match ID.
2. Initialise a new Automerge text document.
3. Create an empty chat array.
4. Emit a joinRoomSuccess event to the client socket with a payload that contains the document and chat array.

Second user

1. Push the client socket into a room with the match ID.
2. Retrieve an existing Automerge text document, existing chat array, and partner ID.
3. Emit a joinRoomSuccess event to the client socket with a payload that contains the document, chat array, and partner ID.

The storage and retrieval of match information is carried out with the in-memory maps as tabulated in Section 5.4.4 and thus occurs quickly. The second user retrieves an additional partner ID that is used to set up the video call communication. As for how the first user establishes the communication is elaborated in Section 5.4.4.2.

5.4.4.2 Video call communication

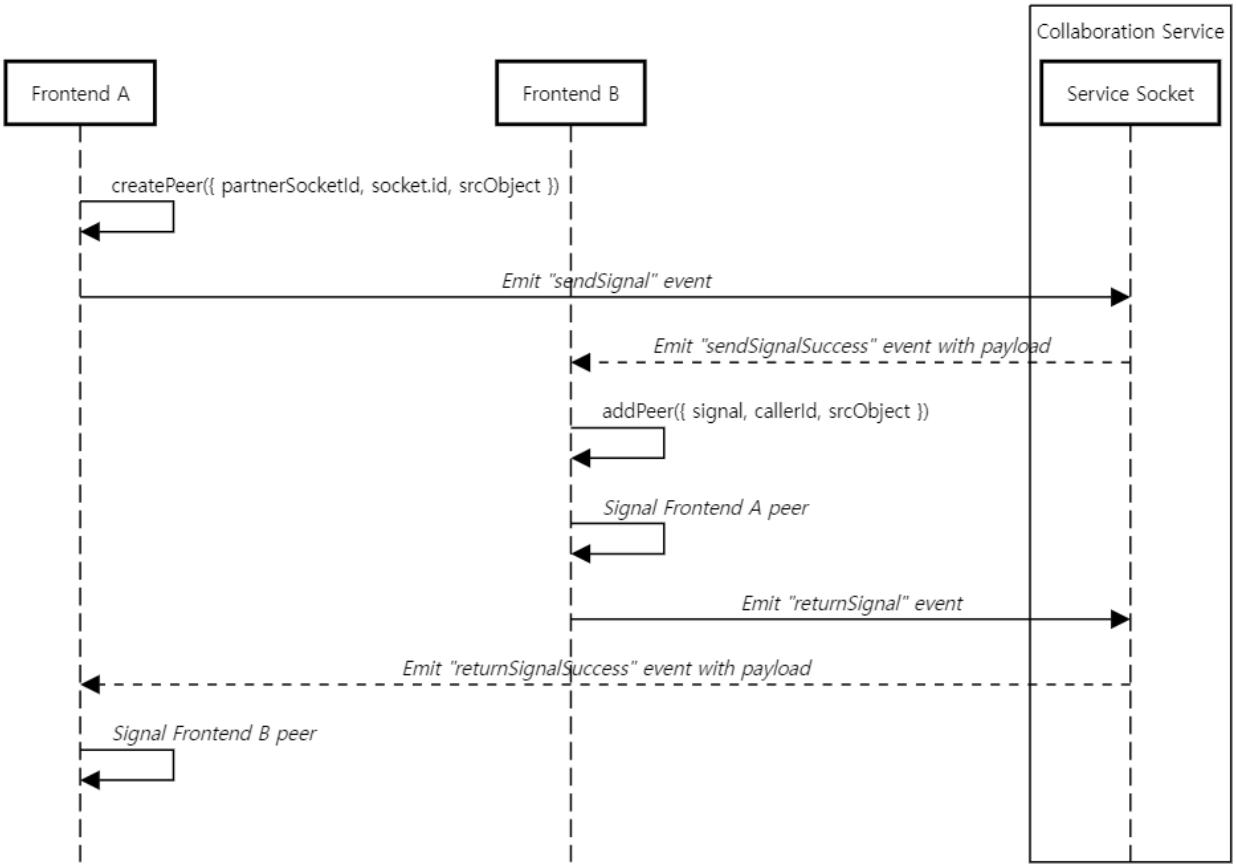

Figure 5.4.4.2.a: PeerPrep Collaboration MS video call communication sequence

The diagram above illustrates the sequence of events that takes place to establish video call communication between two users. Although the second user has access to the socket ID of the first user and can thus initiate its video feed from thereon, the first user does not have any information about the destination to which their video feed will be directed. Below is a high-level step-by-step walkthrough of the above diagram that describes the sequence through which a P2P connection is established between two users.

1. Second user (1) emits “sendSignal” event to the service upon successful connection, with payload that contains second user’s video stream and first user’s socket ID, and (2) creates peer for P2P connection

2. Service receives “sendSignal” event and emits “sendSignalSuccess” event to first user using user’s socket ID, with payload that contains second user’s video stream and socket ID

3. First user (1) receives “sendSignalSuccess” event and emits “returnSignal” event to the service with payload that contains first user’s video stream and second user’s socket ID, (2) creates peer for P2P connection, and (3) establishes connection with second user’s peer using second user’s video stream

4. Service receives “returnSignal” event and emits “returnSignalSuccess” event to the second user using user’s socket ID, with payload that contains first user’s video stream

5. Second user receives “returnSignalSuccess” event and establishes P2P connection with first user’s peer using first user’s video stream

5.4.4.3 Chat communication

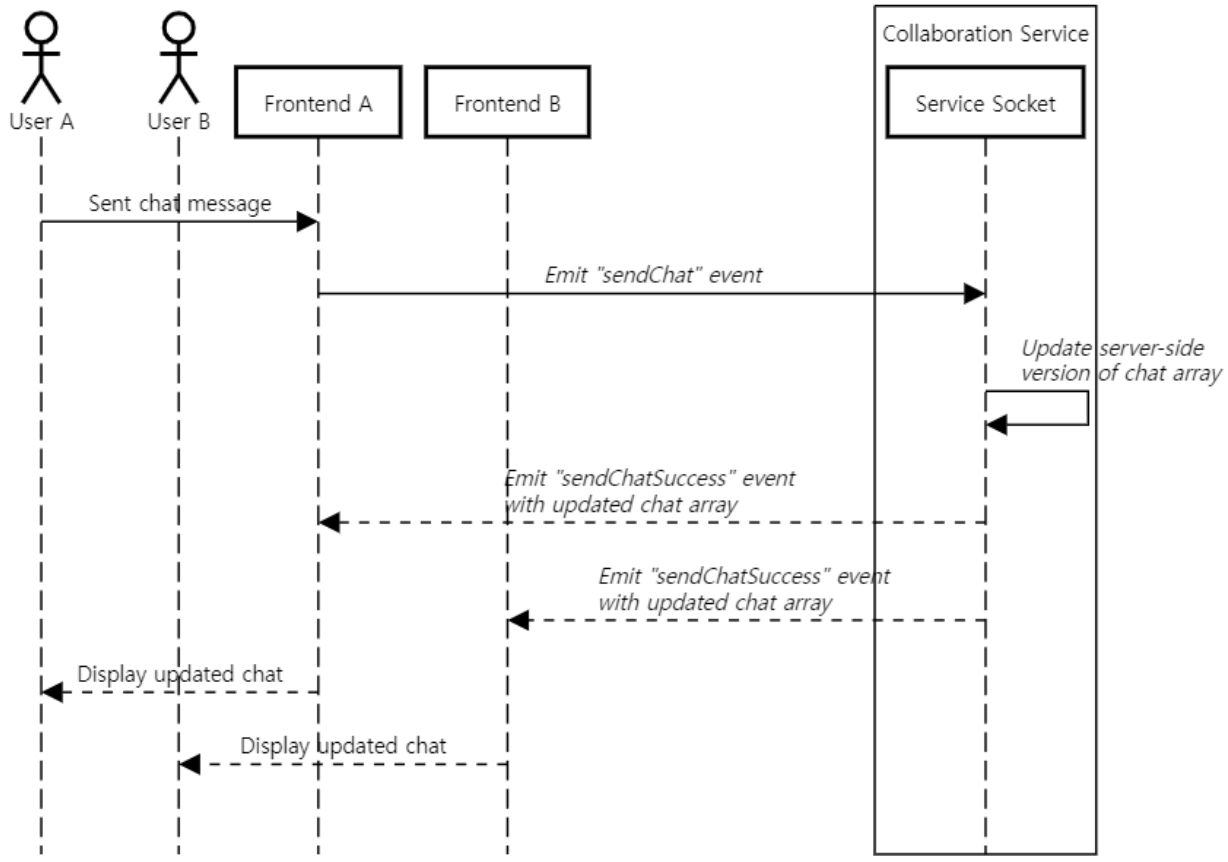

Figure 5.4.4.3.a: PeerPrep Collaboration MS video call communication sequence

The Collaboration MS also facilitates chat communication between two users using WebSockets. The service listens for message send events, from which it appends the message to the chat array of the correct match and broadcasts the updated chat array to all users in the room named with the match ID. This way, messages can be exchanged between two users whilst persistently maintaining the messages in memory such that they can be retrieved by a user should they unexpectedly disconnect and rejoin the collaboration space.

5.4.4.4 Collaborative coding

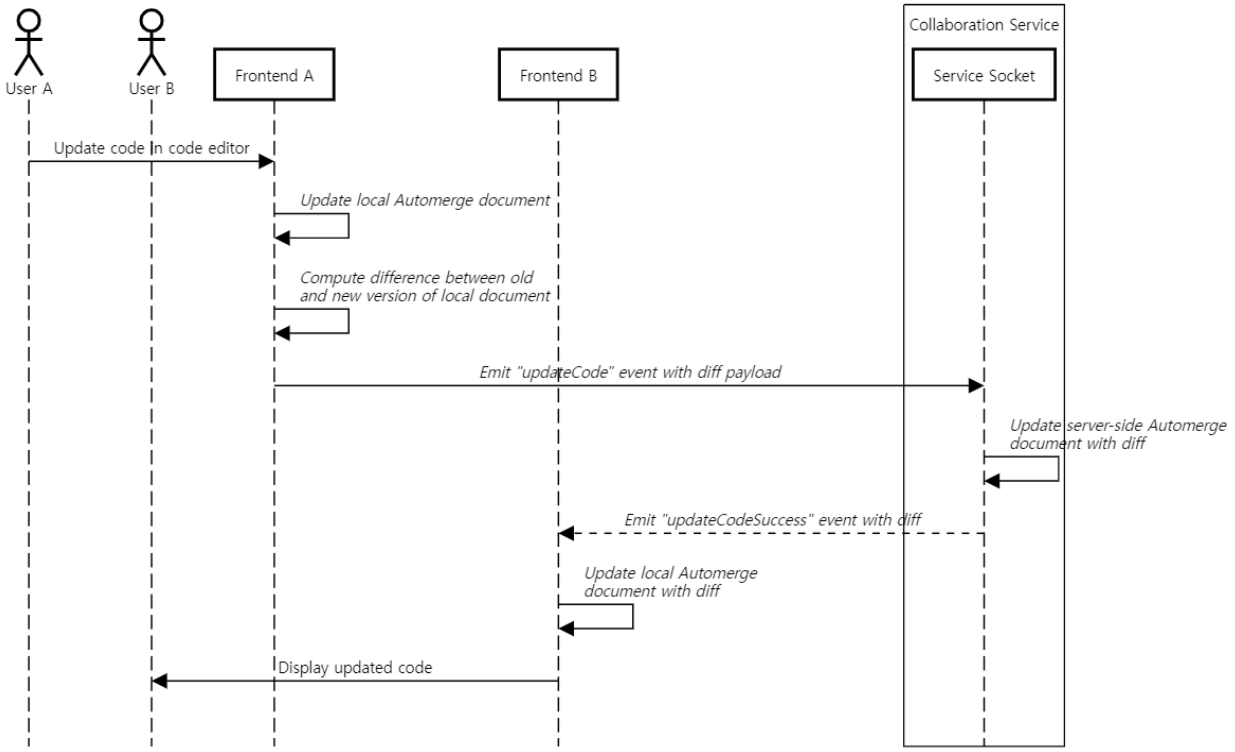

Figure 5.4.4.4.a: PeerPrep Collaboration MS collaborative coding sequence

The code editor in the collaboration space handles real-time collaboration between two users via WebSockets. The only part in which complexity arises is with handling edit conflicts between two users, as one user's changes may overwrite the changes of another. This is resolved by the service by using Automerge as it provides a CRDT that allows detection of conflicting changes and successfully resolves them such that both users can maintain the same version of the document. Users are also able to switch between different languages for the code editor, which is again simply achieved using WebSocket events.

5.5 Frontend

Working with React to design the frontend naturally led to the adoption of the Atomic Design Pattern, which is defined as a methodology for building component-centric interfaces. Each type of component can be defined as either an atom, molecule, organism, template, or page, in which each category is a composition of its preceding category, e.g. molecules consist of atoms, organisms consist of molecules, etc..

The adoption of this pattern has allowed for rapid development of the frontend interface, as components could be reused throughout the application, eliminating the need to create a new component for every new page. For example, a custom button — a relatively smaller component in the context of the Atomic Design Pattern — was reused throughout different parts of the application such as the login form, sign up form, and level select page.

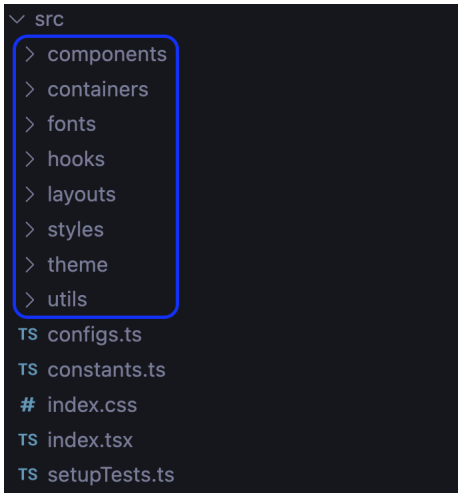

Figure 5.5.a: PeerPrep Frontend Atomic Design Pattern file structure

React is also powered by Hooks which readily allows state-related operations to take place. States are used extensively throughout the system to alter the data displayed to the user as well as internal information such as the user's authorisation status. In particular, the `useContext` hook accompanied by React Router v6 allows React to perform declarative routing whilst globally retrieving and maintaining the user's authentication status. In the context of PeerPrep, the route layout of different pages is as follows:

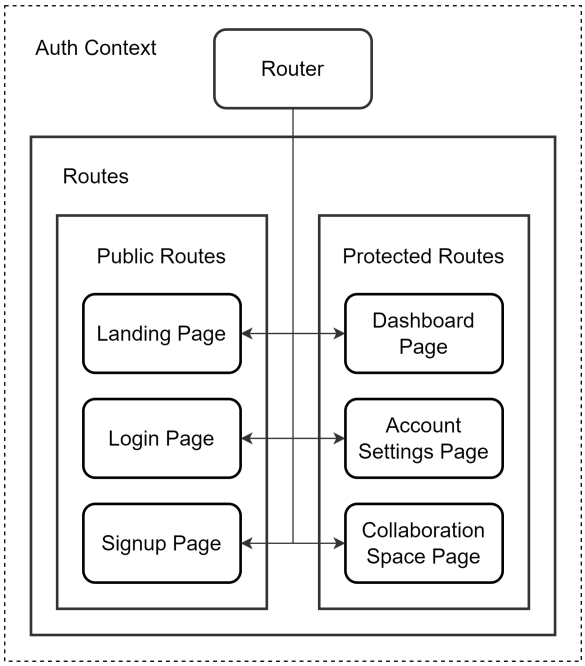

Figure 5.5.a: PeerPrep Frontend route layout

Some key benefits achieved through using React and thus the Atomic Design Pattern are as follows:

- **Virtual Document Object Model (DOM)**
  - React uses a virtual DOM that allows rerendering to take place efficiently in the frontend; effectively, for every rerender caused by state changes in the application, the virtual DOM is used to identify differences in state and only update affected components.

- **Easier file navigation**
  - The compartmentalisation of different components allows for an intuitive file structure that divides each component under their respective category.

- **Development efficiency**
  - The ability to reuse components throughout the application accelerates development as components need not be created for every individual page. Coupled with React Props, common components can allow a certain degree of customisation for specific use cases as well.

The sequence of activities that occur during the interaction between the frontend and microservices are illustrated in the following sections.

5.5.1 Signup

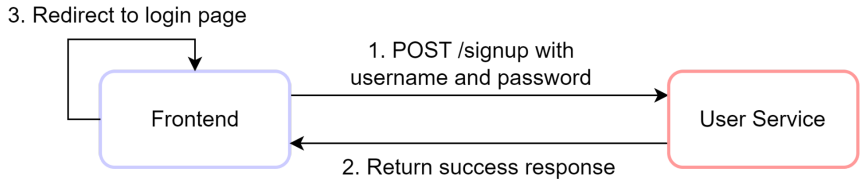

Figure 5.5.1.a: PeerPrep Frontend signup activity

Upon entering the signup page and submitting a username and password, the frontend makes a POST request to the /signup endpoint of the User MS from which the service queries its database to check if the username exists. If the username is already taken, an error message will be displayed to the user to notify them of a duplicate username. Otherwise, the account is successfully created and the user is redirected to the login page.

5.5.2 Login

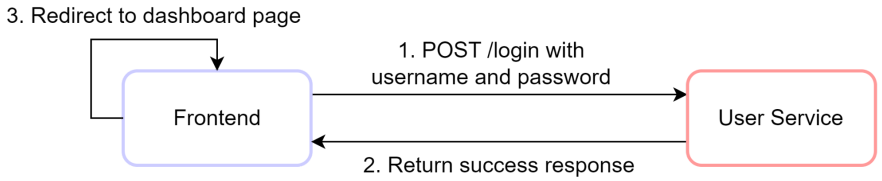

Figure 5.5.2.a: PeerPrep Frontend login activity

Upon entering the login page and submitting a username and password, the frontend makes a POST request to the /login endpoint of the User MS from which the service verifies that the username and its corresponding (encrypted) password can be found in its database. If an account is successfully found, a JWT token of the user is generated and returned to the user, and the user is ultimately redirected to the dashboard page. The token is used to verify the authentication status of the user for every User MS related request; this process is discussed in more detail in Section 5.4.1.6.

5.5.3 Logout

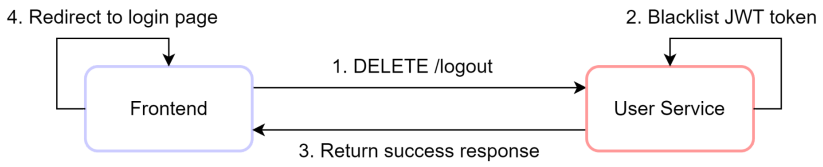

Figure 5.5.3.a: PeerPrep Frontend logout activity

When a user attempts to logout, the frontend makes a DELETE request to the /logout endpoint of the User MS which invokes the service to blacklist the JWT token and clear the token cookie in the browser on which the frontend application is running.

5.5.4 Update password

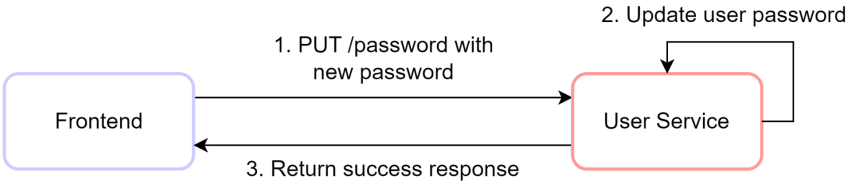

Figure 5.5.4.a: PeerPrep Frontend update password activity

When the user attempts to change their password, a PUT request is made to the /password endpoint of the User MS which triggers an update of the user's password in the database.

5.5.5 Delete account

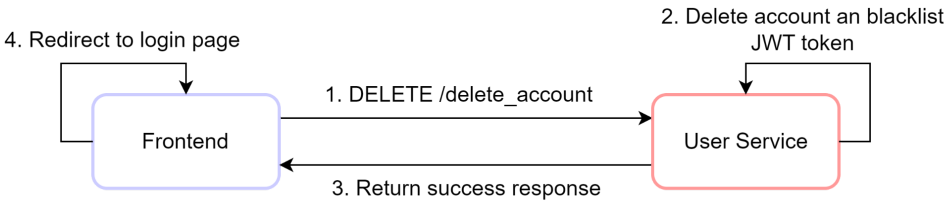

Figure 5.5.5.a: PeerPrep Frontend delete password activity

Upon account deletion, the frontend makes a DELETE request to the /delete_account endpoint of the User MS. The service deletes the specified user account from its database and blacklists the access token in the Redis cache.

5.5.6 Find or create pending match

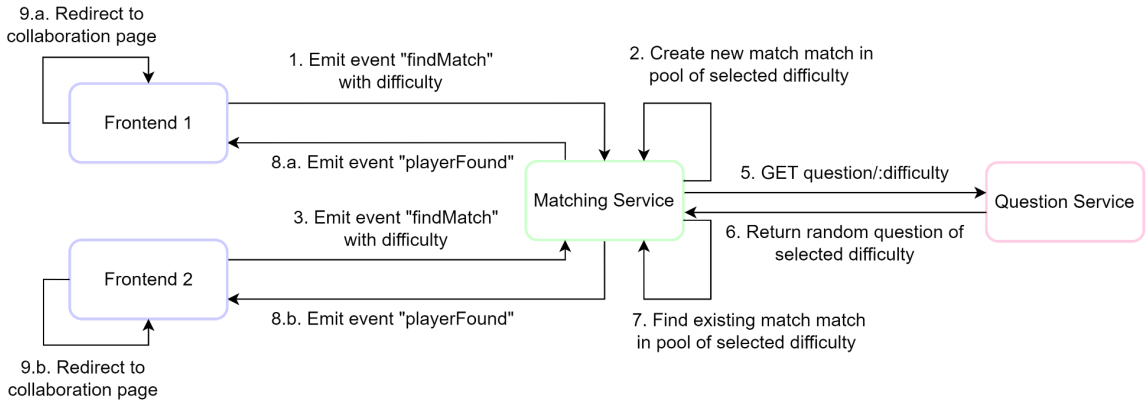

Figure 5.5.6.a: PeerPrep Frontend find or create pending match activity

When the user attempts to initiate matchmaking, the frontend emits a WebSocket event to the Matching MS from which either a new match is created or an existing match is retrieved. In the case where a new match is created, a timeout of 30 seconds will be triggered after which the matchmaking process will be cancelled if a match cannot be found. Observe that from the perspective of the frontend, there is no apparent difference between cases where a new match is created or an existing match is joined.

5.5.7 Leave match

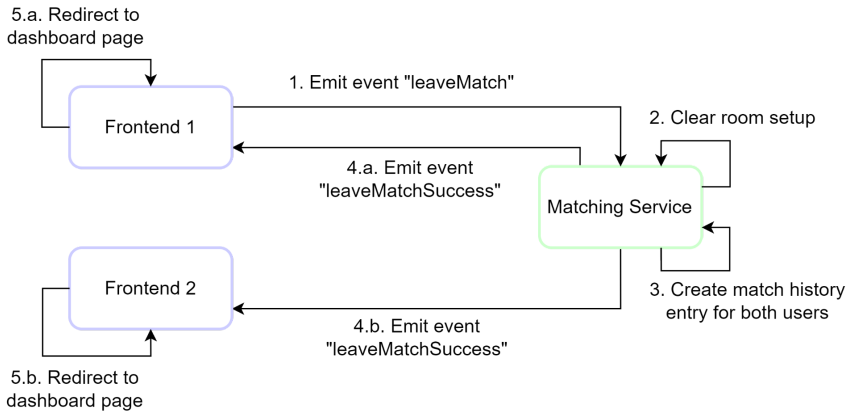

Figure 5.5.7.a: PeerPrep Frontend leave match activity

When a user decides to end a match, the frontend uses the existing WebSocket connection to emit a relevant event to the Matching MS. Upon successful execution the match is concluded and history saved into the service's database.

5.5.8 Video call communication

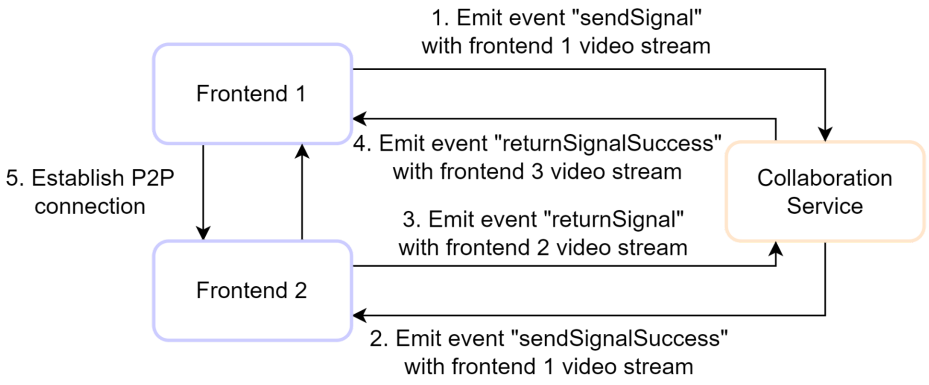

Figure 5.5.8.a: PeerPrep Frontend leave match activity

Video call communication is established using both WebSockets and P2P technology; the technicalities behind the connection is described further in Section 5.4.4.2. In essence, the order in which the users join the room affects the endpoints to which each frontend queries.

5.5.9 Chat communication

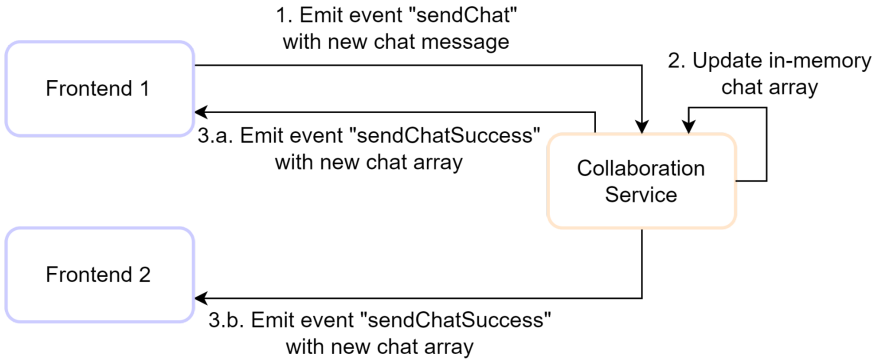

Figure 5.5.9.a: PeerPrep Frontend chat communication activity

Chat message communication also employs WebSockets and is relatively straightforward. As WebRTC is not used for this feature, there exists no direct connection between the two frontends as illustrated in the above diagram.

5.5.10 Collaborative coding

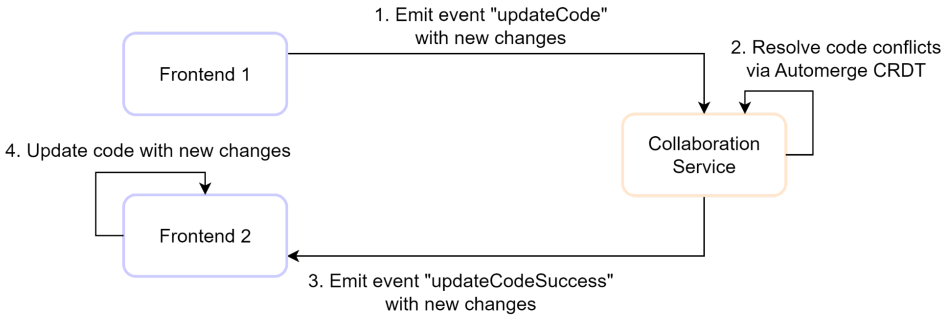

Figure 5.5.10.a: PeerPrep Frontend collaborative coding activity

Collaborative coding is achieved by emitting and receiving changes of the code between the frontends and the Collaboration MS. The service provides a conflict-free solution for concurrent edits and thus abstracts away any handling pertaining to concurrency. The frontend merely receives the changes as produced by the service and applies them to its local version of the code document.

6 USER INTERFACE

6.1 High-Fidelity Prototype

A high-fidelity prototype was created in Figma to draft out the user interface. An interactive version of the prototype can be accessed here.

6.2 PeerPrep User Interface

6.2.1 Landing

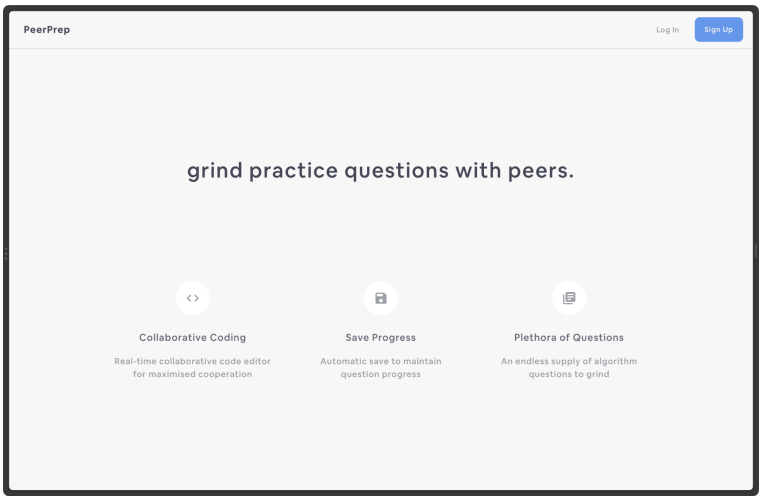

Figure 6.2.1.a: PeerPrep UI landing page

Upon entering the platform, the users are greeted by a welcoming landing page from which they may proceed to login or signup.

6.2.2 Login

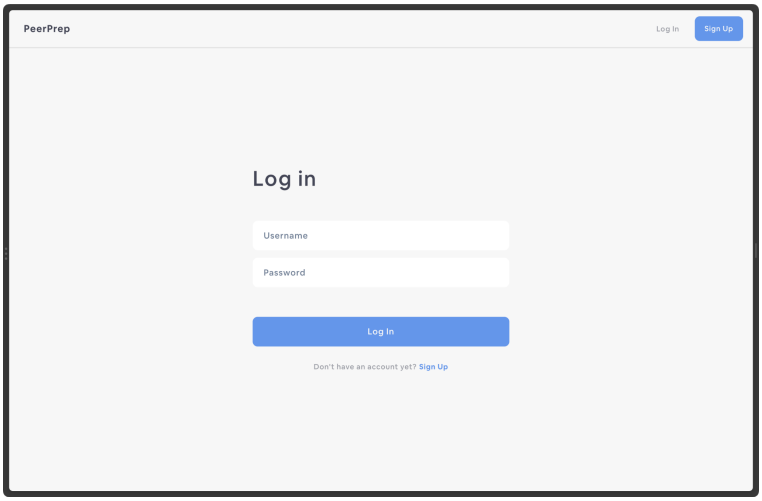

Figure 6.2.2.a: PeerPrep UI login page

The user may submit their account details at the login page if they already have an account.

6.2.3 Signup

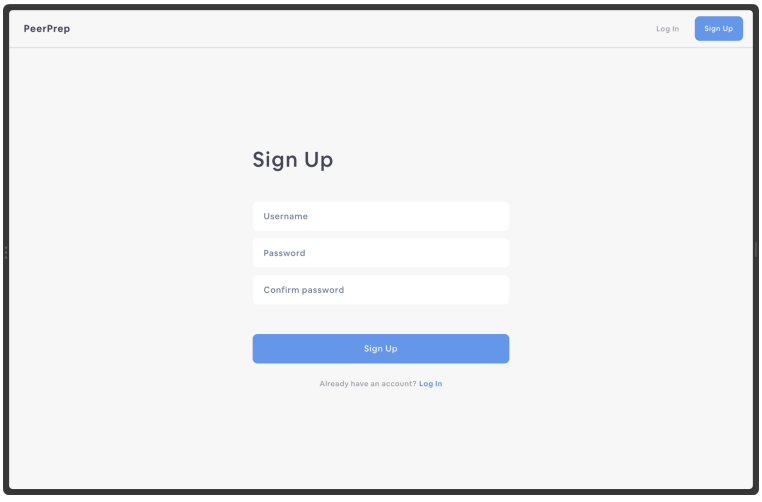

Figure 6.2.3.a: PeerPrep UI signup page

If a user does not have an account yet, they may create a new one at the signup page.

6.2.4 Dashboard

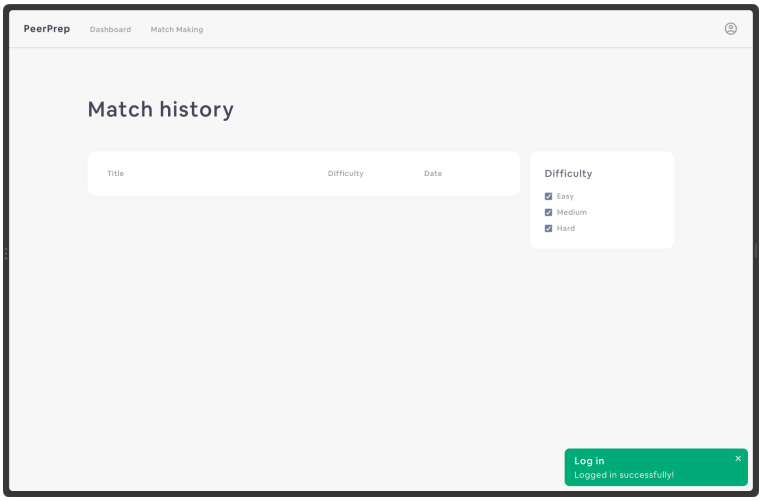

Figure 6.2.4.a: PeerPrep UI dashboard page

Upon successful login, the user is redirected to the dashboard in which they can review their previous matches. In the above screenshot, the match history is empty as the user had just created a new account and has not engaged in any matches yet.

6.2.5 Account settings

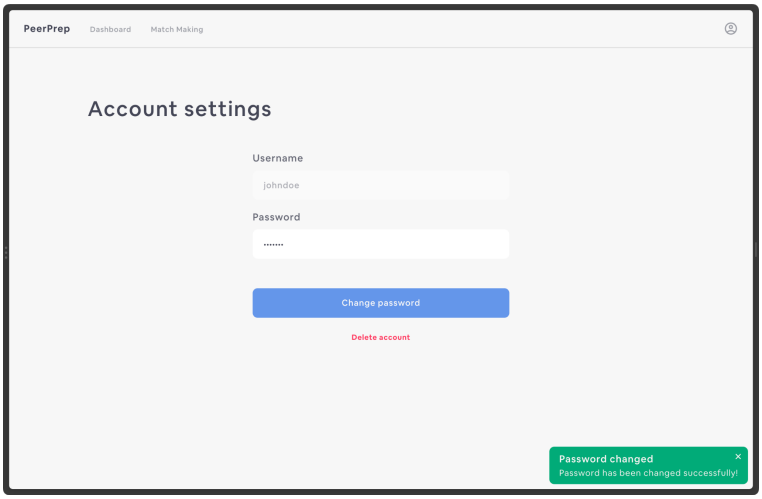

Figure 6.2.5.a: PeerPrep UI account settings page

In the account settings page, the user is able to view their username, update their password, and delete their account. In the above screenshot, the user had just changed their password and was notified of a successful update via the toast in the bottom right.

6.2.6 Matchmaking

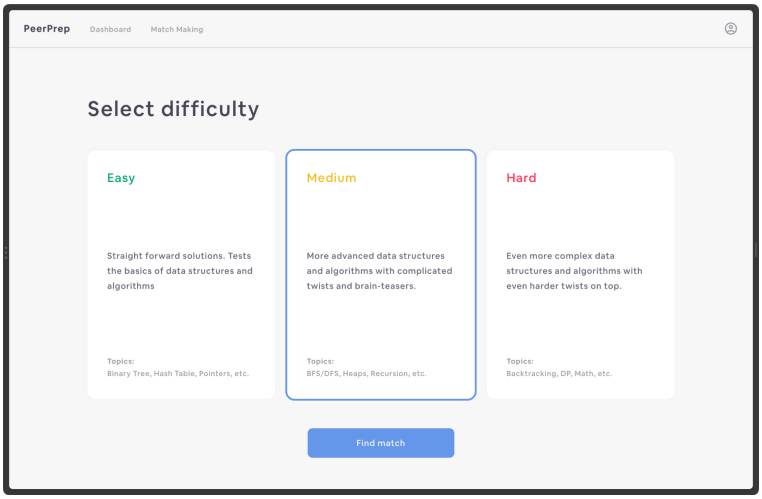

Figure 6.2.6.a: PeerPrep UI matchmaking page

The matchmaking page allows the user to select one of three difficulties and initiate the matchmaking process.

6.2.7 Collaboration space

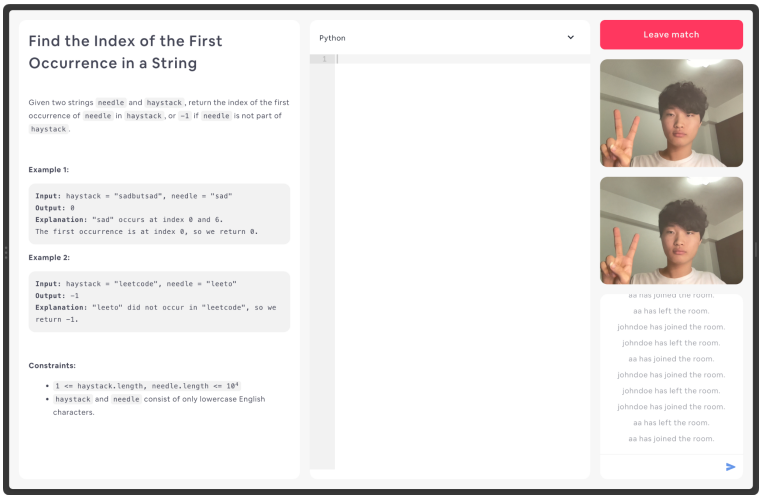

Figure 6.2.7.a: PeerPrep UI matchmaking page

Upon successfully finding a match, the user is redirected to the collaboration space where they may work together on a randomly generated question whilst communicating with each other through video call or chat communication. Note that the above screenshot displays two identical video feeds as the two clients were instantiated on the tester's localhost and thus used the video feed from the same webcam.

7 PROJECT MANAGEMENT

7.1 Agile Methodology

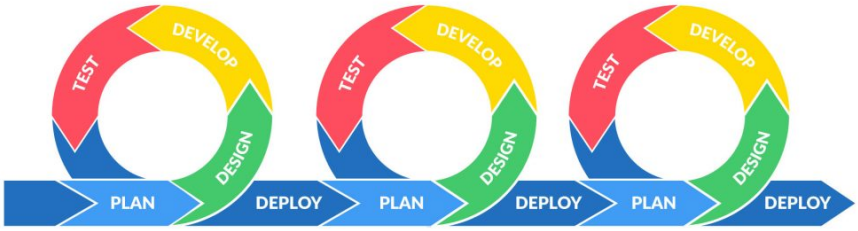

Figure 7.1.a: Agile methodology

The team collectively decided to adopt the agile methodology for PeerPre's development, considering its widespread use in the industry as well its flexibility and consistency in terms of project development. Knowing that around 12 weeks were given to develop PeerPrep, we split up the work to individual sprints, with each spring lasting two weeks; with that, a total of six sprints were conducted for the development of PeerPrep.

The team would hold a meeting at the beginning and end of every sprint to (1) plan the sprint and (2) evaluate our progress. Our delegated SCRUM master, Jiahao, would facilitate the meetings and would record any noteworthy items during sprint planning and stand-up. The sprints also gave the team the opportunity to identify any unresolved tickets and backlog them for the next sprint. By concretely covering "what worked" and "what didn't work", the team was able to determine how to improve the development workflow to further expedite development.

7.2 Sprint Schedule

7.2.1 Overview

The following is a high-level illustration of the project timeline that displays each member's responsibilities for every sprint. The following legend maps colours to each major implementation/task:

- Project planning
- User MS
- Question MS
- Matching MS
- Collaboration MS
- User Interface
- Report

Week	3	4	5	6	Recess	7	8	9	10	11	12	13
Sprint	1		2		3		4		5		6	
Peter												
Nadine												
Andy												
Jiahao												

A more detailed representation of each member's responsibilities is available in Section 7.2.2.

7.2.2 Team Responsibilities

Sprint	Peter	Nadine	Andy	Jiahao
1	Project planning: • Craft rough project schedule • Brainstorm implementation for Collaboration MS and User MS FRs • Expand on existing NFRs			
Review	Sprint retrospective: • Discussed team's understanding of the project scope and finalised FRs and NFRs for Collaboration and User MS Sprint Goal: • Peter and Nadine: implement FRs for User MS • Jiahao and Andy: implement FRs for Matching MS			
2	User MS: • Implement FR1.1-1.6 Frontend: • Draft up rudimentary UI		Matching MS: • Implement FR2.1-2.3, 2.5	
Review	Sprint retrospective: • Discussed implemented features and onboard implementation to each other • Discussed completed backlogs • Discussed Milestone 1 review with mentor; what we have done well and what can be improved Sprint Goal: • Andy and Nadine: implement FRs for Question MS • Peter: test features for User MS and design UI • Jiahao: resolve backlog and improve Matching MS			
3	User MS: • Validate model schema and manually test data access UI: • Create high-fidelity prototype and begin actual implementation	Question MS: • Implement randomisation algorithm for question retrieval • Create rudimentary array of questions in JSON file • Set up WebSocket connection		Matching MS: • Fix issue with unreliable WebSocket connection • Implement Redis cache for pending match storage
Review	Sprint retrospective: • Discussed completed backlogs • Verified completion of features for Question MS Sprint Goal: • Andy and Nadine: implement FRs for Collaboration MS • Peter and Jiahao: improve FRs for Matching MS and implement UI			
4	Matching MS: • Improve connection reliability with existing implementation Frontend: • Implement UI for	Collaboration Service: • Set up WebSocket connection • Experiment with chat box feature for collaboration space • Implement join room feature for collaboration space • Experiment with real-time collaborative code editing		Matching MS: • Perform manual testing with new Redis cache implementation • Implement persistent data storage for match

	landing, login, and signup pages		history
Review	Sprint retrospective: ● Discussed completed backlogs ● Discussed rudimentary implementation of collaboration service ● Discuss progress on frontend implementation Sprint Goal: ● Peter and Nadine: implement FRs for collaboration service ● Jiahao: integrate Matching MS with Frontend and Question MS ● Andy: resolve backlog and draft out report		
5	Collaboration MS: ● Implement real-time collaborative coding ● Complete chat message feature ● Implement language switch for code editor	Backlog: ● Experiment with real-time collaborative coding Report: ● Draft out sections and begin rough work	Question MS: ● Integrate with Matching MS for question retrieval
Review	Sprint retrospective: ● Discussed completed backlogs ● Discussed complete implementation of Collaboration MS Sprint Goal: ● Peter: integrate MSes, complete frontend implementation, and complete report ● Jiahao: integrate MSes and complete report ● Andy and Nadine: complete report		
6	Integration: ● Successfully integrate all MSes with frontend and with each other ● Complete front UI implementation ● Finish up report		

8 SUGGESTIONS AND ENHANCEMENTS

8.1 Deployment

Due to time and resource constraints, the team was unfortunately unable to deploy PeerPrep on time. Without deployment, the team was unable to test whether PeerPrep would work effectively in a real-life scenario, as testing was limited to being conducted locally on one device. Containerising the application and applying load-balancing with Kubernetes would have effectively prepared PeerPred for development as well. A future enhancement would be to dockerise the application such that testing and deployment can be carried out in a consistent manner.

8.2 API Gateway

Currently, the PeerPrep frontend queries the endpoints for each individual microservice whenever a particular service is required. This makes it difficult to scale the application as the number of endpoints to maintain in the frontend grows proportionally to the number of services. A way to mitigate this would be to employ an API gateway that forwards all frontend queries to the correct microservice. This way, the frontend would only need to maintain the API gateway as opposed to maintaining the endpoints for all microservices.

8.3 More than two users in collaboration space

In terms of improving the functionality of PeerPrep, we believe that increasing the maximum number of users in the collaboration room could be of great benefit to some users. A specific use case is when two users struggle to solve a hard question together and require the help of additional users. Thus, scaling the system to account for more than two users in the collaboration space would improve the usability of PeerPrep as the needs of the target user group are more appropriately met.

9 EFFORT

This section details the must-have and nice-to-have features that were implemented in PeerPrep as demonstration of the team's effort.

9.1 Communication MS

Although technically categorised under the Collaboration MS, the features of the Communication MS are prominent in the application, namely, in the collaboration space. To emulate a real-life technical interview experience, the collaboration space provides the ability to communicate with a part via chat or video call. This means of interaction is typical in that of mainstream technical interviews, as the interviewer often requires the interviewee to communicate their thought process. The implementation of the Communication MS via WebRTC and WebSockets, gives way for effective communication between two users, providing great value to individuals in our target user group.

9.2 Fancy UI

The team spent a lot of time and effort designing a sleek and intuitive UI, and believes that this was achieved well. A mockup of the UI was first designed in Figma, which was a powerful design tool with its interactive prototype feature. With respect to Section 4.2.5.1, an intuitive UI not only is visually pleasant to the user, but is also more accessible to a wider audience, as the flow of interaction is made clearer to the user. By adopting Nielson's 10 Usability Heuristics, especially "Aesthetic and minimalist design", throughout the UI, user experience is greatly enhanced; in particular, the interface's consistent colour scheme paired with its clean layout of components effectively eliminates potential distractions and keeps the user engaged.

9.3 History MS

Though a part of the Matching MS, the features of the History MS can be accessed in the application, namely in the dashboard page. Upon successful completion of a match, the user's dashboard is populated with previous matches, which the user can reference to keep track of their progress. The reason for implementing the History MS under the Matching MS is to isolate all concerns regarding matches under the Matching MS. This allows for the grouping of relevant data and operations such that potential chained operations can occur in close proximity with each other, improving system performance and efficiency.

9.4 Collaborative Coding Conflict Resolution

The collaborative coding feature was initially implemented simply using operations that update the entire code document for every change. Although this implementation worked well at the start, the team later realised that this may cause issues in a deployment environment, as users may make concurrent edits that could potentially overwrite each others' changes. To prevent this, the Automerge library was employed in the Collaboration MS to collect changes in local versions of users' code documents and generate a conflict free version that is forwarded back to the users. This allows the Collaboration MS to serve a "single source of truth" where users can retrieve a universally consistent version of the code document.

9.5 Feature-packed Code Editor

The collaborative code editor provides numerous features that may be of value to the user, such as syntax highlighting and code auto-completion, as well as the ability to switch languages. Though a rather simple implementation, the effects of it are significant as users are able to enjoy a more pleasant experience on the platform. With mainstream coding platforms like HackerRank providing the aforementioned features, the team thought PeerPrep would be easier to use if they adopted similar features.

10 REFLECTIONS

Below are individual reflections from the members of Team 44, pertaining to the development of PeerPrep and the learning experience along the way.

10.1 Jung Hyunseok (Peter) (A0220141E)

This team project was a great learning experience through and through, from learning new technical skills, to understanding the significance of architectural designs and software design patterns, to realising the importance of teamwork and communication in a fast-paced, deadline-driven team environment.

Most evidently, this project was a grand test of my abilities in learning new technologies and quickly adopting them in a large-scaled project. Before this project, I had not worked extensively with NoSQL databases before, specifically MongoDB, and struggled quite a bit with understanding how to use the technology as well as best practices for adopting them. However, thanks to the team's selection of a strongly-supported technology stack, there were a lot of great resources that helped onboard the technology. I was able to appreciate the importance of community support for modern technologies and how it can greatly help to accelerate project development.

The project was also an opportunity for me to realise how a clear definition of a system's architectural design and the software design patterns it adopts can help to refine the direction and thus the implementation of a realistic and useful product. That is, I learnt that besides the technical implementation of the product itself, it is crucial that the anatomy of the product is explicitly defined as well, as to ensure not only understandability of the system as a whole and how its components work together, but also improve maintainability as future developers would have a better idea of what creative or technical direction to take the product in.

The importance of teamwork and communication truly shined during our project as well. There were often times where miscommunication within the team would lead to an incomplete implementation of a feature or multiple members implementing the same feature, which effectively decreased development productivity. For instance, due to misunderstanding of the project requirements, two members of the team implemented the code editor separately, meaning that one of either had to be scrapped during integration.To tackle this, the team decided to more clearly outline the list of tasks during our bi-weekly SCRUM meetings, and to delegate tasks thereafter as well. It was not easy adopting this change at the start, because each member was still figuring out their strengths and weaknesses in the team and felt as though the work they were delegated were not applicable to the skills that they had. As we spent weeks upon weeks working together, however, the team gained a better understanding of each others' requirements and ultimately was able to delegate skill-relevant tasks effectively.

There are a lot of things I wish I had done differently for the development of PeerPrep, namely spending more time during the planning stage and focusing on the deployment side of things, but nevertheless I am proud of the team's final product and am grateful for the lessons that were taught in CS3219 as they greatly helped to pave the way for the product's development.

10.2 Zhou Jiahao (A0218561E)

This is the first micro service project that I attempted. The experience was very fresh. It is interesting to apply theories that we have learned in class on the design of our software.

Micro service allows different parts of the software to be developed concurrently without having blockers in between. I think this is a huge advantage for group work as it greatly reduces the wait time in between members' work. One could just focus on one service at a time.

As a team project, we get to discuss our implementation and design choices. The discussion was often very fruitful. By questioning each other's reasoning and suggestions, we were able to make trade offs and come up with the optimal solution that everyone is satisfied with. For example, we had a long discussion about how the question should be generated after a match is successful. We list out a few possibilities and refine them. I think the final solution was elegant and the most logical one.

Doing this project also allows me to pick up new tech stacks that I am unfamiliar with. For example, I have very limited server development experience. Doing the server development this time has allowed me to learn things such as Socket. I also get to do things that I did not try before. For example, I normally only use 1 database in any project. After learning about different databases in class, I used both Redis and MongoDB to take advantage of both databases.

It has been a year since I did CS2103 and doing this project helped me to re-emphasize many of the points raised in both CS2103 and CS3219 such as project management as well as SOLID principle.

Overall, I think the project is meaningful and I learned a lot from it. However, we could have focused more on the design patterns to apply and explore more possibilities after learning about them.

10.3 Andy Lam Wei Jie (A0218309E)

I think that the project allowed me software engineering concepts such as architectural design and design patterns better. In particular, for microservices, I realised both the benefits and disadvantages of using microservices.

For example, while using microservices, it is easier for us to divide and conquer the project, in a sense, split the work and focus on individual areas. This allows us to create our software in a swift manner. In addition, with microservices, we can simply add new services, this also means high scalability. However, when using microservices, since it's more modular, there are more things to integrate when putting everything together. This, in essence, increases the project of complexity.

Besides architecture design, I also learnt a few things about design patterns. As I was involved in designing the backend of matching service as well as collaboration service, I was very involved in using sockets to facilitate communication between two frontend users. This helped me to understand the pub-sub pattern as our program uses the pub-sub pattern, making receivers subscribe to messages and perform actions accordingly.

Besides software engineering concepts, I also learnt to think about the future developments of a project. Essentially, when developing software, many times, we will require the use of external libraries. In order to ensure continuity, we should always use libraries that are more well-maintained. If we use libraries that are not well-maintained, in future, there may be bugs that surfaced and this might affect our software if that library is a critical part of the service that we want to deliver.

This project has also allowed me to better improve my problem solving skills. While faced with problems, I learnt to search for the right keywords to the problems to find existing solutions to the problem. Thereafter, I learnt to adapt the solutions to the problem faced.

Lastly, I learnt that communication is extremely important in a project setting. Communication ensures that everyone is on track with the progress of one another. When faced with issues, it is better to bring the issues up and resolve them so that the team can resolve the problem amicably and move on smoothly. During the project, we had some disagreement regarding the design and implementation of certain services. When someone disagrees with the implementation, we always surface the problem for discussion so that we can come to a conclusion on how something should be implemented.

Overall, I feel that the project has sharpened both my knowledge and skills as a computer scientist, helping me become more proficient at what I do.

10.4 Chiun Jia Qian Nadine (A0221134Y)

Learning about the theory and uses behind the architectural designs and patterns in lecturers, it is eye opening for me to see these concepts being applied in real-life, via this project. I have definitely gained a bigger appreciation for these concepts that were taught during this module as I was fortunate to be able to apply it. As an aspiring software developer, eager to start immediately on any project I have, I had the bad habit of coding immediately without having a concrete plan on what I should implement in the projects. However, this project taught me that although implementing the product is important, having an idea of what to implement, and a seat of rules to follow is equally important. Having implemented the database and its relevant methods for question service, if I were to not follow the repository and chain of responsibility pattern, it may lead to the code structure being messy in the long run, filled with tightly coupled methods.

Secondly, in terms of project management, I have realised how using the agile methodology significantly improves the team's development efficiency. Spreading tasks across sprints, each sprint having a specific goal, really allows us as a group to focus on one thing at a time, treating the development process as a marathon of tiny sprints across each period. Also, these sprints where we update each other on tasks helps to improve our communication as well. Overall, having utilised the agile methodology has improved on both our team collaboration, communication, and efficiency.

In conclusion, this project has definitely been worthwhile, and I better understand the concepts taught in lectures and how it is applied in the industry.

11 GLOSSARY

Terminology	Definition
CS	Computer science
UI	User interface
CRDT	Conflict-free Replicated Data Type
MS	Microservice
FR	Functional requirements
NFR	Non-functional requirements
First contentful paint	According to web.dev (2019), the time it takes the browser to render the first piece of DOM content after a user navigates to a page [2]
DB	Database
XSS	Cross-site scripting
JWT	JSON Web Token
OS	Operating System

12 REFERENCES

[1] “ARIA - Accessibility.” Mdn Web Docs, MDN, 3 Oct. 2022, developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA.

[2] “First Contentful Paint.” Web.dev, web.dev, 2 May 2019, web.dev/first-contentful-paint/.

[3] Microsoft. “Designing the Infrastructure Persistence Layer.” Microsoft, Microsoft, 21 Sept. 2022, learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design.

[4] Thayer, R.H. “Software System Engineering: A Tutorial.” Computer, vol. 35, no. 4, 2002, pp. 68–73, 10.1109/mc.2002.993773.