NUS
National University
of Singapore

CS3219, Software Engineering Principles and Patterns (Semester 1, AY2023/24)

Group 54

Team Members	Student No.
Loh Xian Ze, Bryan	A0217636B
Gowri Lakshmikanth Bhat	A0240374L
Wong Yew Jon	A0234230X
Gerald Teo Jin Wei	A0234645A

Table of Contents

1. Introduction

1.1. Background and purpose

When preparing for technical interviews, we believe that mastering Leetcode problems is just one part of the equation. Beyond crafting solutions, effective communication of one's thought process is equally crucial. To tackle this issue, we created OhMyGode, a platform that connects you with a matched peer in real time. It provides a shared space to work on questions together and gain experience for technical interviews.

1.2. Must have features

- User Service – responsible for user profile management.
- Matching Service – responsible for matching users based on some criteria (e.g., difficulty level of questions, topics, proficiency level of the users, etc.) This service can potentially be developed by offering multiple matching criteria.
- Question service – responsible for maintaining a question repository indexed by difficulty level (and any other indexing criteria – e.g., specific topics).
- Collaboration service – provides the mechanism for real-time collaboration (e.g., concurrent code editing) between the authenticated and matched users in the collaborative space.
- Basic UI for user interaction – to access the app that you develop.
- Deploying the application on your local machine (e.g., laptop) using native technology stack OR on a (local) staging environment (e.g., Docker-based, Docker + Kubernetes)

1.3. Nice to have features

We have implemented additional features like:

- N1: Video calling service to facilitate communication and simulate the interview environment
- N2: Save past attempts of questions along with date, time, collaborator name, language, title
- N5: Enhance code editor with syntax highlighting and code formatting for Python, Java and Javascript
- N8: Extensive (and automated) unit, integration, and system testing using CI
- N9: Application is deployed on the production system (EC2)
- N11: Application uses NGINX on the production system to redirect requests to other microservices

2. Requirements

2.1. Functional Requirements

2.1.1. Questions

S/N	Description	Priority	Justification
FR1.1	Users should be able to see a list of questions, including their titles, difficulties and tags.	High	This is a core feature of our project. Users need to be able to see the questions to prepare for interviews or maintain the website content
FR1.2	Users should be able to view each question individually	High	This is a core feature of our project. Users need to select questions to view the question prompts as they are too long to be shown in the list.
FR1.3	Users should be able to write their own programs for each question	High	This is a core feature of our project. This feature is very important for users trying to prepare for their technical interviews.
FR1.4	Maintainers should be able to add a new question	High	Website maintainers should be able to add in new questions to the platform when needed to maintain the currency of the list of questions.
FR1.5	Maintainers should be able to delete an existing question	High	Website maintainers should be able to delete questions from the platform when needed to maintain the currency of the list of questions.
FR1.6	All questions should be persisted across browser sessions and be able to be viewed by all users.	High	This is a necessary feature of our project. Website maintainers should only need to add/delete questions from the application once for it to be reflected on all users' perspectives.
FR1.7	Maintainers should be able to edit an existing question, including its title, description, difficulty and tags.	Medium	Website maintainers should be able to add in new questions to the platform when needed. Editing can also be done by deleting and thereafter creating a new question, hence the priority.

FR1.8	Users should be able to search for the questions by title, complexity and categories.	Medium	Users should be able to easily filter the question based on their needs. Since our list of questions is still small, this feature is not as important yet.
FR1.10	Maintainers should be able to fetch questions from leetcode to populate the question repository.	Low	Website maintainers should have an easy way to populate the question repository without needing to create questions by themselves. This is something to enhance the maintainer's experience but does not affect users' experience in our application.

2.1.2. Users

S/N	Description	Priority	Justification
FR2.1	Users should each have a user account.	High	This is to ensure that each user can be identified via the account. Identification is necessary for matchmaking and collaboration.
FR2.2	Users should be able to login to their account only if they know the credentials.	High	This is to ensure that unauthorized access to the user accounts are prevented.
FR2.3	Users should be able to log out of their accounts.	High	This is to prevent unauthorized use of the account by other people on the same computer.
FR2.4	Users should be prevented from creating accounts with weak passwords.	High	This is to ensure that passwords are not guessable and increases the security of the user accounts.
FR2.5	Users should be prevented from creating accounts with emails that already exist.	High	As user accounts are primarily identified by the email address of the account, this preserves the integrity of the accounts.
FR2.6	Users should be able to delete their accounts.	Medium	This prevents inactive users from filling up the users database and keeps costs low.
FR2.7	Users should be able to see helpful error messages if logging in or signing up is unsuccessful.	Medium	Error messages assist users in identifying and addressing issues on their own without the need for technical support.
FR2.8	Users should be able to change their name and profile image.	Low	Users need to be able to customize and differentiate their own profile from others.

2.1.3. Matchmaking

S/N	Description	Priority	Justification
FR3.1	Users should be able to search for a collaboration partner based on question difficulty.	High	This is a core component of our application. Users need to do this in order to access the collaboration features.
FR3.2	Users should be able to cancel the search process at any time by clicking on a cancel button.	High	This is a core component of our application. We do not want to keep the user in the matchmaking page if they decide to cancel the operation.
FR3.3	Users should be able to see some indication after 30s if they are still not matched.	Medium	This helps to keep the number of simultaneously opened sockets with our server low.
FR3.4	Users who are unable to find a match should be able to retry the search process after clicking on a button.	Medium	This feature is not as important because the user can reinitiate the retry by using the start searching button again.
FR3.5	Users who are successfully matched together should not be moved into a collaboration room automatically.	Low	This is to ensure that both the users are present in front of their computers when the collaboration starts.

2.1.4. Collaboration

S/N	Description	Priority	Justification
FR4.1	The system should make 2 matched users join into the same room.	High	They need to join into the same coding space to begin their collaboration. This is a core feature of our application.
FR4.2	The system should allow users to write code in a code editor simultaneously.	High	Users need to be able to see the code of others in order to be able to communicate and collaborate on the question. This is a core feature of our application.
FR4.3	The system should allow users to leave a session.	Medium	Users should be able to stop their collaboration and close their session when they want to.
FR4.4	The session should terminate when all users have left the session.	Medium	After all users have left the collaboration space, the session should close because their collaboration is finished.
FR4.5	The system should inform other users in a room if someone has left the room.	Medium	This will allow a user to know if the collaboration session is still going on, and if they should keep working on the question.
FR4.6	A user should be able to get the latest version of the code if they disconnect from the system and the session has not been terminated.	Medium	If a user loses connection and is able to rejoin the session, the collaboration session should still continue.
FR4.7	In the editor, users should be able to format code in their chosen language.	Low	This ensures code readability and facilitates debugging and code review processes.
FR4.8	In the editor, users should be able to write code with syntax highlighting available for that language.	Low	This improves code visibility and leads to more efficient and accurate coding.

2.1.5. History

S/N	Description	Priority	Justification
FR5.1	A user should be able to save and access their previous code attempts.	High	This allows the user to save their progress and have future reference when solving similar questions.
FR5.2	A user should be able to know the name of the collaborator when attempting the question.	Medium	The user might want to refer to the collaborator's feedback and comments which might be included in the saved attempt.
FR5.3	A user should be able to get the timestamp of when a question attempt is saved.	High	Users need to be able to differentiate between multiple saved attempts of the same question.
FR5.4	A user should be able to sort and search through past attempts based on question title, time, language and partner.	High	Users need to be able to search through all of their past attempts quickly.
FR5.5	Users should be able to view their most updated questions and users. (e.g Question title is updated, collaborator's name is updated, collaborator's account is deleted)	Medium	This ensures a better overall user experience as they stay informed about any changes to their attempted questions or past collaborators

2.1.6. Video

S/N	Description	Priority	Justification
FR6.1	The system should allow users to see each other via video.	Medium	Seeing each other may not be as important for communicating and explaining the coding process for technical interviews.
FR6.2	The system should allow users to hear each other via video.	High	Users need to be able to effectively communicate to convey their coding process, this is important as verbal communication is an aspect that interviewers look out for.
FR6.3	The users should be allowed to turn on and off the video call whenever they require it.	Medium	Users should be able to turn on and off their video cameras based on individual needs or situational requirements, this ensures their privacy.
FR6.4	The users should be able to reconnect back to the video after closing the video call.	High	Users should be able to reconnect to the video chat if they lose connection or change their mind

2.2. Non-Functional Requirements

2.2.1. Application

S/N	Description	Priority	Justification
Performance (NFR1)			
NFR1.1	Code written by the user should be displayed in real-time.	High	Users need to be able to view their own code as they type to facilitate a smoother user experience.
NFR1.2	Code written by other users should be displayed in near-real time (<1s).	High	The users should be able to see the changes in real-time to facilitate collaboration and reduce confusion.
NFR1.4	CRUD functions of questions and user profiles should be completed within 2 seconds.	High	This is to create a seamless and responsive UI for the user to interact with.
NFR1.5	User interfaces should be responsive and smooth even under heavy loads.	High	A responsive interface will increase the ease of use for the user, and improve user experience.
NFR1.6	If two or more users are looking for matches at the same time, the matching should take less than a second.	Medium	If the matching is too slow, it will negatively affect the user's experience.
Usability (NFR2)			
NFR2.1	The system should be intuitive and easy for users to understand each button and how to navigate through the pages be intuitive enough or well guided by prompts for users to navigate.	High	Users need to understand the capabilities of the application interface in order to bridge the gulf of execution.
NFR2.2	The server should be able to support the load of up to 100 simultaneous searches.	Medium	In order for the searching to be effective, we need to have multiple clients searching simultaneously, hence we need to ensure that we can support this.
Security (NFR3)			

NFR 3.1	Sensitive data should be protected during transmission and storage.	Medium	Sensitive data should be encrypted before it is sent over wireless networks as they can be snooped on by malicious parties.
NFR 3.2	Users' passwords should be hashed and salted in the DB.	High	Even if the database is leaked, user account details should be protected.
Scalability (NFR4)			
NFR 4.1	The system should be able to handle 100 concurrent collaboration sessions and 100 simultaneous matching searches.	High	Website should be able to run smoothly even with many concurrent collaboration sessions and be adapted to a high user load.
NFR 4.2	System should be able to store questions up to 50GB storage space.	High	The application should have a large database of questions to assist students in interview preparation.
NFR 4.3	Should be able to store up to 1000 user profiles.	Medium	The application needs to have a large user pool so as to facilitate user matching and concurrent programming.
Portability (NFR5)			
NFR5.1	The system should be able to run on Chrome, Edge, Firefox and Safari.	High	Different users may have different preferences for web browsers, and these preferences can also change over time.

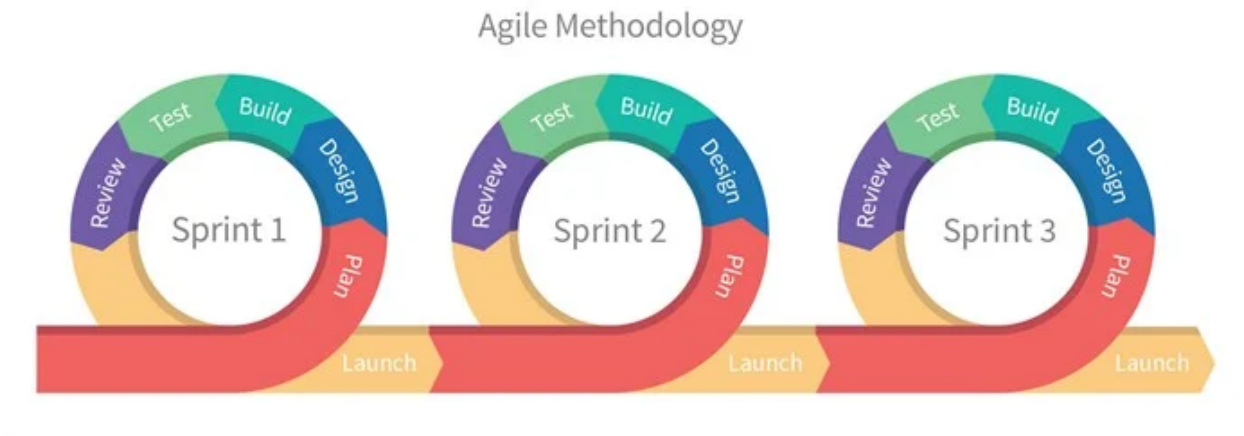
2.2.2. Testing

S/N	Description	Priority	Justification
NFR6.1	Tests should be able to run asynchronously.	Very High	Asynchronous testing allows multiple tests to run concurrently, reducing the overall execution time.
NFR6.2	Developers should be able to run the tests in a development environment.	High	Supports a rapid development cycle by enabling developers to validate changes locally, ensuring quick feedback.

NFR6.3	Tests should run on an environment mirroring production.	Very High	Ensures test independence and data consistency, simplifying test scenarios and preventing data conflicts.
NFR6.4	Each test should easily set up and teardown test data.	Very High	Enhances reliability and accuracy by validating the system in an environment that closely mirrors the production setup.
NFR6.5	Tests should run within 15 seconds.	Medium	Facilitates a quick feedback loop for developers and timely identification of issues.

3. Application

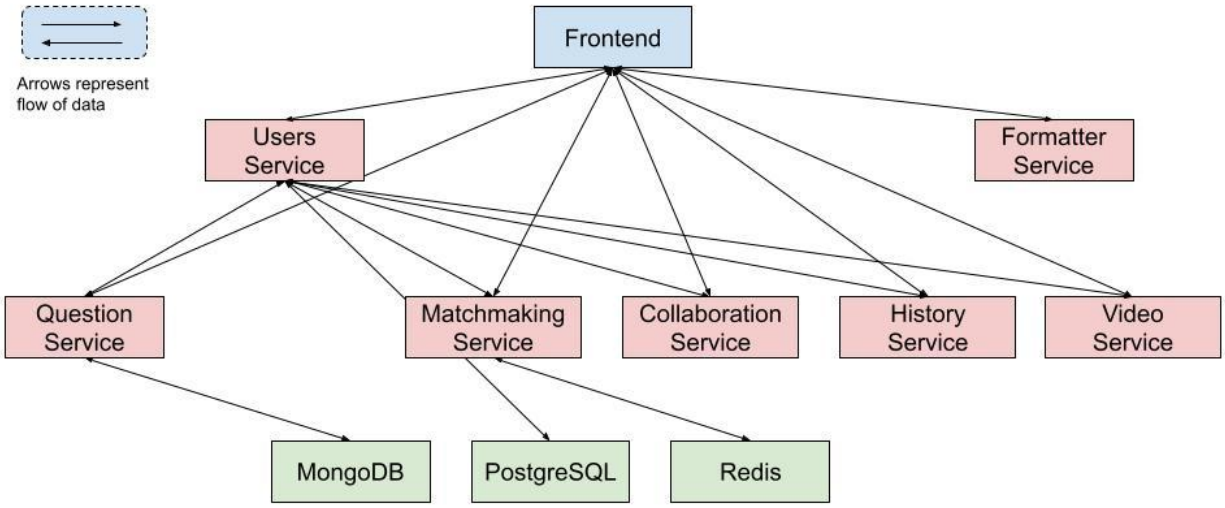
3.1. Software Development Life Cycle



Agile Methodology

We embraced the Agile methodology, placing a strong emphasis on continuous collaboration, iterative review, and ongoing improvement. Our project timeline was organized into dynamic one-week sprints, each encompassing a comprehensive cycle of planning, design, development, testing, and thorough review. This approach allowed us to work in small increments and fostered a culture of continuous refinement throughout the development process.

3.2. Microservices



OhMyGode's Architecture Diagram

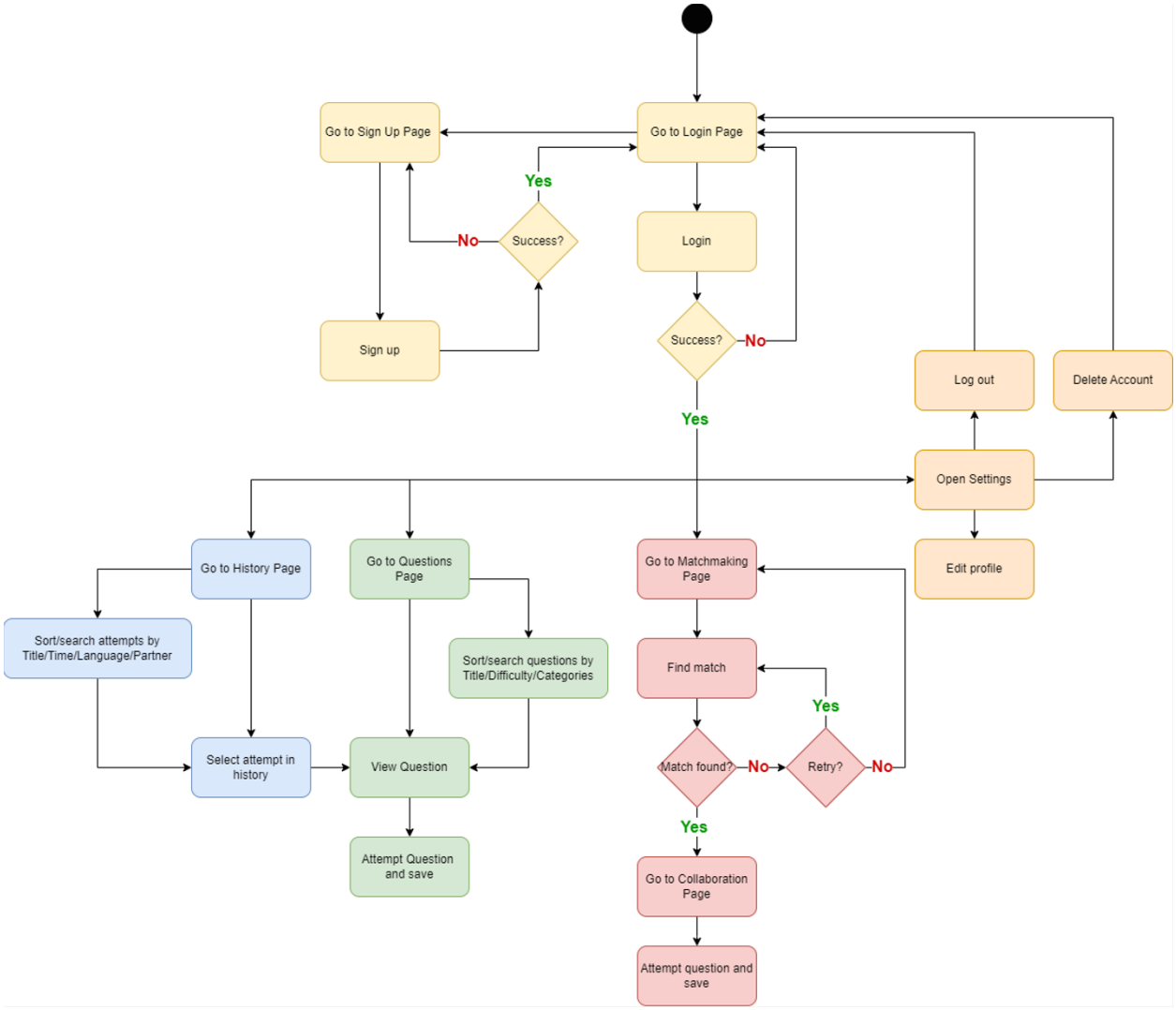
We have adopted the microservices architecture and implemented domain-driven design**.** We decomposed the application by assigning each microservice to a subdomain and a docker container. For example, the User Service is responsible for user-related functionalities like registration, authentication, profile management, etc. This aligns with the Single Responsibility Principle, where each microservice should only have one responsibility and only change based on the requirements of the subdomain. This also provides fault isolation, preventing issues in one service from affecting the entire system. Microservices empower us to use different technology stacks for each service, allowing team members to choose the most suitable tools and frameworks for their specific domain. This approach also facilitates agility by allowing independent development, deployment, and updates of services, reducing dependencies and accelerating our development lifecycle.

3.2.1. Frontend Service

The frontend service serves as the user interface for OhMyGode, providing a minimalist and intuitive interface that is easy to learn and comprehend. It is built using React and Material UI and the service runs on port 3000. It interacts with the other backend services through HTTP requests. Other notable dependencies include Monaco Editor for code editing and PeerJS for video conferencing.

User Flow

The chart below shows the various different user flows that are expected in the frontend.



User flow diagram

Code Organisation

The frontend service of our application is built using React and Material UI. This codebase is organized into three main entities:

1. Pages: High-level views or screens of our application. Each page is designed to encapsulate a distinct user interface, and the code associated with each page is organized to provide a clear separation of concerns. They may be composed of multiple components.
2. Components: Modular building blocks for pages. The reusable units are designed to abstract out certain functionalities to improve the readability of the code.
3. Contexts: Play a crucial role in managing the global state within our application. Leveraging React Context API, we ensure that relevant state information is shared efficiently across components without the need for excessive prop drilling.

Design Decisions

For storage of JWTs, the team debated between two popular methods, browser local storage and cookies. Despite local storage being more vulnerable to XSS attacks as compared to cookies, we decided on using the browser's local storage because we wanted the user sessions to persist even though the user may have closed the browser. This cannot be done using cookies. Since our application is solely focused on technical interview preparation and does not require users to save sensitive information outside of their email, we decided to let go of some security for ease of use of our platform.

In deliberating between Client-Side Rendering (CSR) and Server-Side Rendering (SSR) for our application, we decided on CSR after careful deliberation. CSR empowers the client's browser to handle rendering dynamically, fostering quicker interactions and real-time updates, which we believe is important for our project. In contrast, SSR requires the server to render each page and send fully rendered HTML to the client, which can impose a performance overhead and potentially slow down the server.

3.2.2. Question Service

The question service serves as the question repository of Ohmygode, providing a place for maintainers of our platform to create, read, update and delete technical interview questions. The questions service depends on the users service for request authentication, as well as our MongoDB database for data storage and persistence. The main technologies used in this service are NodeJS and Express.

Database Schema

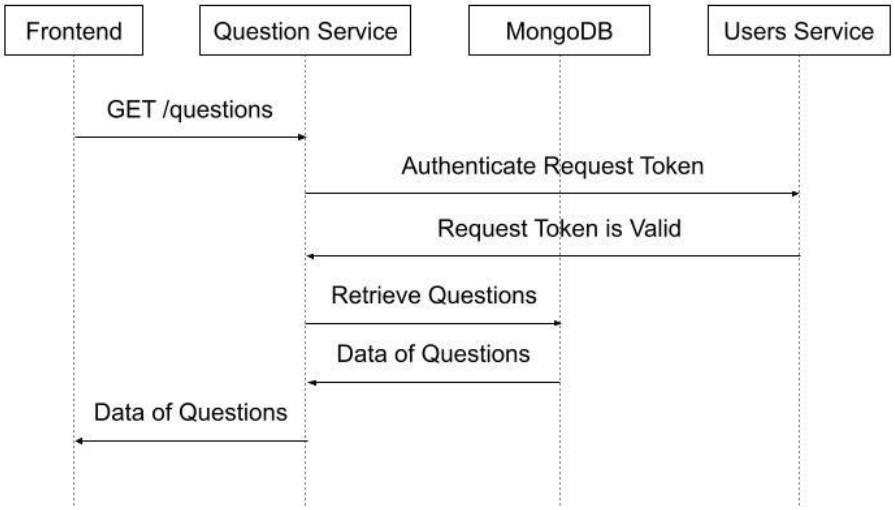
The MongoDB schema of our questions collection is shown below. As shown, the schema for the question repository is small. As such, we kept all the properties within the same collection to increase the efficiency of database reads.

questions	
question_id	Number
title	String
categories	String
complexity	String
link	String
description	String

MongoDB collection of questions

Example Sequence Diagram

The sequence diagram below shows what happens when a GET request is made to the questions service to retrieve the list of questions. The sequence diagram for the create, update and delete endpoints are similar.



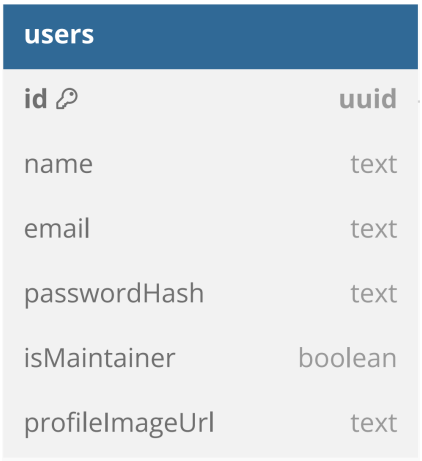
Sequence diagram for GET /questions endpoint

3.2.3. Users Service

The users service serves as the users repository of Ohmygode, providing a place to save details about the users of our platform as well as perform authentication and role checks. Most other services depend on the users service for authentication of request tokens. The main technologies used in the users service are NodeJS, Express and PostgreSQL as our database.

Database Schema

The MongoDB schema of our questions collection is shown below. As shown, the schema for the question repository is small. As such, we kept all the properties within the same collection to increase the efficiency of database reads.

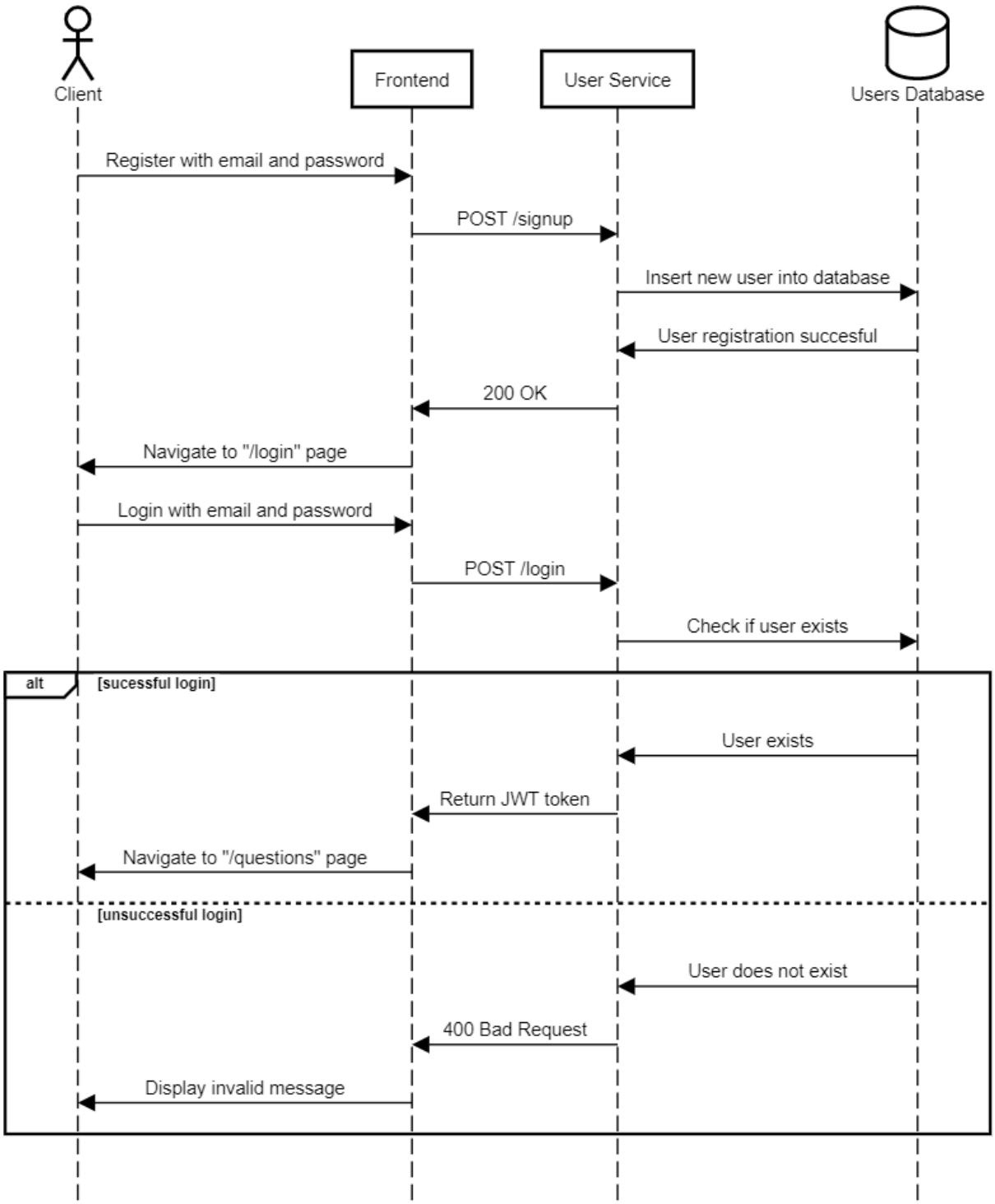


Database schema of users table in PostgreSQL

Authentication

After successfully signing up, each user is stored inside the PostgreSQL “users” table. To store the passwords securely, the password is hashed using BcryptJS.

When logging in, a JSON Web Token (JWT) is generated when the user service signs the JWT payload consisting of the userId, and the timestamp of the signing using a secret key. The JWT is then stored in the client browser’s local storage. Shown in the next page is the sequence diagram for sign up and login.



Sequence diagram for sign up and login

When making a HTTP request to other services, the JWT is included in the Authorization header of the HTTP request. The service receiving the HTTP request will then verify the JWT by forwarding it to the users service. If the JWT is valid and the user that the JWT belongs to has the necessary permissions, the server proceeds to process the request. Else, it will respond with an error and reject the request.

In addition to authenticating API calls, JWTs also ensure that private routes are only accessible to authenticated users. Users who are not logged in and attempt to access private routes will be redirected to the login page on the frontend. The JWTs are set to expire after 12 hours. This forces periodic re-authentication so that in the event of unauthorized access, an attacker would have a limited time window to misuse the account.

Design Decisions

Aspect	Implementation 1: Access token only	Implementation 2: Access token and refresh token
Authentication Mechanism	Single JWT for authentication	Two JWTs for authentication (Access Token and Refresh Token)
Token lifespan	Access token: Short (e.g. 1 hour)	Access token: Short (e.g. 1 hour) Refresh token: Long (e.g. days or weeks)
Security	Limited control over token revocation	Has ability to revoke refresh tokens for enhanced security control.
Maintenance Effort	Simpler to implement and maintain	Additional complexity in handling token refresh and maintaining authentication server.
User Experience	Requires user to login frequently as access token expires relatively quickly	Better user experience. Refresh token allows for longer-lived sessions as new access tokens can be obtained without requiring the user to log in again.

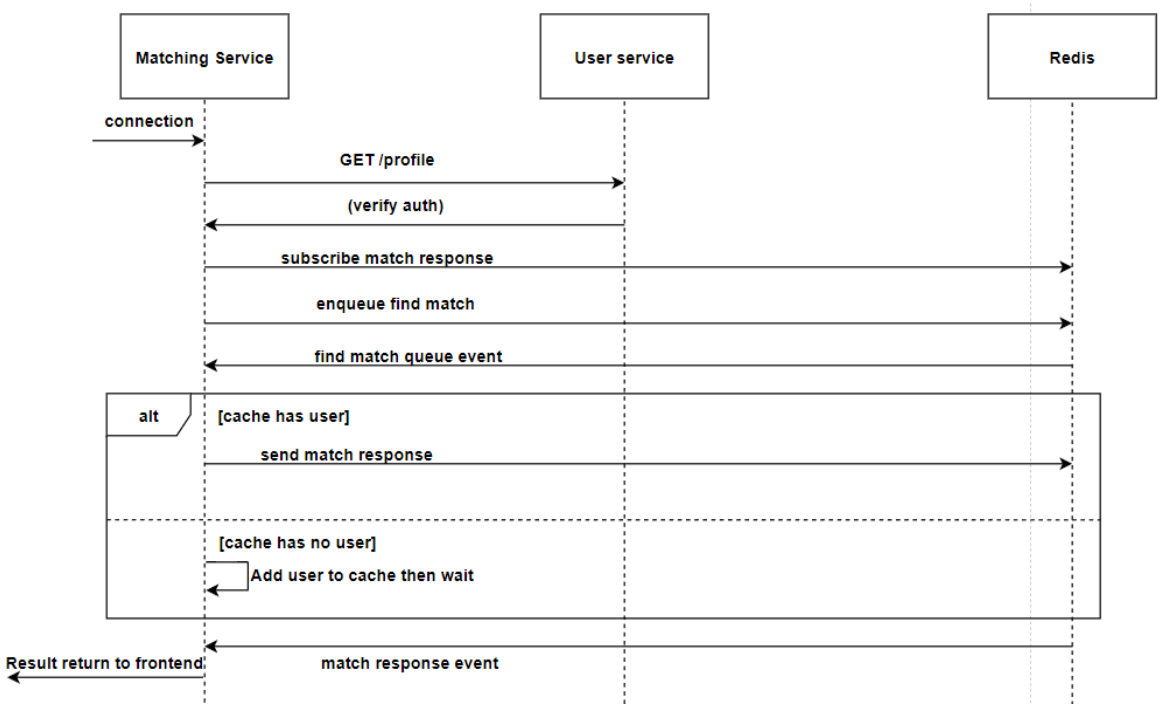
We chose the first implementation as it is faster to implement and maintain. This allows us to allocate more time to develop other critical services and functionalities. Another reason for this decision was due to the lack of the need for security in our platform. Additionally, given that the current nature of our application does not involve handling a lot of sensitive information, a simple access token-only authentication should suffice. As the application evolves and user demands grow, we can potentially incorporate a refresh token mechanism for extended sessions and improved security when the need arises.

3.2.4. Matchmaking Service

The matchmaking service serves as the matchmaker for Ohmygode users when they are looking to collaborate with other users on questions of a specified difficulty level. The main technologies used in the matchmaking service are NodeJS, SocketIO and Redis as our message broker.

The reason redis was chosen over alternatives like RabbitMQ and ActiveMQ was because Redis is an in-memory database. As an in-memory database, Redis provides our application with faster reads and writes, allowing our matchmaking process to be as fast as possible.

Sequence Diagram



Sequence diagram for matchmaking

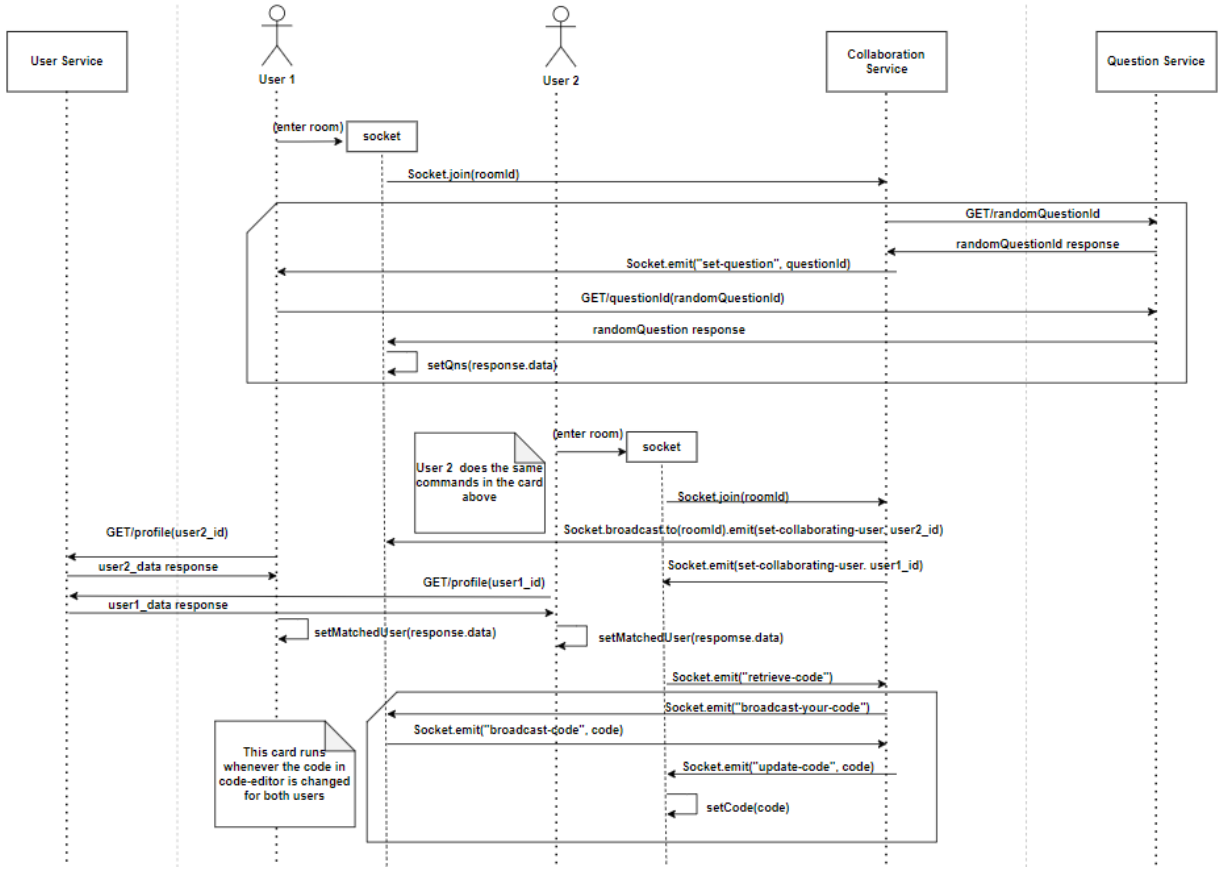
Above is the sequence diagram of the matchmaking process. When the first user requests to find a match to solve a hard question, the user opens a SocketIO connection to the matching service. The matching service then verifies the connection by checking the authentication token provided with the users service. If the token is valid, the matching service will then subscribe to the match responses that are published by the process that consumes our Redis message queue to find matches. After the subscription is done, the matching service adds a message to the find match queue that includes the user's id and the question difficulty.

In the find match queue consumer, the messages are received and one of two things happen. If the local cache already has one other person looking for a match, the consumer will publish a match response event containing the user ids of the two matched users and then clear the cache. Else, the consumer will store the user id of the user looking for a match in the cache. Here, a NodeJS cache is used because at any one time, only a maximum of 3 user ids will be stored in the cache, one for each difficulty. As such, we wanted to use the fastest cache possible. Upon receiving the match response event, the matchmaking service will then return the match found response to the user.

3.2.5. Collaboration Service

The collaboration service serves as the real time communication node for Ohmygode users when they are collaborating with other users on technical interview questions. The main technologies used in the matchmaking service are NodeJS and SocketIO.

Sequence Diagram



Sequence diagram for collaboration

The collaboration is initiated when the user client connects to the collaboration service using a SocketIO connection. The first step that the collaboration service takes is to verify the authentication token of the connection by making a GET request to the users service with the provided token. If authenticated, the collaboration service proceeds to the next steps. Firstly, the socket will join a SocketIO room using the roomId provided.(FR4.1) Next, the service will emit a few events in parallel. Firstly, a random questionId of the chosen difficulty will be retrieved via GET/randomQuestionId request to the question service. The server emits a “set-question” event along with the questionId that was previously retrieved. The user will then retrieve the question data via another GET/questionId request to the question service and render it in the frontend.
At the same time, the server also emits a “set-collaborating-user” event along with the userId. This tells the user to retrieve the matched user data via GET/profile request to the User service and render the matched user profile in the frontend as well. Lastly, the users will emit a “retrieve-code” event which in turn causes the server to emit a “broadcast-your-code” event which tells the user to broadcast their code to the server via a “broadcast-code” event with their current code. The server will then pass the code to the other user via a “update-code” event which will then be rendered in the frontend as well (FR4.6). Whenever a change is made to the code-editor, the user will emit a “broadcast-code” event with their updated code. The server will then pass the code to the other user via a “update-code” event which will then be reflected on their side in real time as well (FR4.2). The server also broadcasts notification events which informs users when the matched user has joined the room or left the room (FR4.5).

Enhancements

The collaboration room not only allows users to format their code according to their preferred language (FR4.7), but also enhances the coding experience by providing automatic syntax highlighting tailored for various programming languages (FR4.8).

Improvements

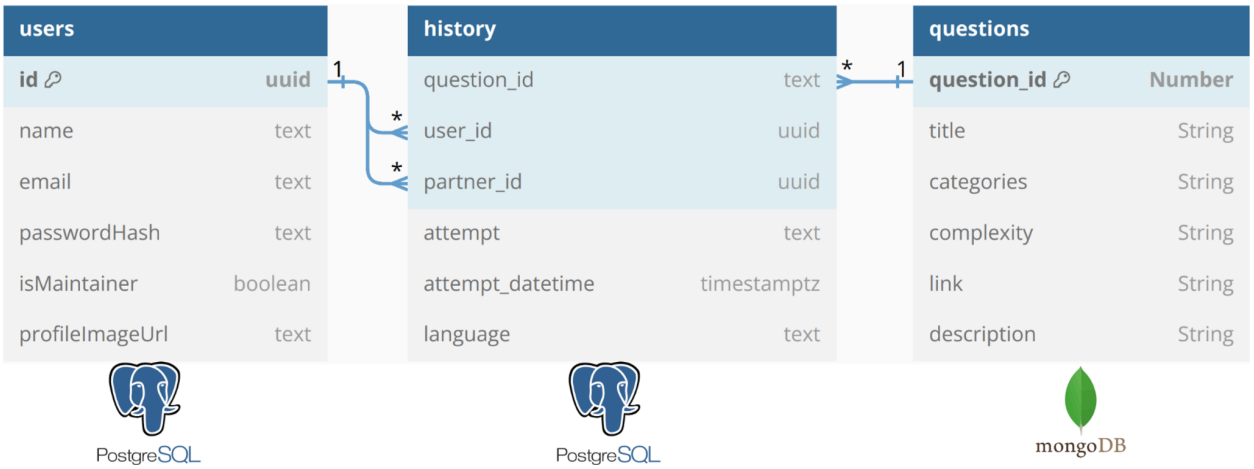
The collaboration service still is a little buggy when writing code simultaneously by multiple individuals. In the context of a technical interview, the interviewer would generally observe and intervene only when necessary. The emphasis is often on the clarity of thought, problem-solving strategies, and effective communication of ideas, which can be evaluated even if only one person is actively writing code at a time. As a result, we decided that it was a low priority bug.

3.2.6. History Service

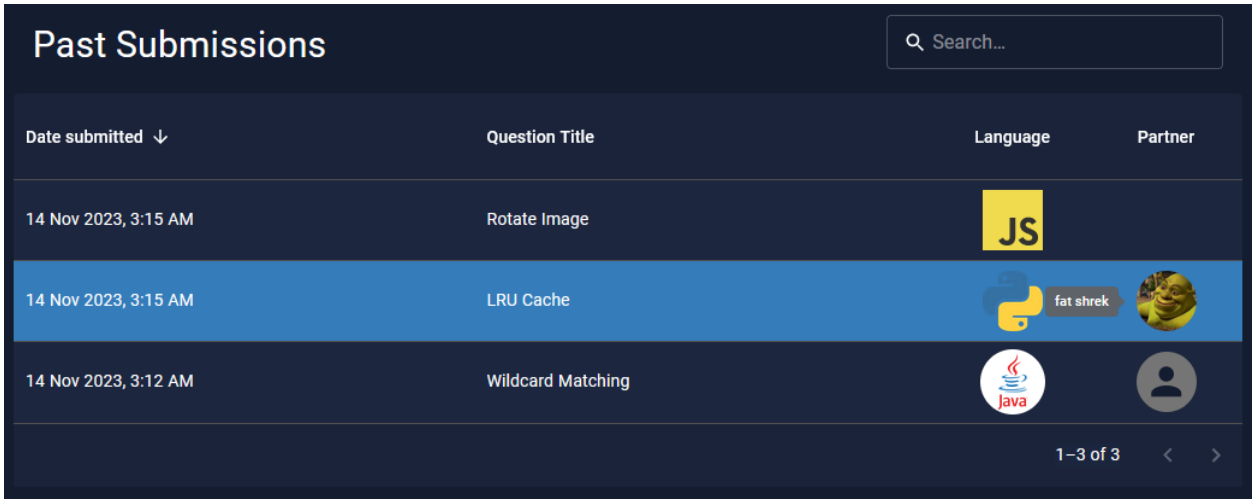
The history service stores the user's past question code attempts. When a user attempts a question with or without collaborating with a partner and saves it, the relavant information like the partner's profile and timestamp will be stored in the database (FR7.2, FR7.3). The code can be retrieved when the user clicks on the history record. (FR7.1)

The user can also sort and search through past attempts based on fields like the question title, time, language, partner name. (FR 7.4)

To display each history row on the frontend, relevant user fields(name, profileImageUrl) and question fields(title) are fetched in real-time through GET HTTP requests to the "users" table and "questions" collection. This ensures that updates to question and partner details will be reflected in the history table frontend.(FR 7.5)



Database schema of Users, History and Questions



History table frontend

Design Decisions

For the implementation, we had 2 approaches in mind:

1. **Data redundancy**: Store the other relevant fields (e.g name, profileImageUrl, title) directly into the “history” table.
2. **Fetch on demand**: For each “History” record, fetch the other relevant fields through GET HTTP requests to the “users” table and “questions” collection

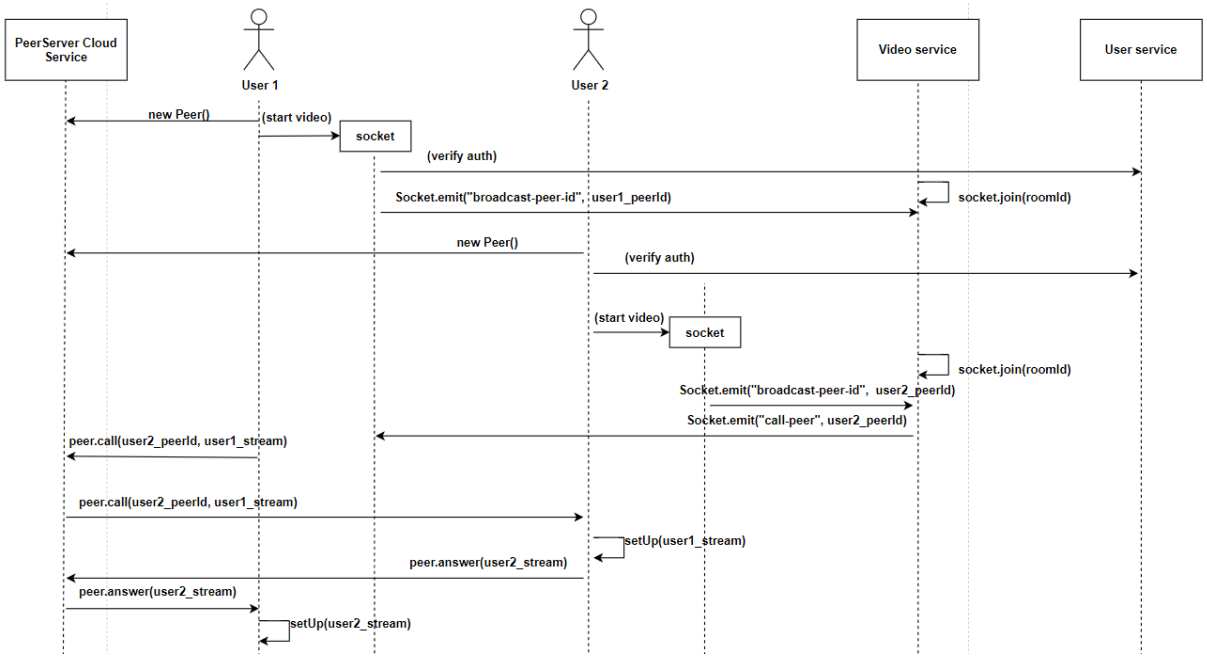
Aspect	Approach 1: Data Redundancy	Approach 2: Fetch on Demand
Data Retrieval	Faster. No need for HTTP requests during retrieval	Slower. Requires multiple HTTP requests to display History records
Storage Space	Increased storage requirements due to redundant data	More efficient storage usage as data is not duplicated
Data Consistency	Higher risk of inconsistent data if updates are not synchronized	No risk of inconsistent data. GET HTTP requests ensures real-time data retrieval from the "users" and "questions"
Maintenance Effort	Updates to user or question details require updates to multiple records in “history” table	No need to update “history table

We decided on the second approach to fetch on demand as the pros outweigh the cons.

3.2.7. Video Service

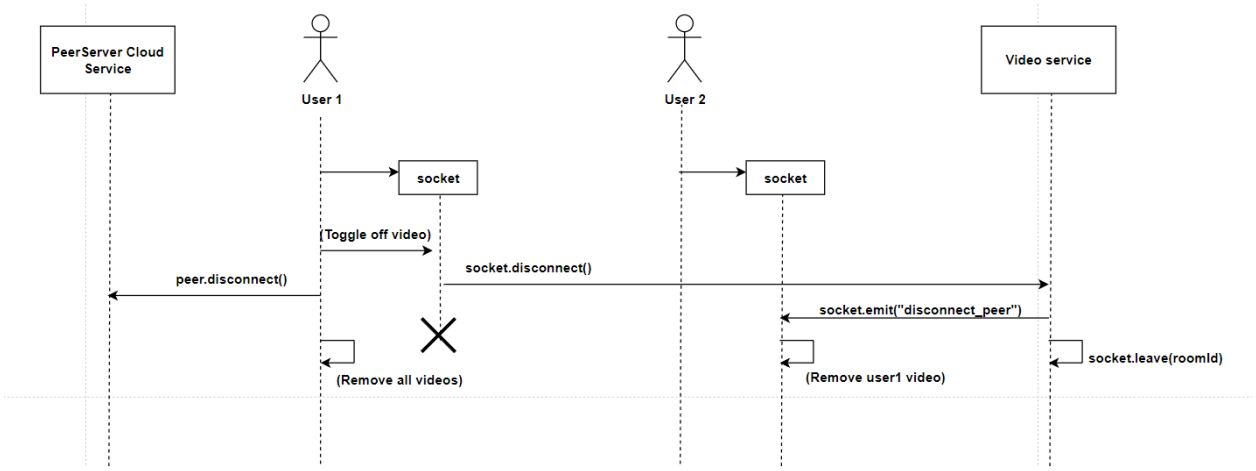
Features

The video service allows users to communicate through a video (FR8.1, FR8.2). Users are able to toggle on and off the video chat feature so as to ensure privacy for users (FR8.3). Toggling on and off the video chat allows users to reconnect back to the call without having to disturb the flow of the collaboration process(FR8.4).



Sequence diagram for toggling on video call

When a user clicks on the video camera icon, it first retrieves and displays its own media stream and creates a peer connection to the peerServer Cloud Service. Next, it creates a socket which connects to the server in the video service. The server first checks for the user's authentication and authorization. The server then makes the socket join a room based on the roomId provided. When the second user clicks on the video cam icon as well, it creates a peer connection to the peerServer Cloud Service as well and their socket will join the same room in the video server. The first user will then be initiated to call the second user with its own media stream via a peerJS server and the second user will answer the call while retrieving the sent stream and sending back its own media stream. After retrieving their matched user’s stream, it will subsequently be rendered as well.



Sequence diagram for toggling off video call

When a user toggles off the video call, it will send a disconnect event to the video service, signaling the other user to remove their matched user's video. The user will also disconnect from the peerServer Cloud Service.

The PeerServer Cloud service is a free cloud-hosted version of the PeerServer. The frontend creates a peer instance and establishes a media stream connection to other peer instances through the server.

Design Decisions

Considerations	WebRTC with PeerJS	Agora
Customization and Control	High flexibility, fine-grained control and uses a separate signaling server	Moderate customizations options and built-in signaling
Scalability	May face challenges with large participant numbers	Seamless scalability with a distributed infrastructure
Cost Considerations	Hosting costs for infrastructure	Usage-based pricing, potential cost savings

We decided to implement WebRTC with PeerJS over Agora for our video chat component. While Agora provides a robust and simplified infrastructure with built-in signaling and advanced features, WebRTC and PeerJS offers a higher level of customization and control and this allows us to tailor the video chat component to our specific project requirements. The challenge of requiring us to set up and manage our signaling server allows us to gain a deeper understanding of the underlying technologies and have fine-grained control over the entire video chat system.

4. Testing

Overview

Testing was conducted through integration tests on the dedicated EC2 instance, chosen for its mirroring of the production environment. Jest framework was used for testing and a CI pipeline was built to run the tests using GitHub Actions.

Design Decisions

Our framework of choice was Jest, selected as an all-encompassing testing solution. Although Mocha and Jasmine were considered as alternatives, Jest emerged as the preferred option due to its comprehensive suite of features, including an integrated test runner, assertion library, mocking capabilities, and support for snapshot testing.

Unlike Mocha, which lacks built-in mocking and assertions libraries, relying on additional tools such as Sinon.js and Chai, Jest streamlines the testing process by bundling these functionalities. Similarly, Jasmine necessitates plugins like jasmine-snapshot for snapshot testing.

The decision to use Jest was reinforced by its simplicity and self-sufficiency, offering an integrated solution tailored to our project's needs. While we didn't leverage all of Jest's functionalities immediately, such as snapshot testing, its flexibility allows for scalability in that direction for future expansions of our testing suite.

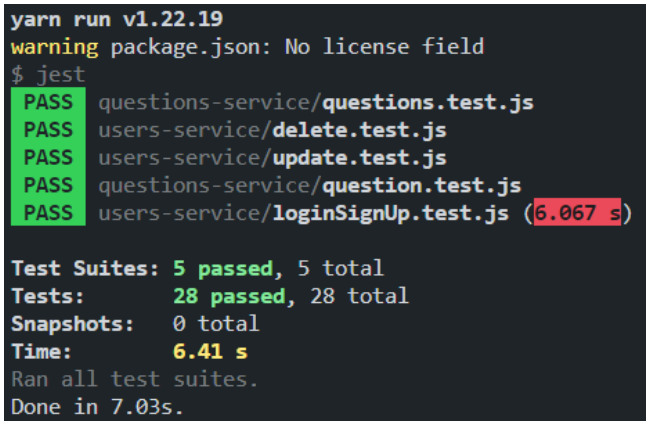
Features

Jest inherently supports asynchronous testing (NFR6.1). Leveraging its built-in capabilities, our tests seamlessly handle asynchronous code, ensuring reliable and efficient execution.

For testing in the development (NFR6.2) and test environment (NFR6.3), we configure the Jest set-up files to accommodate testing in both environments. Testing on the production environment is intentionally avoided, given the existing mirroring of the production environment on the dedicated EC2 instance for testing purposes. This approach not only safeguards the integrity of the production environment but also maintains consistency in testing outcomes across environments.

To guarantee a consistent setup and teardown process for our tests, we implemented specialized helper functions. These functions handle the creation of a new user, including the signup and login processes, before the commencement of each test suite. Subsequently,

post-test execution, the same functions efficiently delete the user, ensuring a clean and isolated testing environment. To enhance the uniqueness of users created for each test suite, we integrated the ‘uuid’ package. This ensures the generation of unique identifiers, preventing any potential data conflicts and contributing to the reliability and accuracy of our test scenarios.



Screenshot of tests passing

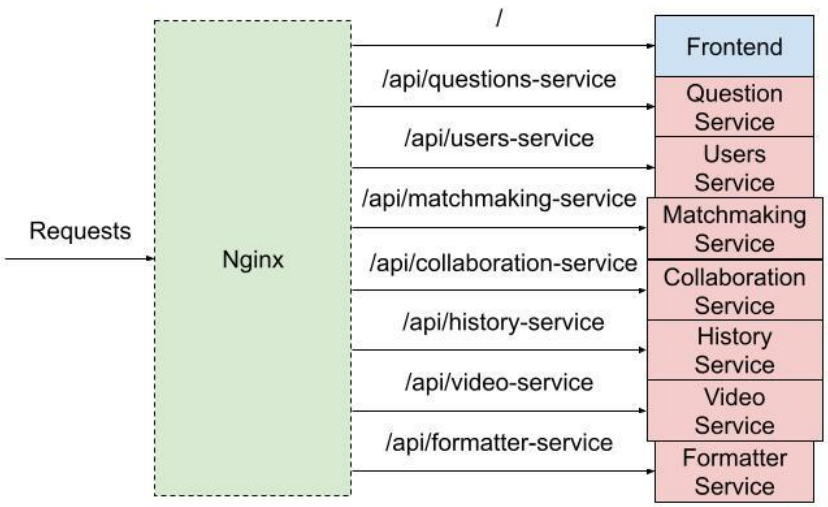
The tests are separated based on functions to ensure they reduce coupling and directly call the APIs which allows our test to run in less than 15 seconds (NFR6.4)

In our commitment to enhancing testing processes and maintaining consistent code quality, we seamlessly integrated Continuous Integration (CI) into our deployment workflow. Utilizing GitHub Actions as our CI platform of choice, our test suite is initiated manually on the GitHub runner following each deployment to our EC2 instance. This deliberate decision not to automate testing after every commit is strategic—it strikes a balance between ensuring robust testing and facilitating a rapid development pace. By coupling CI with pull request (PR) reviews and manual local testing, we maintain a flexible yet effective approach that aligns with our goal of achieving both reliability and agility in our development practices.

5. Deployment

In this section, we will go into detail about the architecture of the production environment. The production environment of Ohmygode can be viewed here.

5.1. NginX



Nginx diagram

In this project, we used Nginx as a reverse proxy to redirect requests from the internet to the various microservices accordingly as shown in the diagram above. At the same time, Nginx also helps to secure the communication channel between our server and the client by using HTTPS protocols.

5.2. AWS EC2 (T2.Micro)

Our group decided to use one instance of the Amazon Web Services (AWS) Elastic Compute 2 (EC2) to host our application. To do this, we created an instance of the T2.Micro EC2 machine and created a private key pair for our GitHub action to SSH into the EC2 and run the production version of the docker compose command. The T2.Micro instance was chosen because it was the most cost-effective for our current user base.

One major difference between the development environment and the production environment is the use of the build artifacts from react. Since the react development server had a lot of unnecessary overhead, we should not be using it in the production server. As such, in the deployment workflow, we will build the react app, and then serve the resultant static files in our EC2.

6. Improvements

6.1. Security

To make the application more secure, switching the JWTs frontend storage from browser local storage to cookies would be key in making the system less vulnerable to XSS attacks as the application's features increase and the user base grows. This is because cookies as compared to local storage are not accessible via javascript.

Additionally, our application uses a single JWT for authorisation. A potential improvement to increase the security of the application will be the use of two JWTs for access and refresh tokens. Access tokens are short-lived and used for regular API requests, while refresh tokens have a longer lifespan and are used to obtain new access tokens. If necessary, refresh tokens can be revoked, invalidating both the current access token and the refresh token. This provides an additional layer of security control.

6.2. Testing

To bolster our testing strategy, we plan to implement several key improvements. Firstly, we aim to increase the coverage of integration testing, ensuring that various components work seamlessly together. Additionally, introducing unit testing will enhance the granularity of our tests, allowing us to validate individual functions in isolation. Jest's powerful snapshot testing capabilities can also be used for frontend testing.

Recognizing Jest's capacity to generate detailed code coverage reports, we aspire to enhance our CI workflow. In the future, we aim to incorporate testing for every pull request, leveraging Jest's unit testing capabilities to prevent new code from inadvertently breaking existing functionality.

6.3. Code Execution

To enhance our users' experience, we plan to introduce a code execution service that facilitates testing and validation of their attempted solutions for correctness. This feature is designed to assist users in honing their coding skills through the practice of debugging, a crucial aspect since interviewers often assess how candidates approach and solve problems.

7. Contributions

Sub Group	Nice-to-haves	Member	Technical	Non-technical
Group 1	N8, N9, N11	Loh Xian Ze, Bryan	- Worked on frontend pages - Implemented question service - Implemented users service - Implemented matchmaking service - Implemented collaboration service - Deployed application to AWS EC2 - Configured Nginx in AWS EC2	Contributed equally to the project report and presentation slides
		Gowri Lakshmikanth Bhat	- Worked on frontend pages - Worked on some APIs for users service - Implemented CI workflow - Deployed application to AWS EC2 for testing - Configured Nginx in AWS EC2 for testing - Implemented testing	
Group 2	N1, N2, N5	Wong Yew Jon	- Worked on frontend pages. - Worked on collaboration service - Implemented Video service - Implemented syntax highlighting	
		Gerald Teo Jin Wei	- Worked on frontend pages - Worked on some APIs for users service - Implemented user profile management - Implemented History service - Implemented code formatting	

8. Reflections and learning points

Project Management
Through this project, we realized the importance of implementing a Scrum process in our development workflow. We did one-week sprints, each encompassing a comprehensive cycle of planning, design, development, testing, and thorough review, ensuring that everyone was on the same page and was clear on their assigned tasks. This approach allows us to work in small increments and allows us to react effectively to changes in requirements, specifications, and deadlines. Additionally, listing the functional and non-functional requirements at the start helped us to plan better. It allowed us to estimate the time and effort needed to implement each requirement and focus on what microservices and functionalities we needed to prioritize for our project.

Exposure to different technologies
This project motivated us to pick up many new technologies. We had to do a lot of self-learning by reading up on documentation and experimenting with frameworks and technologies like React, PostgreSQL, MongoDB, Mongoose, Socket.io and many others. Some of us have little to no prior experience in web development which made the initial stage of development very difficult. When choosing our technology stack, our team made a decision only after considering multiple factors like the team's familiarity and expertise in the technology, popularity, quality of documentation, security, etc

In conclusion, this project helped us to learn more about software engineering practices, familiarize ourselves with both frontend and backend development and gain valuable experience in working as a team.

9. References

Crash Article in Agile Development | by Nick Ivanecky. (2016, Sep 6). *Medium*.
Retrieved November 15, 2023, from
https://medium.com/open-product-management/crash-article-in-agile-development-da960861259e