CS3219 Final Report
Code2Gather
Team 32

He Xinyue (A0204712U)
Wang Luo (A0180092L)
Wen Junhua (A0196683L)
Zhu Hanming (A0196737L)

1. Introduction

At many tech companies today, algorithm and coding problems form the largest component of the interview process. Called technical interviews, these problem-solving sessions allow for interviewers to evaluate candidates' abilities to solve algorithmic problems.

However, unlike solving problems on your own, these technical interviews expect candidates to be able to not just come up with solutions, but also clearly articulate their thought process and engage the interviewer. This makes technical interviews very difficult to prepare for alone, and finding peers to practice with, as well as coordinating mock interviews, can be an arduous process. This makes proper technical interview preparation very inaccessible to most individuals.

The goal is thus to build a software that helps users to prepare for technical interviews, by matching them with a peer and providing them with questions to practice with. It will simulate the environment of actual online technical interviews, allowing students to build up confidence before a technical interview.

The software was built by a team of students from the National University of Singapore (NUS), as part of the module CS3219: Software Engineering Principles and Patterns, and is aptly named Code2Gather. This document will thus go into both the requirements of this project and the development of Code2Gather.

The working application can be found at https://code2gather.io.

The repository for this project can be found here: https://github.com/CS3219-SE-Principles-and-Patterns/cs3219-project-ay2122-2122-s1-g32

1.1. Glossary

This document uses the following terminologies and conventions:

Terminology	Meaning
DB	Database.
Mock Interview	A practice session that simulates the real technical interview, where two people will take turns to be the interviewer and interviewee.
Pair(ing/ed), Partners	Two users who are paired will do the mock interview together.
Room	A session or virtual room for the two paired users to practice mock interviews in.

1.2. Intended Audience and Reading Suggestions

This document is intended for developers, designers and project managers who are keen to implement a solution to target the aforementioned problem, as well as graders and the teaching team of CS3219.

1.3. Project Scope

This software is a web-based application that can be hosted on the Internet. It also provides a clean and user-friendly interface to the users and enables users to find others to practice technical interviews with and run through a realistic technical interview together. It should be similar to how job and internship applicants are doing mock interviews today, where matched users will take turns to play the roles of the interviewer and interviewee.

This software is targeted towards anyone who is interested in securing an internship or job and wishes to practice before their interviews. We aim to provide users with an accessible and enjoyable experience, the former of which is also why the application will be a web-based application that does not require any intensive installation or set-up.

1.4. References

- The Hardest Part About Microservices: Your Data
- Microservice Architecture pattern
- Security Risks in Online Coding Platforms

2. Overall Description

This section will provide a blackbox view of the software that we are specifying in this document. This is also the product perspective that Code2Gather was developed with.

2.1. Software Objectives

These are effectively high-level requirements that are implementation-agnostic.

- Allows users to easily find people to practice technical interviews with.
- Allows users to practice questions of specific difficulties while communicating with the interviewers.
- Allows users to keep track of their progress and how they performed.

More will be fleshed out under the Requirements section.

2.2. Product Perspective

For this section, we will refer to the software as the product. This section will provide a blackbox overview of the product, and touch upon non-technical perspectives of this product.

2.2.1. Product Overview, Context and Scope

(The context was covered earlier under Introduction.)

The objective of this product is to provide a platform for users to practice technical interviews. The interview experience should be as close to that of a real technical interview as possible, albeit with slightly more control for the users, such as having the option to choose their preferred question difficulty.

This product is relatively standalone, though integration with other services can and will be considered, such as external authentication services like Firebase and GitHub authentication.

2.2.2. Stakeholders and Stakeholder Analysis

First, we have the direct stakeholder - the user. As technical interviews are conducted in the industry across all levels, even for senior engineer roles, the product expects that users can range from very young to very old. We are thus looking at individuals who wish to procure a job in the technology industry as the target user.

These users can range from inexperienced to very experienced. As such, the product will cater to users with different levels of experience and expertise, such as providing a comprehensive onboarding process.

Furthermore, this product will be easy to access and allows users to start practicing easily, given that the problem we are trying to resolve is the lack of accessibility and the inconvenience of setting up mock interviews.

We must also consider the possibility of trolls or malicious users, who are out there to sabotage the interview preparation for other users. As such, this product will have a rating system, where users can rate each other for their performance as a practice partner (and not in terms of their coding ability). The product will then match users with similar ratings, so as to encourage good behaviour and slowly isolate users who degrade others' experiences.

Lastly, given that preparing for technical interviews is not an overnight process, this product can also help in providing actionable data and statistics to the users, such as letting them know how they have performed in the past.

Second, one possible group of indirect stakeholders would be the tech companies and interviewers/recruiters. They often have to keep up with the trends in this interview preparation industry, since they need to ensure that their hiring process can correctly measure the suitability of the candidates. Products that help candidates "cheat" through the process would thus reduce the reliability of their existing practices.

The product thus ensures that the interests of such stakeholders are not compromised, as we will not be providing any information or data that will unfairly help potential candidates. For example, the questions come from publicly available sources, and will not be from any specific company's interview question bank. The product will also be geared towards practice and self-improvement.

In the future, we can potentially more actively engage this second set of stakeholders, such as allowing them to use the platform to measure the capabilities of candidates. However, this is not a priority for now, given that there are a lot of existing players in this market, such as HackerRank, CodeSignal and more.

Another possible group of stakeholders is educational institutions and career offices, who may see this application as a way to help their students and counselees receive the interview preparation they need.

2.2.3. Usage Process

The main usage process (or the happy path) will be as follows:

1. A user who wishes to practice for their technical interview logs into the application using e.g. GitHub login.
2. After logging in, the user can select a question difficulty level and begin searching for another user to practice with.
3. The user then waits until they get paired with another user who has selected the same question difficulty level, and they will enter a room together, where they will begin a video call automatically with one another. Both of the users will also have access to a collaborative code editor.
4. One of the two users will be randomly selected as the interviewer, who will be able to see the question and the solutions. The other user will be the interviewee, who has no access to the question. It is up to the interviewer to decide how they wish to communicate the question to the interviewee.
5. The interviewee can select the language they prefer (out of a set of languages that the platform supports), and then attempt the question. The code that the interviewee writes can be executed, which allows the interviewee to test the code that they have written.
6. Once the interviewer is happy with the code OR if the interviewee gets stuck and can no longer proceed, they can talk through the solutions and mark the interviewee's attempt as solved or unsolved. The roles will then switch, and steps 4-6 will repeat with the previous interviewer as the new interviewee and vice versa. The difference is, that upon the second interviewer marking the second interviewee's attempt as solved or unsolved, the room will close.
7. Users can then rate each other from 1-5, based on the other user's performance as an interviewer.
8. Users can then review their attempt under their attempt history, as well as see updated statistics based on all their past attempts.

3. Individual Contributions

The following table summarises the technical and non-technical contributions of the members involved in the project.

Name	Technical Contributions	Non-Technical Contributions
He Xinyue	Implement API Gateway Implement Authentication MS Implement Room Management MS	Requirements documentation Project final report
Wang Luo	Implement Frontend Implement Video Call MS DevOps	Requirements documentation Project final report
Wen Junhua	Implement Coding Service and Code Execution Service DevOps	Requirements documentation Project final report
Zhu Hanming	Implement Frontend Implement Pairing MS Implement User History MS Set up CI/CD DevOps	Application Design Requirements documentation Project final report

4. Requirements

4.1. Functional Requirements

The below functional requirements are categorised based on the different parts of the usage process.

4.1.1. Authentication

ID	Functional Requirement	Priority
A-1	The system will allow the user to log in.	High
A-2	The system will allow the user to create an account.	High
A-3	The system will allow the user to log out.	High

Code2Gather uses GitHub authentication, since GitHub is a commonly used tool by developers and coders today, and many actually prefer how GitHub authentication helps to centralise their accounts across multiple platforms.

4.1.2. Pairing

ID	Functional Requirement	Priority
P-1	The system will allow users to indicate their preferred question difficulty.	High
P-2	The system will pair users who prefer the same question difficulty.	High
P-3	The system will pair users with similar ratings.	Medium
P-4	The system will allow users to cancel pairing before a pair is found.	High
P-5	The system will cancel the pairing for a user if the user does not get paired after 30 seconds.	High

As mentioned in the Product Perspective above, the software aims to encourage good behaviour, which is why the pairing is done between users of similar ratings (P-3).

As for pairing based on the question difficulties (P-2), this is a requirement that will be tested and reviewed subsequently. Since the software allows for users to receive separate questions, it is entirely possible for users who want different question difficulties to be matched. However, there is a possibility that users who aim to practice easier questions may be unable to do well as an interviewer for a question that is harder,

e.g. unable to understand the problem or solutions. As such, this requirement will require further analysis.

4.1.3. Room Management

ID	Functional Requirement	Priority
RM-1	The system will add two paired users into a room.	High
RM-2	The system will randomly assign a user the role of an interviewer first.	High
RM-3	The system will allow the user to leave a room.	High
RM-4	The system will allow the user to leave the room if he is left alone in the room for over 30 seconds.	Medium
RM-5	The system will display the question to the interviewer.	High
RM-6	The system will show the interviewer some pointers to the question.	Low
RM-7	The system will allow the interviewer to type down feedback for the interviewee.	Medium
RM-8	The system will not allow the interviewee to view the question.	High
RM-9	The system will allow the interviewer to mark the interviewee's attempt as solved or unsolved.	High
RM-10	The system will allow the user to rate his or her peer at the end of the interview.	High

4.1.4. Video and Audio Communication

ID	Functional Requirement	Priority
VC-1	The system will allow users to join a video call.	High
VC-2	The system will allow users to mute their microphones.	Medium
VC-3	The system will allow the users to turn off their cameras.	Medium

4.1.5. Coding Process

ID	Functional Requirement	Priority
CP-1	The system will allow the two users in the same room to work in the code editor together, i.e. they can both type text, and edit and delete each others' text in this code editor.	High

CP-2	The system will allow the interviewee to select a programming language that they would like to use.	High
CP-3	The system will highlight the syntax based on the language selected.	Medium
CP-4	The system will allow both users in the room to execute the code and see the output.	Medium
CP-5	The system will recognise frequently used keyboard shortcuts.	Low
CP-6	The system will allow users to edit code collaboratively in real-time.	High

The experience that we are going for is much like Coderpad, where users can easily collaborate on code and discuss from there. Since Coderpad is an interview tool used in the industry currently, being similar to it will be beneficial for the users, as they will get to practice with a similar experience and environment as they would use during an actual interview.

Also, do note that the collaborative requirement in CP-6 means that convergence is necessary, i.e. we cannot have a case where two users' local records are permanently diverged, or a case where one user's actions overwrites another user's, if performed concurrently.

4.1.6. User History

ID	**Functional Requirement**	**Priority**
UH-1	The system will track the questions completed by the students.	Medium
UH-2	The system will display a public leaderboard based on the number of questions completed by all users.	Low
UH-3	The system will allow the user to view their history, which includes the code they wrote, the interviewer feedback and whether they successfully solved it.	Medium
UH-4	The system will track and compute the rating of each user.	High

4.2. Stretch Goals (FRs)

There are some functional requirements that would be good to meet, though they are not necessary for the minimum viable product. The priorities for these requirements are defined assuming that we will be trying for these requirements.

4.2.1. Collaborative Drawing

ID	Functional Requirement	Priority
CD-1	The system will allow users to draw diagrams to illustrate their points visually.	High
CD-2	The system will allow users to see the diagrams that the other user in the room is drawing.	High
CD-3	The system will allow users to edit and clear all diagrams being drawn.	High

As some ideas may be difficult to communicate verbally or via text, allowing the users to draw, e.g. in a separate window or panel on the screen, would allow them to communicate much more effectively. An example would be questions that ask about trees and tree traversal.

This was not achieved by Code2Gather.

4.2.2. Chat

ID	Functional Requirement	Priority
CH-1	The system will allow users to communicate via a chat system.	High
CH-2	The system will delete chats after the room is closed.	Medium

There may be instances where the users in the room are unable to communicate via video and audio, be it due to background noise, or due to poor Internet connection. In such cases, a chat will be useful.

This was not achieved by Code2Gather.

4.2.3. Playground

ID	Functional Requirement	Priority
PL-1	The system will allow users to create a single-user room just for experimentation purposes.	High

PL-2	The single-user room will support collaborative code editing and no other functionalities.	High

The playground will allow users to be able to familiarise themselves with the software stress-free, since they can take their time to experiment with the application.

This was achieved by Code2Gather. A guest playground is available for the unauthenticated user, which allows them to code collaboratively with their peers (no limit on number of participants). However, features such as video call, code execution and questions will remain locked to these users.

4.3. Constraints (NFRs)

Below are some constraints for this software. Note that this list is not exhaustive, and it may have some overlap with NFRs that will be covered subsequently.

ID	Non Functional Requirement	Priority
NFR-1	The client application will need to work on the following modern browsers: Chrome, Safari, Firefox, Edge, IE 11.	High
NFR-2	The system should be secure against all forms of user input and code execution.	High
NFR-3	The collaborative experience should be real-time and smooth.	High
NFR-4	The code written by two users in the same session should always end up in the same final state, regardless of the Internet connections of the users.	High
NFR-5	The system should discourage malicious/unconstructive behaviour from users.	Medium

4.4. Quality Attributes (NFRs)

4.4.1. Security

The first quality attribute that this software must have is security. Since the code editor is collaborative and code execution is supported on this software, in addition to how there may be multiple moving parts in the software, there are a few places where security attacks can take place.

ID	Potential Threat	Possible Mitigation
SEC-1	Malicious user input being sent via the code editor, which may affect	User input sanitation to be performed.

	both the server and the other user in the room.	
SEC-2	Execution of malicious code through the software.	User code is to be executed within a sandbox. Execution is to be terminated after a set time limit, and the instance will be stopped. Memory usage on the instance will also be strictly limited.
SEC-3	Crafting of malicious packets to be sent via the video call data transmission.	Use established communication protocols for video calls, e.g. Real-time Protocol (RTP), RTP Control Protocol (RTCP).
SEC-4	Man-in-the-middle attacks on the packets being sent to and fro.	Use of HTTPS protocol for all transmissions.
SEC-5	(Distributed) Denial of Service attacks on the servers.	To use the correct configuration and hosting infrastructure. Also, some form of rate-limiting to prevent a single IP address from sending too many requests, e.g. via NGINX.
SEC-6	Data leaks from data storage.	All data storage will be secured with appropriate credentials and IP access controls.

4.4.2. Performance

Since the software should simulate a live coding experience like that in a technical interview, the performance of the software is key.

There are a few aspects that we will want to target. All the performance targets below assume a strong Internet connection. A good reference would be the WiFi at the National University of Singapore.

ID	**Operation**	**Targeted Performance**
PER-1	Video call communication.	Less than 2 seconds of delay between sending and receiving of video input. 5 seconds under degraded performance.
PER-2	Collaborative code editing.	Less than 1 second of delay between sending and receiving of changes made to code on the code editor.

		5 seconds under degraded performance.
PER-3	Code execution.	For simple code, e.g. just a print statement, the result should be visible within 10 seconds. 30 seconds under degraded performance.
PER-4	Page loading.	Navigation between different client-side pages should take less than 5 seconds. 15 seconds under degraded performance.
PER-5	Authentication.	Logging in and registering should take less than 10 seconds 30 seconds under degraded performance.
PER-6	Dynamic capacity.	The software should be able to handle up to 100 concurrent users before experiencing performance issues. This number may be refined subsequently.

The number of **100 users** is rather arbitrary at this point, but it gives teams an approximate expectation to work towards. All subsequent numbers are thus relative to this number of 100 users. For teams that wish to go beyond, do scale the later numbers accordingly.

4.4.3. Usability

Since the software aims to make the technical interview practice process easier and more accessible to a myriad of users, who may have different backgrounds and be from different demographics, the software should try to be as user-friendly as possible. The aim is to balance usability optimally for the whole spectrum of users, not just a single community.

ID	**Aspect of User Experience**	**Proposed Solution**
USA-1	User is new to the software.	Onboarding process. Help screens and tooltips. Similarity to other systems used for technical interviews.
USA-2	User is new to technical interviews.	Walkthrough of technical interviews available on the website.

		Example videos of how to use the software.
USA-3	User-friendliness for general operations.	Verbose prompts. Meaningful, plain-language messages. A limited number of options and widgets are displayed.
USA-4	User-friendliness while coding.	Keyboard shortcuts. Syntax highlighting. Line numbers.
USA-5	User-friendliness when authenticating.	Integration with GitHub accounts for authentication.

4.4.4. Others

There are some other quality attributes that would be good to discuss but are not the primary focus of this document.

- **Reusability:** Reusability is more of an internal quality, where software components can be reused within this project or even for other projects! It would be good if the functionalities related to video, code editing and code execution are reusable since these are all usable for other projects.

- **Robustness:** Robustness, or *tankness*, is the degree to which a system continues to function properly despite invalid inputs, defects in software or hardware components, or unexpected conditions. Since the software will be executing code written by users, this quality is of relative importance.

- **Scalability:** This software is one that can potentially capitalise on the network effect, i.e. the more users it has, the more users it will have. As such, scalability is of importance, as growth in users is likely to occur over time. There may even be a need to scale across different geographical locations. As such, it would be good if the software is designed with scalability in mind.

4.5. External Interface Requirements (NFRs)

External interface requirements specify hardware, software, or database elements with which a system or component must interface. This section provides information to ensure that the system will communicate properly with external components.

4.5.1. User Interfaces

In this section, we will discuss three types of requirements for user interfaces:

- Interfacing required with the user's machine
- Interfacing required with other external services/interfaces
- Interfacing required with the user themself

For the first point, the web application will be interfacing with the user's browser. As such, it will need to work with existing browser APIs. The web application should ideally work with different resolutions, but at the minimum, support resolutions that are minimally tablet size. The web application can also process and make use of keyboard shortcuts.

For the second, there are a few services that the software may depend on. Since it's a web application, it may be dependent on certain CDNs and APIs such as Google Fonts and Font Awesome, which enable the web application to have a beautiful user interface.

For the third, the web application will be compliant with the W3C Accessibility Standards, as well as The A11y Project Checklist. Messages and content on the web application will also be understandable by the general population, to increase the ease and accessibility of usage.

Mockups can be found subsequently, which further concretise the requirements for user interfaces. Code2Gather was developed with reference to these mockups.

4.5.2. Hardware/Communications Interfaces

The web application will need to run on a browser, and one that's running on a laptop or a desktop. It would be ideal to also support mobile usage. Code2Gather has been designed to work on both desktop and mobile.

As such, there will be interfacing with the web browser, and between the web browser and the necessary servers. The latter interfacing will be using protocols such as HTTPS and WebSockets for secure communication. For video communication, there will also be the use of relevant real-time conferencing protocols, such as RTP and RTCP.

4.5.3. Software Interfaces

The software may also require external services, such as for authentication and for code execution. Between the client application and the server, interfacing will also be required, through HTTPS requests or WebSockets events. For video communication, there will also be the use of relevant real-time conferencing protocols, such as RTP and RTCP.

Furthermore, there will be a need for data persistence, since the software will need to keep track of user history and statistics. For that, there will be a need for a database layer, and hence a database interface.

4.6. System Requirements (NFRs)

The client application will need to be able to work on all modern browsers, namely Chrome, Safari, Firefox, Edge and IE 11. As for operating systems (OS), features such as keyboard shortcuts should work for most OSes and OS-specific keyboard configurations. The OSes that we will support are Mac, Windows and Linux.

As for the server application that the client depends on, it should work on common server infrastructures and operating systems. It should be able to be deployed using various hosting infrastructures.

4.7. Data Requirements (NFRs)

As this software relies on interview questions to facilitate the interview practice, it will be unable to support functionalities that require data beyond what is generally accessible and available. For questions, this generally includes:

- Question description
- Question difficulty
- Question constraints and pointers
- Question solutions
- Question test cases

Providing information beyond these may potentially damage the interests of indirect stakeholders, which are the tech companies and recruiters.

The quantity and availability of this data also bound the growth of the software. If there are insufficient questions in the question bank, users may stop using the app as intended once they begin to see repeat questions, e.g. they may come up with their own interview questions to ask, which is not the intended user flow for this software.

At the current moment, Code2Gather is developed with a small set of questions for testing purposes. It is, however, easy to extend the current pool of questions.

4.8. User Requirements

The below are some rough user stories that may be relevant. This is not the focus of this document, however.

As a...	**I want to...**	**so that...**	**Priority**
Tech Student	Practice mock interviews	I have a higher chance of securing an internship	High
Unemployed IT	Practice mock interviews	I have a higher chance of	High

professional		securing a full-time job	
Recruiter at a Software engineering company	Practice mock interviews	I can conduct interviews better	Low
Interviewer	See the code of my interviewee near real-time	I can follow their train of thought and evaluate them	High
Interviewer	View the interview question	I can conduct my interview	High
Interviewer/ Interviewee	Execute the code	We can check for the correctness of code easily	Medium
Interviewer/ Interviewee	Speak to each other	We can conduct the interview	High
User	See my question history	I can review the questions I have done	Medium
User	Choose my difficulty	I can do the questions appropriate for my level	Medium
User	Match with other users	I can have someone to practice with	High
User	Secure my account	Others cannot impersonate me	Low
User	Keep my information private	Others do not have access to my information	Low
Guest	Practice mock interviews without logging in	I do not have to go through the hassle of creating an account	Low

5. System Architecture

5.1. Microservice Architecture

This document recommends a Microservice Architecture for implementing the software specified above, for the following reasons:

- The functionalities required are relatively modular, and there is not much need to couple them together. For example, the video call functionality can run independently of the collaborative code editing functionality.
- A Microservice Architecture allows us to break a large application into loosely coupled modules that communicate through APIs. Each of these modules can then be individually scaled based on usage, allowing for greater adaptability and flexibility.
- The Microservice Architecture also allows separate developers to develop and test each microservice without any dependencies on another. This speeds up development and iteration.

5.2. Architecture Diagram

Here is the architecture diagram for Code2Gather:

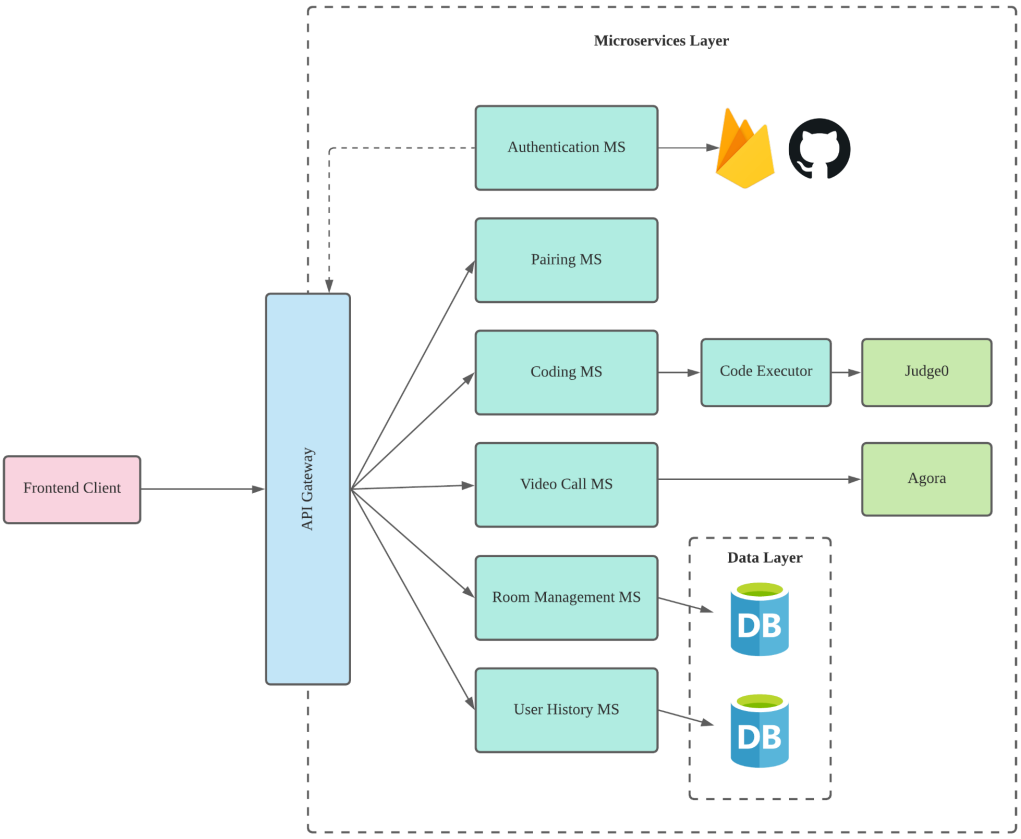

Figure 5.2a: Preliminary Architecture Diagram

In the architecture above, the team has separated the various concerns and functionalities into a few main microservices:

- Authentication service, which will be an adapter for Firebase authentication with GitHub. This allows us to easily switch out Firebase, if e.g. we suddenly change our mind about the underlying authentication infrastructure.
    - This authentication service will be utilised largely by the API gateway to perform authentication on all incoming requests, so that the user can be verified and "resolved". It's much like a middleware but at a high level.
- Pairing service, which can be a simple WebSocket server that helps to pair users together. No data persistence is required, since the pairing is done in real-time.
- Coding service, which helps the users manage their code and execute the code when necessary. There are two main parts to this:
    - Collaboration: To resolve changes made by the two users, in order to arrive at a single synchronised state. There are many ways to implement this, such as using Operational Transformation, Conflict-free Replicated Data Types, or Differential Synchronisation.
    - Code execution: To execute the code written. This will require some form of runners, e.g. VMs that are spawned on demand.
- Video call service, which will be an adapter for the Agora SDK. This helps to reduce the coupling from our application with the vendor-specific functionalities, allowing us to easily change the underlying video streaming provider.
    - This service will be used by the API gateway to establish and authenticate audio and video channels between two users, so that the channel remains private and secure.
- Room Management Service, which manages question selection and users' roles for the mock interviews. It also needs to link up with Video Call Service for video call related management and sends meeting data to the User History Service by the end of the mock interview session. There are two main components to this service:
    - Question selection: To randomly select a question of the selected difficulty for each round of the interviews.
    - Role Management: To display different contents for interviewer and interviewee (for example, the interviewee should not have access to the interviewer's notes panel), and to switch users' roles when the first round of interview is completed.
- User History Service, which stores the bulk of the data from the various mock interviews. All past data and statistics will be stored and computed in this service, allowing users to retrieve leaderboard statistics, interview histories, etc., and it will perform data persistence.

All these will be abstracted behind an API gateway, which will handle the routing of

these various requests accordingly. As mentioned above, this gateway will also heavily depend on the Authentication Service as a middleware of sorts.

Lastly, the client web application will communicate with the API gateway to perform the relevant operations.

6. Design and Implementation

This section contains details specific to Code2Gather, as well as some general suggestions and thoughts on how things can be improved.

6.1. Software Features

To summarise the requirements above and reiterate the features required:

- Users are able to log in to the application, i.e. there is authentication.
- Users can select a question difficulty that they want to practice.
- Users can get paired with people to practice technical interviews with.
- Users can communicate with the other user in the room through collaborative code editors and video calls.
- Users are able to execute their code to see if their solution is correct.
- Users are able to take turns to interview one another.
- Users are able to rate the other user at the end of the practice interview.
- Users are able to keep track of their progress and how they performed.

6.2. Technology Stack

This is the technology stack that the NUS team will be working with.

6.2.1. Frontend

1. React
2. Redux

6.2.2. Backend

1. Golang, Node.js and Python
2. PostgreSQL, NoSQL Databases

For the backend, as the team is using a microservices architecture, each individual service can be written using a different language or framework, so long the interface is implemented. The team thus plans to select the appropriate option for each service, depending on the requirements, such as speed and latency.

As for the database layer, the team is considering different options as well for different services. See the next section for more information.

6.3. Design Patterns

6.3.1. MVC Pattern + Delegation Pattern

We utilised the MVC pattern on our frontend extensively to achieve the separation of concerns, though it must be acknowledged that the MVC segregation is rather blurred in React and JSX.

We followed the following structure for all of our routes (pages): we have a base component, which is stateful, that acts as a controller for that page and handles the rendering of smaller stateless components. These stateless components effectively form the views of the application.

These base components then delegate the work of communicating with the backend to respective REST API and socket services. These services help bridge the frontend with the backend.

Subsequently, all information retrieved from the backend will be stored into the Redux store, which the controllers are observing and passing into the views. As such, the Redux store forms our models, and will cause the views to re-render based on changes to the models.

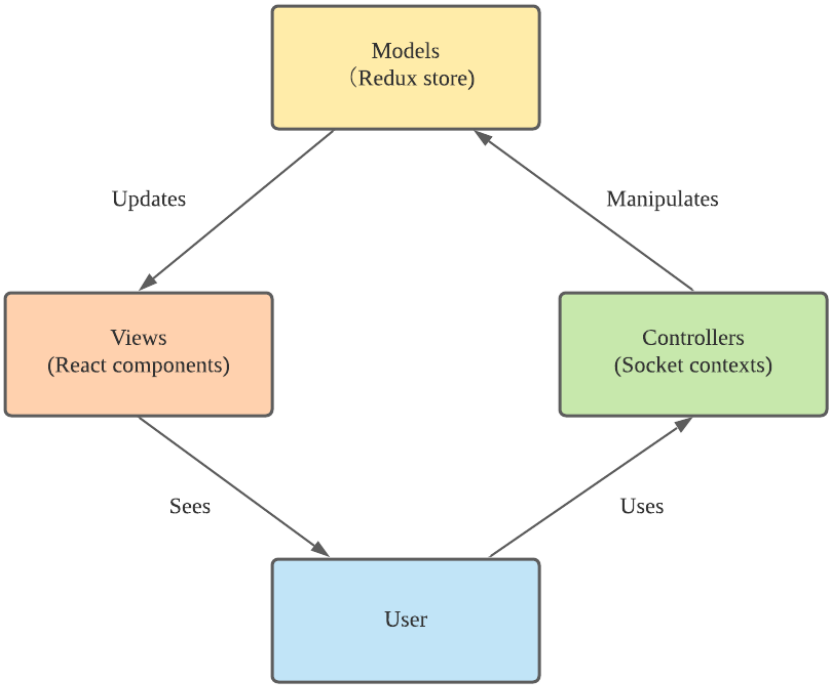

Figure 6.3.1a: MVC pattern

6.3.2. Pub-Sub Pattern

Websockets are extensively used in Code2Gather. For instance, our coding microservice and room microservice utilise websockets to send and receive real-time updates to and from the frontend. This is an implementation of the Pub-Sub pattern whereby the pairing, coding and room microservices publish changes to the observers (which are the frontend socket clients). With this pattern, the user will be able to see their partner's activities in real-time.

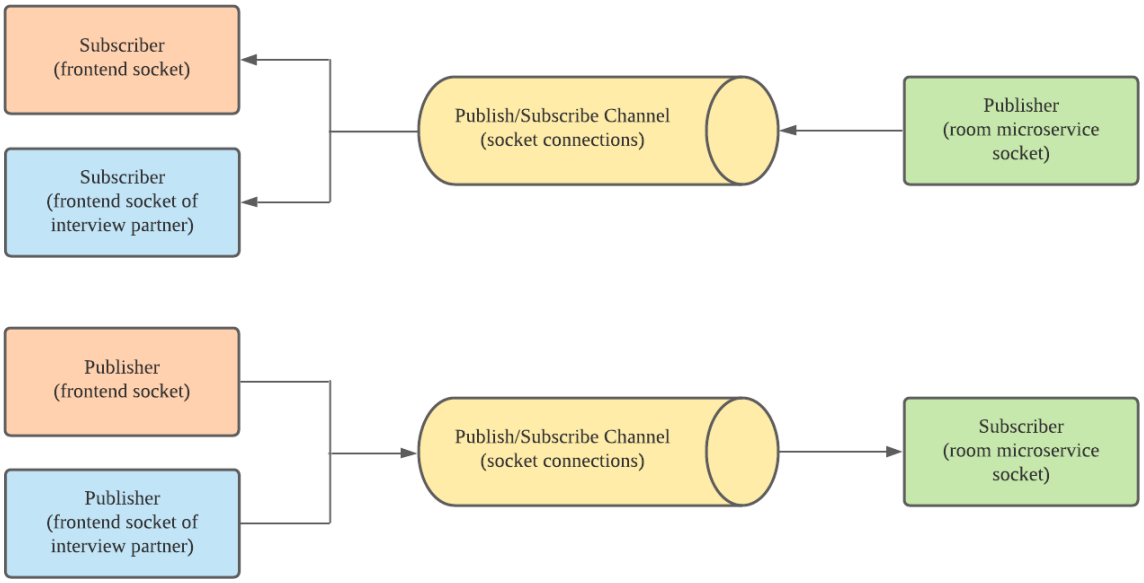

Figure 6.3.2a: Pub-Sub pattern

6.3.3. Strategy Pattern

We utilised the Strategy Pattern in the History service to provide scalable authorization of actions performed by users. This is executed in the form of Policies, where access can be granted or denied based on what the policies configured.

Each database entity or model has a specific and separate sequence of policies for its creation, reading, updating and deletion. Each policy takes in the object being operated on and the user that is attempting to perform that operation, then returns one of the following Authorization decisions:

- ALLOW: The user is allowed to perform that operation. This terminates any subsequent policy checking and allows the operation to proceed.
- DENY: The user is not allowed to perform that operation. This terminates any subsequent policy checking and stops the operation from happening.
- SKIP: This policy cannot make a final decision. Check the next policy.

Whenever an operation is performed, the policies are run in the sequence they are defined in until an ALLOW or DENY decision is made. As such, the sequences must all end in an *AlwaysAllowPolicy* or an *AlwaysDenyPolicy*.

```
class UserService extends BaseService<User, UserCreateData> {
  protected override createPolicies = [
    new AllowIfUserIsBotPolicy(),
    new AlwaysDenyPolicy(),
  ];
  protected override readPolicies = [
    new AllowIfUserIsSelfPolicy(),
    new AllowIfUserIsBotPolicy(),
    new AlwaysDenyPolicy(),
  ];
  protected override updatePolicies = [
    new AllowIfUserIsSelfPolicy(),
    new AlwaysDenyPolicy(),
  ];
  protected override deletePolicies = [
    new AllowIfUserIsSelfPolicy(),
    new AlwaysDenyPolicy(),
  ];
}
```

Figure 6.3.3a: Code Snippets showing *UserService* policies

This design is highly extensible and allows the implementation of highly complex logic. For example, we may want to have a policy that does a further API call to an external service to verify that a user has the permissions for a specific action. With this design, it is very easy to implement - simply slot in a new policy that does so asynchronously.

As such, the policies act as strategies that can be easily added and utilised.

6.3.4. Repository Pattern and DAO Pattern

Repository Pattern is used in Room Service to provide an abstraction of the data layer, and mediate the domain logic and persistence layer. It hides lower level database queries and centralises the handling of domain objects. The implementation of Repository Pattern in Room Service builds on top of DAO (Data Access Object). While DAO also provides an abstraction of data persistence, it is closer to the underlying database storage compared to repositories.

Taking the example of domain object, Question, in Room Service, its DAO and Repository interface look somewhat similar as it mainly deals with the basic CRUD operation of the object, however, the implementation of the DAO keeps reference of the database collection, and queries the database directly, while the implementation of the Repository fetches and access data through DAO, and hides all database-related details.

```go
type QuestionDAO interface {
  CreateQuestion(question *models.Question) error
  GetQuestionById(id string) (models.Question, error)
  FindQuestions(query bson.M) ([]models.Question, error)
  ClearQuestions() error
}

type QuestionRepository interface {
  Add(question *models.Question) error
  Get(qid string) (models.Question, error)
  GetAll(query bson.M) ([]models.Question, error)
}
```

Figure 6.3.4a: Code Snippets showing *QuestionDAO* and *QuestionRepository* interfaces

```go
type QuestionDAOImpl struct {
  collection *mgo.Collection
}

type QuestionRepositoryImpl struct {
  dao dao.QuestionDAO
}
```

Figure 6.3.4b: Code Snippets showing *QuestionDAOImpl* and *QuestionRepositoryImpl* interfaces

DAO sits directly on top of the database implementation, and repositories sit at a higher level, on top of DAO, and is being accessed directly by the business logic.

Imagine if we are to change Room Service's database from NoSQL MongoDB to Postgres, we only need to change the DAO implementations and keep the repositories and business logic of the application completely intact.

The use of both Repository and DAO Design Pattern improve the scalability of the codebase, which provide clean abstractions of the data persistence layer.

6.4. Microservices

6.4.1. Authentication MS

Authentication Microservice is essentially an adapter and extension of Firebase Authentication with GitHub. We choose Firebase Authentication as our underlying

authentication infrastructure because of the conveniences it provides to integrate with frontend and its support of multiple authentication methods, specifically GitHub login.

Authentication Microservice provides two routes, one for user login and the creation of persisted users in our own database, the other for authenticating clients' authentication tokens. We did not implement a route for sign up as Firebase Authentication took care of the signing up and integrating with GitHub parts of the logic for us.

When a user logs in, we first verify the Firebase token sent to us together with the user's profile information. If verified, the user's information is sent to the History Microservices to create a persisted user in the database. Subsequently, Authentication Microservice generates a JWT token using the user's user id obtained from Firebase. An additional JWT token is generated on top of the existing Firebase Authentication token due to the fact that Firebase Authentication token expires after one hour, which is too short for the expected duration of our mock interview process (45 minutes to 60 minutes). In order to prevent premature expiration of the authentication token during the mock interview, we generate a JWT token that expires in one day for the user upon logging in. The user will then store this token in Local Storage, and attach it in the Authorization header for subsequent requests for the authentication purposes.

The authentication route of the Authentication Microservice is mainly utilized by the API Gateway. The checkAuth middleware of Gateway intercepts HTTP requests sent to other microservices, reads the JWT token from request Authorization header and sends it to Authentication Microservice for verification. If successfully verified, it replaces the Authorization header of the request, with the user's user id returned by the Authentication Microservice. As such, when the request is forwarded to its designated microservice, it has user id in its Authorization header, allowing other microservices successfully identify the user.

6.4.2. Pairing MS

The Pairing Microservice helps to pair users together. It maintains runtime structures to save users who have joined the waiting queue and runs pairing algorithms to match users based on their preference of question difficulty as well as their ratings. No data persistence is required for this microservice, since the pairing is done in real-time.

The pairing algorithm will take into account the following factors:

1. Question difficulty chosen
2. User ratings

The first factor is strongly enforced, i.e. users who have selected easy difficulty will only be paired with other users who have selected the easy difficulty as well.

As for the second factor, we will only consider user ratings when a user has **five or more ratings**, so that the average rating is more accurate and representative of the user's behaviour. For users with less than five ratings, they can and will be paired with anyone else available.

For users with five or more ratings, the pairing algorithm will pair them with other users that are within ± 1.0 of their rating. In future iterations, this algorithm may be adjusted such that users with good ratings get more matches than users with poorer ratings.

The following runtime data structures are utilised:

1. A mapping of socket ID to user ID and user object.
    a. This allows us to identify the user that the socket represents. It is a many-to-one mapping, since a user may use multiple tabs to attempt to pair (though there are validations to enforce uniqueness).
2. A queue stratified based on selected difficulty.
    a. This allows us to identify users that the new user joining the queues can pair with.

6.4.3. Coding MS

The Coding Microservice helps the users to manage their code and execute the code when necessary. There are two main parts to this microservice:

- Collaboration: To resolve changes made by the two users, in order to arrive at a single synchronised state. Conflict-free Replicated Data Types (CRDTs) are used to resolve changes and achieve synchronisation.
- Code execution: To execute the code written. This involves some external runners that are spawned on demand.

6.4.4. Video Call MS

The Video Call Microservice handles the authentication of users when joining a video call channel. When the user joins a room, the frontend makes an API call to the Video Call Microservice to authenticate the user, which returns a unique token specific to the current session. With the token, the frontend then connects to the Agora server (the third party provider) with the token to join the video channel. Essentially, the Video Call Microservice is an authentication service that sits in between our frontend application and the third party video conferencing provider, Agora. Below is a sequence diagram showing the user activities related to video call upon joining a room.

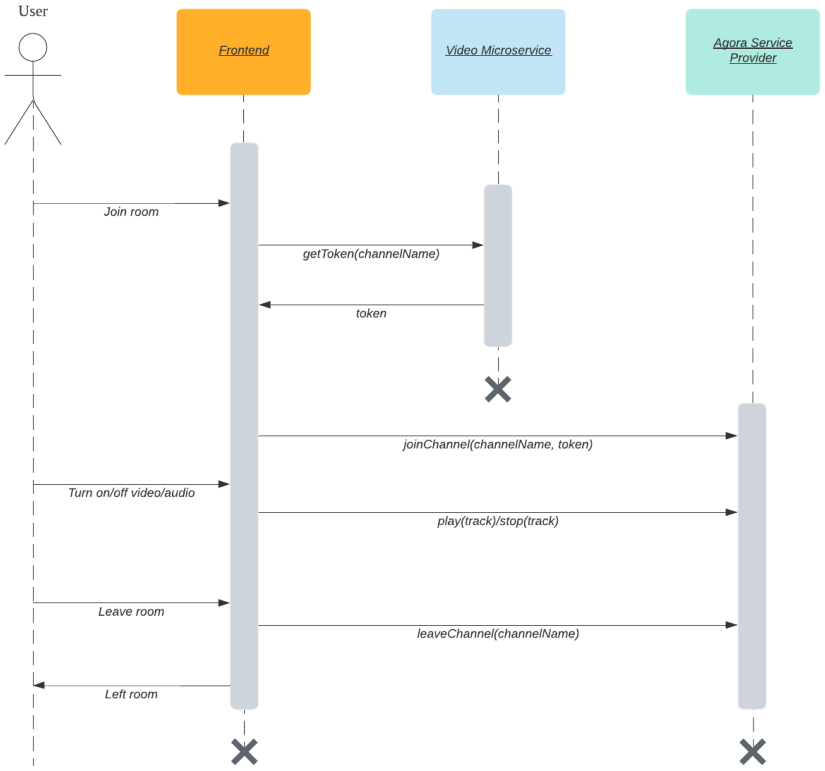

Figure 6.4.4a: Sequence diagram for video

6.4.5. Room Management MS

Room Management Microservice is in charge of room creation, question assignment and interviewer/interviewee role assignment for each paired mock interview. It also coordinates various events that may happen within the room, such as the switching of interviewer/interviewee roles, interviewer feedback submission, rating submission and disconnection of clients, etc. In order to fulfil the described functionalities, Room Management Microservice accepts both HTTP requests from other microservices and the frontend client, and it also maintains WebSocket connection with frontend client in order to manage the client's in-room status and handles various in-room events.

For the persistence layer of the Room Management Microservice, as mentioned in Section 5.3.4, a combination of Repository and DAO Design Patterns were used to provide abstraction to database access. There are two domain objects in the Room Management Microservice, Question and Room, which also corresponds to the two tables in the NoSQL MongoDB, questions and rooms. Questions table serves as our question bank for the mock interviews which will be randomly assigned to the user based on the selected difficulty, and rooms table persists information regarding a created room, including its room id, users in the room, questions assigned to the room, etc.

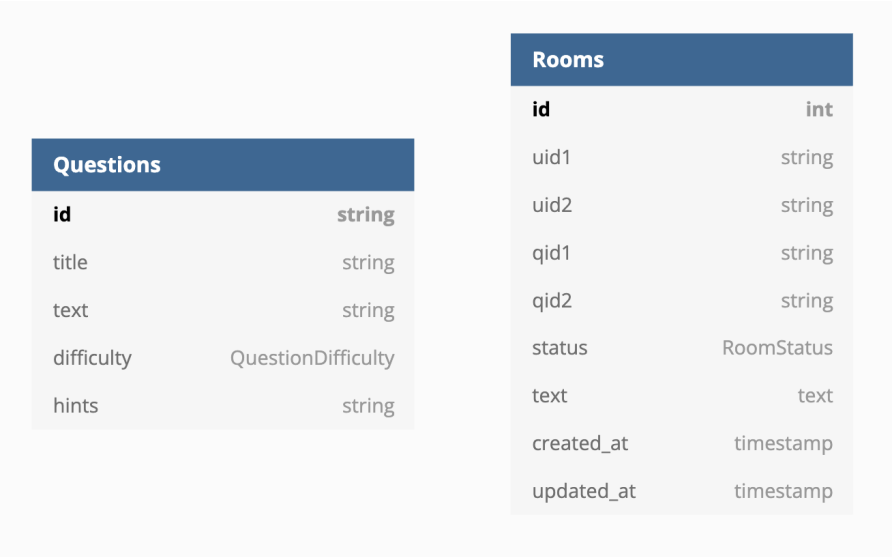

Figure 6.4.5b: Room Service DB Diagram

The two domain objects, Question and Room, also signifies the two components of the domain logic of the Room Management Microservice. Question Agents and Room Agents directly access repositories and contain most of the domain logic, which is then utilized by the processor layer. Processors are designed such that there are one specific processor for each type of incoming request, and they all implement the RequestProcessor interface, such that higher level components can call .Process() and .GetResponse() method on each processor directly, so as to retrieve response to return.

Once a HTTP request hits the Room Management Microservice server, it is directed to HTTP handler and WebSocket Handler, based on if it is a Websocket connection request. The HTTP Handler handles requests directly, by constructing respective processors for the requests and returns the responses. Websocket connection request will be sent to WebSocket Handler for upgrade, if the user JWT token included in the path parameter is verified, a Socket Client will be created for the connection and it will be registered by Socket Manager. Since the Socket Manager keeps track of all active clients, room id to active clients mapping as well as user id to active clients mapping, it easily supports the broadcasting of messages to all users in a room. For each Socket Client, two designated goroutines are created to execute readers and writers for the client connection respectively, this makes sure that at most one writer/reader to a connection is being executed at one time, while allowing concurrency between the reader and writer across multiple client connections.

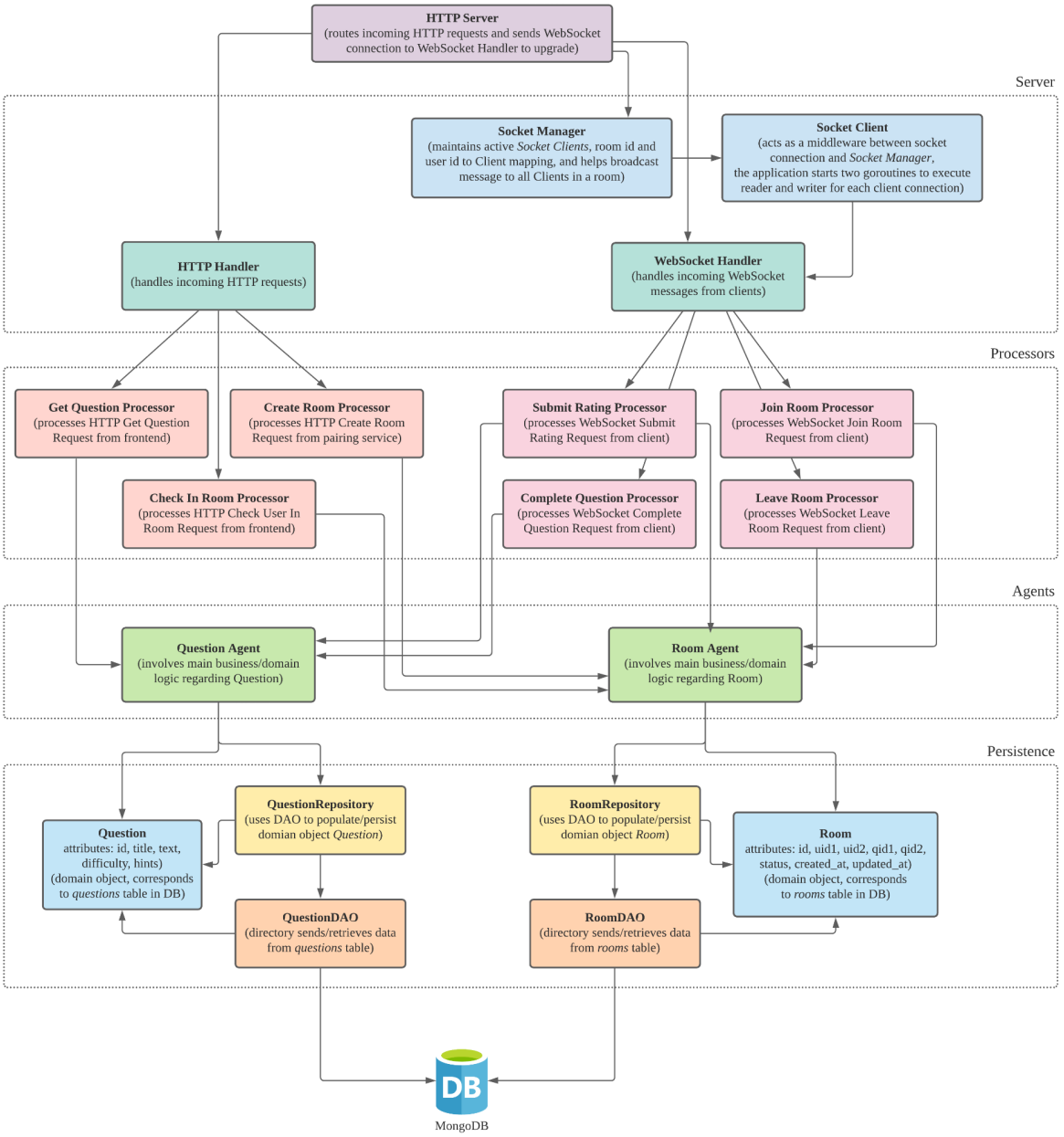

Figure 6.4.5a: Room Service Overall Architecture Diagram

6.4.6. User History MS

The User History Microservice stores the bulk of the data from the various mock interviews. All past data and statistics will be stored and computed in this microservice, allowing users to retrieve leaderboard statistics, interview histories, etc., and it will perform data persistence.

Figure 6.4.6a: User History MS DB Schema

6.4.7. MS Intercommunication

The below diagram briefly summarises the intercommunication between different microservices.

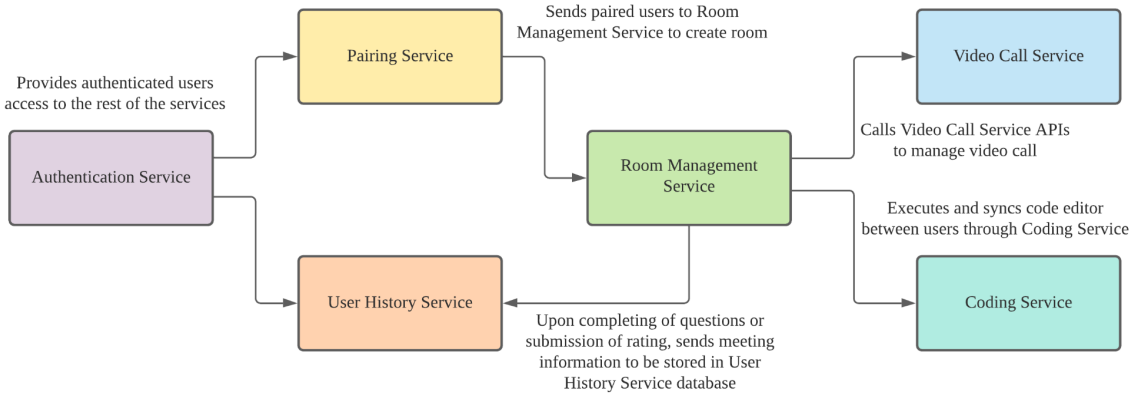

Figure 6.4.7a: Microservices Intercommunication Summary

6.5 Frontend

6.5.1. Client Architecture Diagram

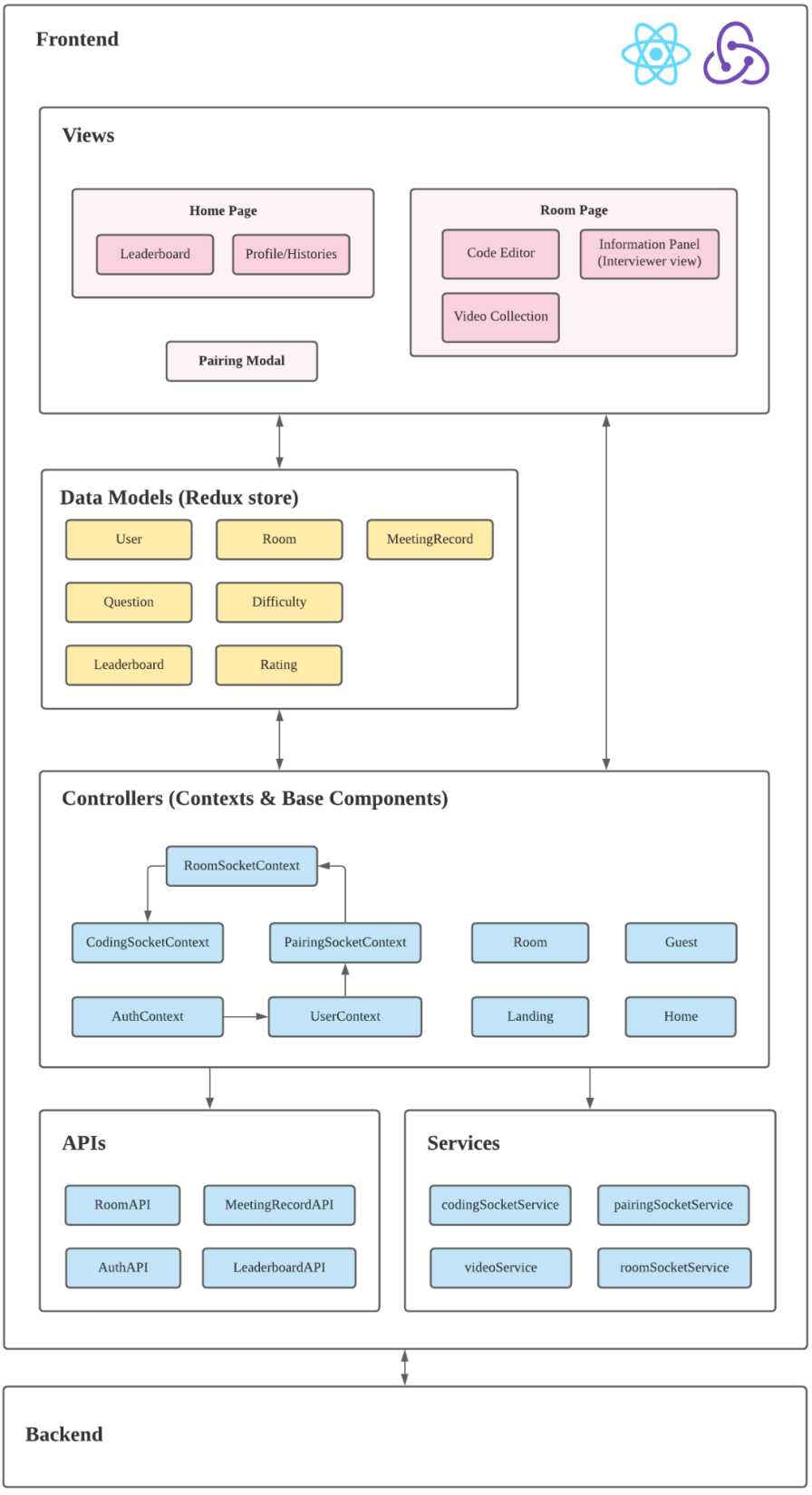

Figure 5.5.1a: Frontend (Client) Architecture

The above diagram is an overview of our frontend (client) architecture. Our frontend architecture generally follows the Model-View-Controller (MVC) architecture. However, due to the technology stack we have chosen for frontend development (React & Redux), the line between the view and controller is rather blurred, i.e. a React component contains both the view and some logic to handle user events, such as button click and form submission.

An example of the use of MVC in Code2Gather can be found in Section 5.3.1.

6.5.2 Collaboration and Synchronisation

Our application relies quite a fair bit on collaboration, since we have the collaborative code editor, powered using Automerge. This is necessary because we cannot have a case where two users have their states permanently diverged, and we will need to ensure that states converge.

In addition to Automerge, a custom **cursor synchronisation** algorithm was also implemented by our team, which utilises the socket to propagate the cursor on the code editor. This improves the user experience for the end user. However, it is not guaranteed to be 100% accurate due to limitations of the APIs available, but since this is not a collaborative context, it is acceptable.

6.6. API Calls

A Microservice Architecture is adopted by Code2Gather, hence, the application requires communication between frontend client and the microservices backend, as well as intercommunication between different microservices.

This section illustrates in detail the API calls that may occur throughout the lifespan of the Code2Gather.

6.6.1 Home Page Events

For user sign up and log in, the frontend client directly communicates with the Firebase Authentication to sign up or log in using his/her Github account. As discussed in 5.4.1, the Firebase authentication token will then be sent to Authentication Service to generate another JWT token which will be stored in local storage.

After log in, frontend client pings User History Service to fetch data that will be displayed on the home page, involving leaderboard information and meeting records.

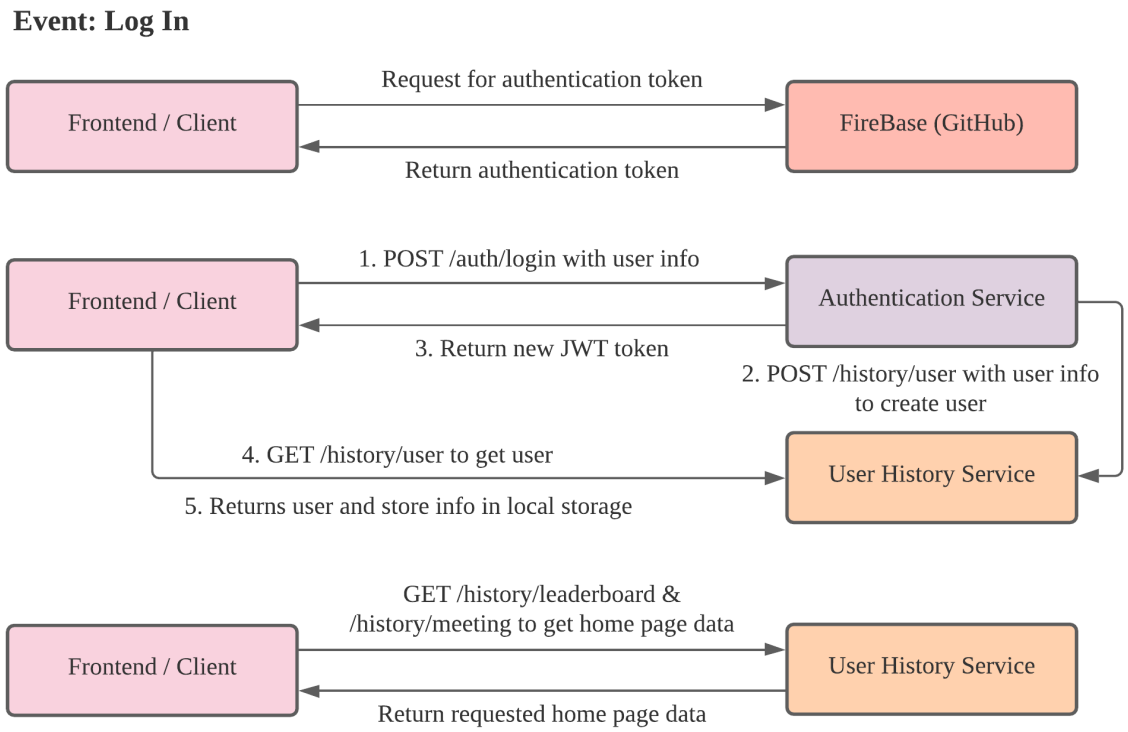

Figure 6.6.1a: API Calls when user logs in

6.6.2 Pairing Stage Events

At the start of the pairing stage, the client will first check with Room Management MS if the user is already in a room. If the user has already joined a room and is yet to leave, the “Practice Now” button will be disabled for the user, so that he/she cannot start a new round of pairing

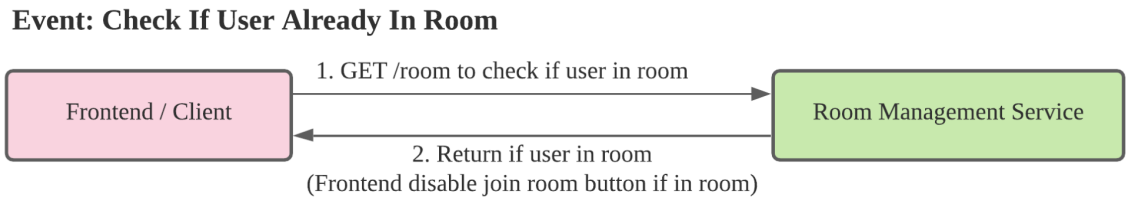

Figure 6.6.2a: API Call checking if a user is already in room

If the user is not currently in any room, he/she will be able to start the pairing process. The diagram below shows the API calls involved in a successful pairing process. At the end of the process, the user will be redirected to the room page and establish a WebSocket connection with the Room Management MS.

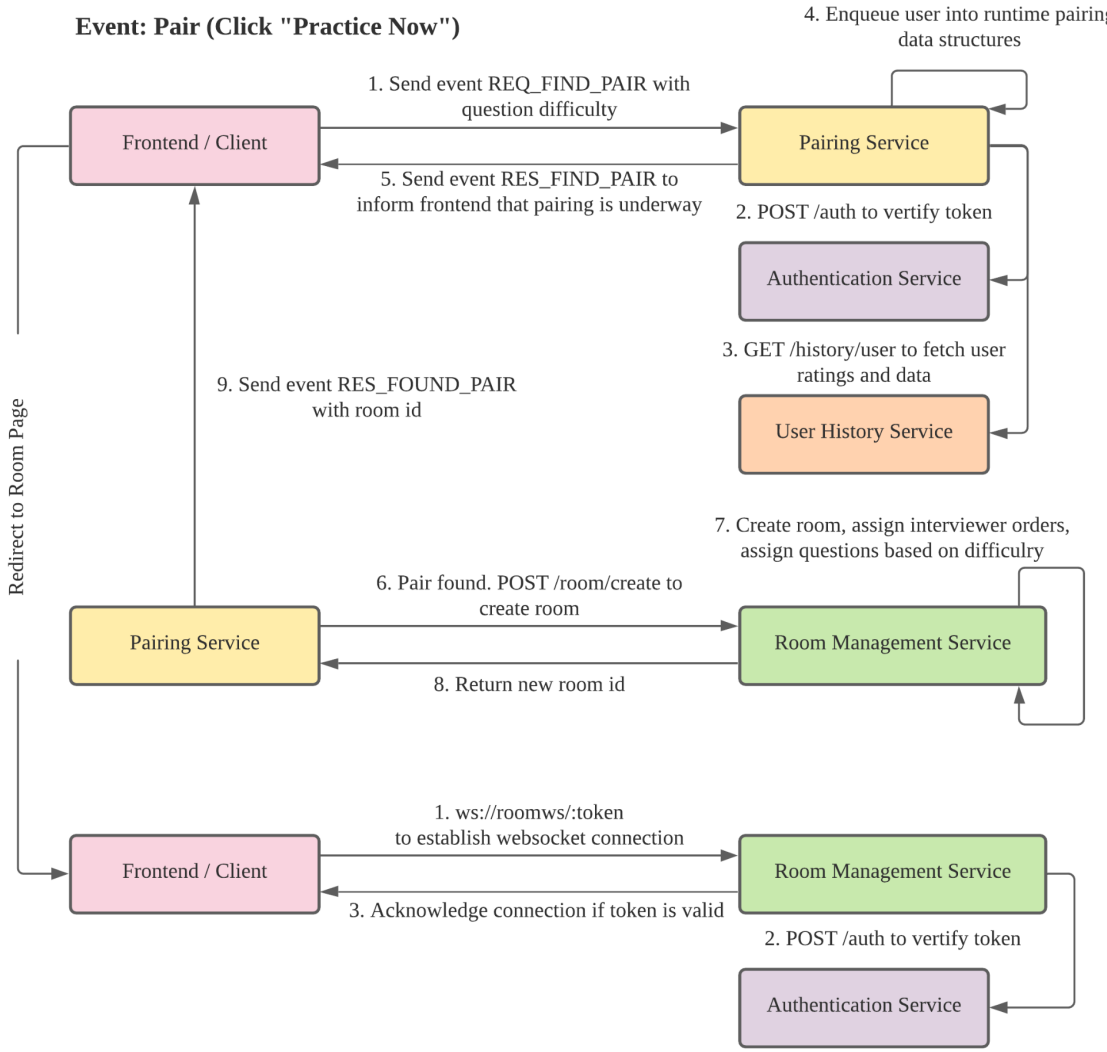

Figure 6.6.2b: API call during successful pairing process

The diagram below illustrates a few cases of failure for the pairing process. In summary, the pairing process may terminate if the pairing service connection times out and is unable to find a pair, or it can be terminated by the client from the frontend.

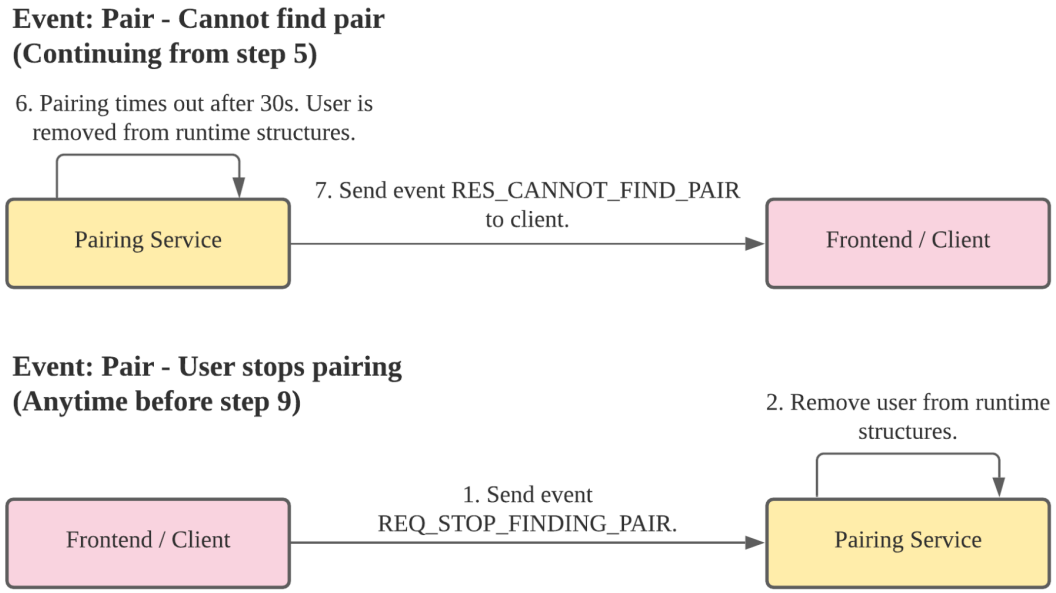

Figure 6.6.2c: Two cases of failure of pairing process

6.6.3 Room Management Events

When the user is redirected to the Home Page after the pairing process, the client will first send a JoinRoomRequest to the Room Management MS. If the client belongs to the room, the client will establish connection with both Video MS and Coding MS, and a JoinRoomBroadcast will be sent to other clients in the room.

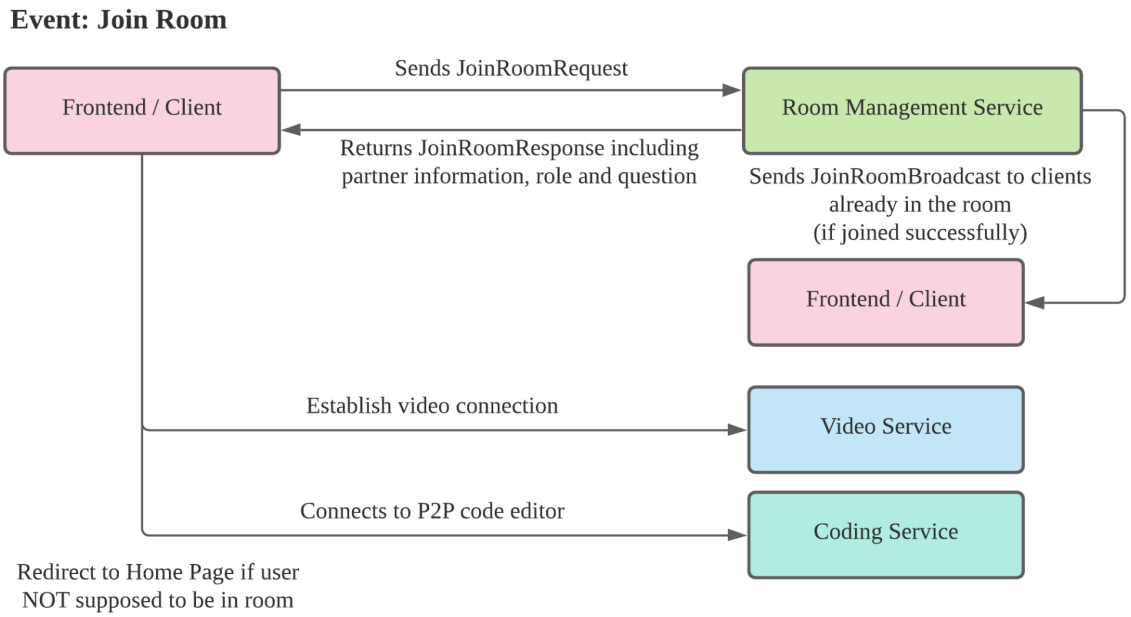

Figure 6.6.3a: Join Room Request

After the first interviewee completed the question, the interviewer can end the turn and a CompleteQuestionRequest is sent to the Room Management MS, which will then be sent to the User History Service for persistence.

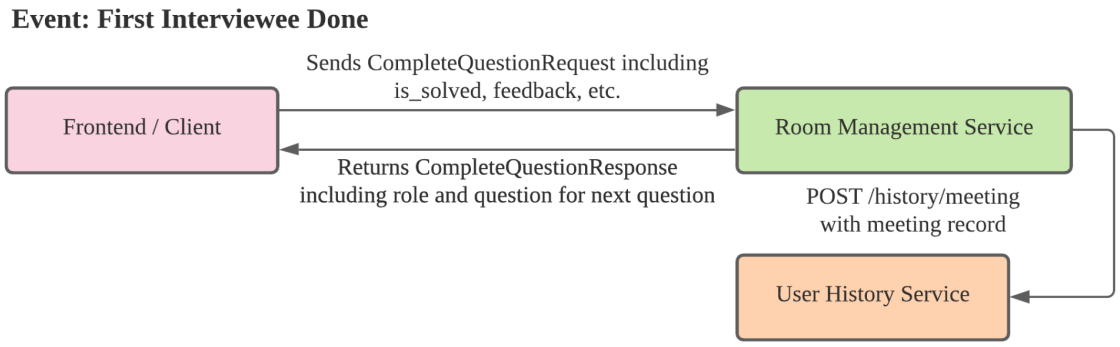

Figure 6.6.3b: First Interviewee Done

The difference between the completion of the first interview and the second is that the client will be prompted for rating after the completion of the second interview. Then, both clients will send a SubmitRatingRequest to Room Management MS.

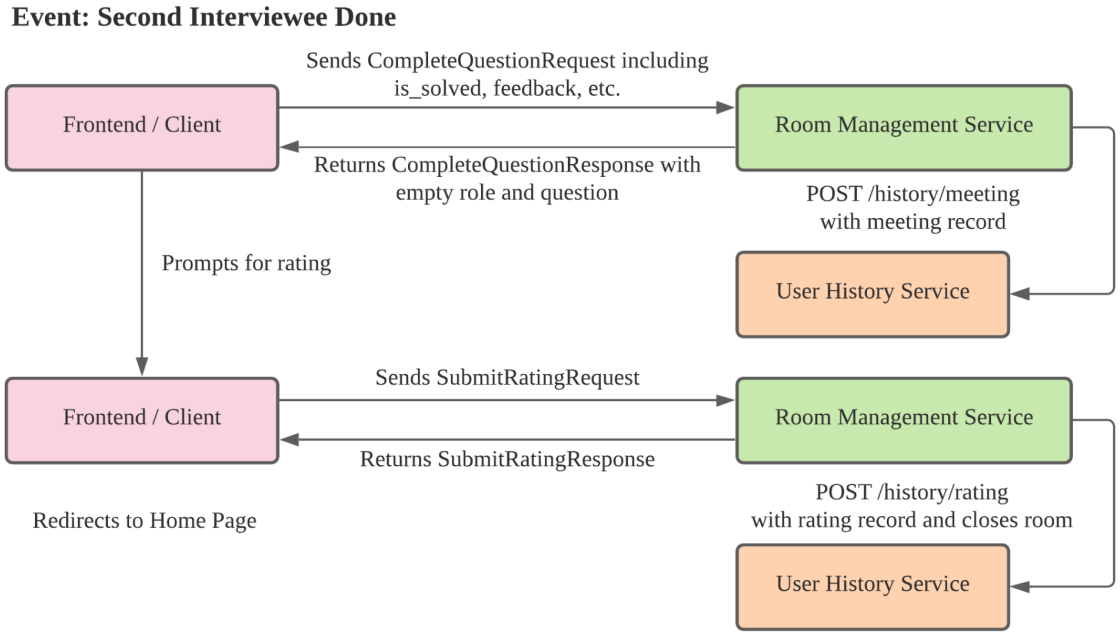

Figure 6.6.3c: Second Interviewee Done

During the interview, the client may disconnect or prematurely leave the room and terminate the interview process, below shows the handling of the two cases, where both involve the broadcasting to the remaining client in the room to improve their user experience.

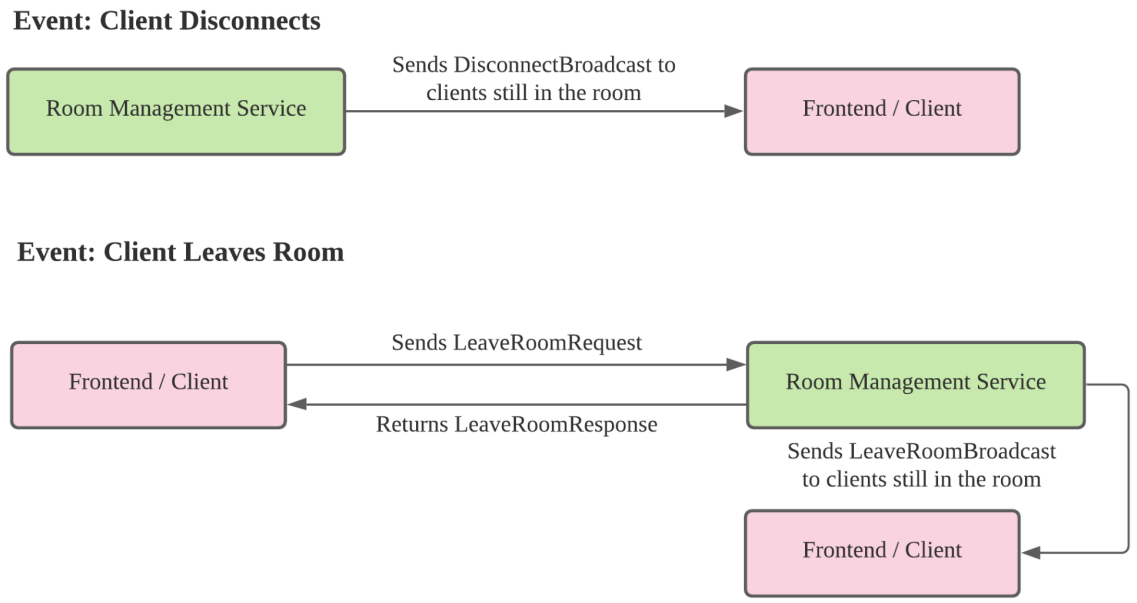

Figure 6.6.3d: Handling of client disconnection and leave room request

6.6.4. Coding Events

During the interview, interviewer and interviewee share a collaborative code editor, which updates the code content, cursor position and language used in real time.

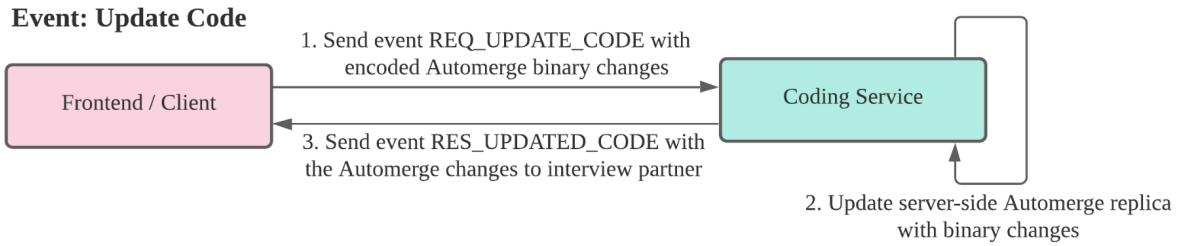

Figure 6.6.4a: Update Code Event

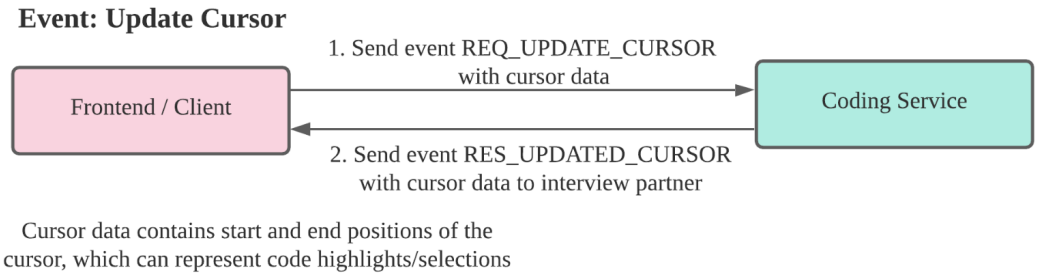

Figure 6.6.4b: Update Cursor Event

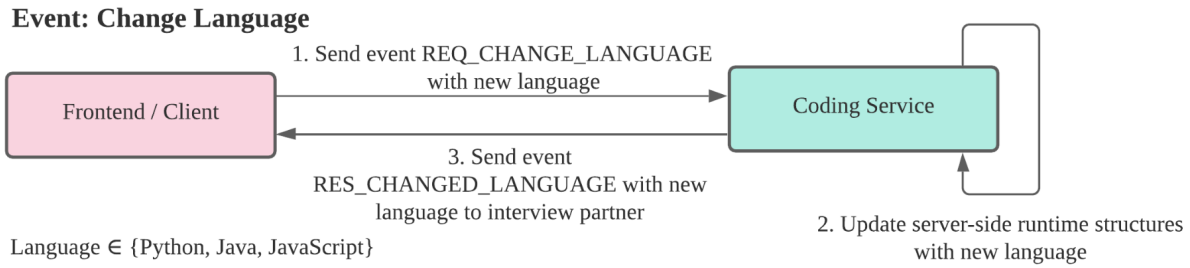

Figure 6.6.4c: Change Language Event

Clients are also able to execute code by sending a request to execute code to Coding Service, which will then send it to Code Executor Service, which pings the external library Judge0 for code execution output. Both clients will be updated with the code execution output upon completion of the execution.

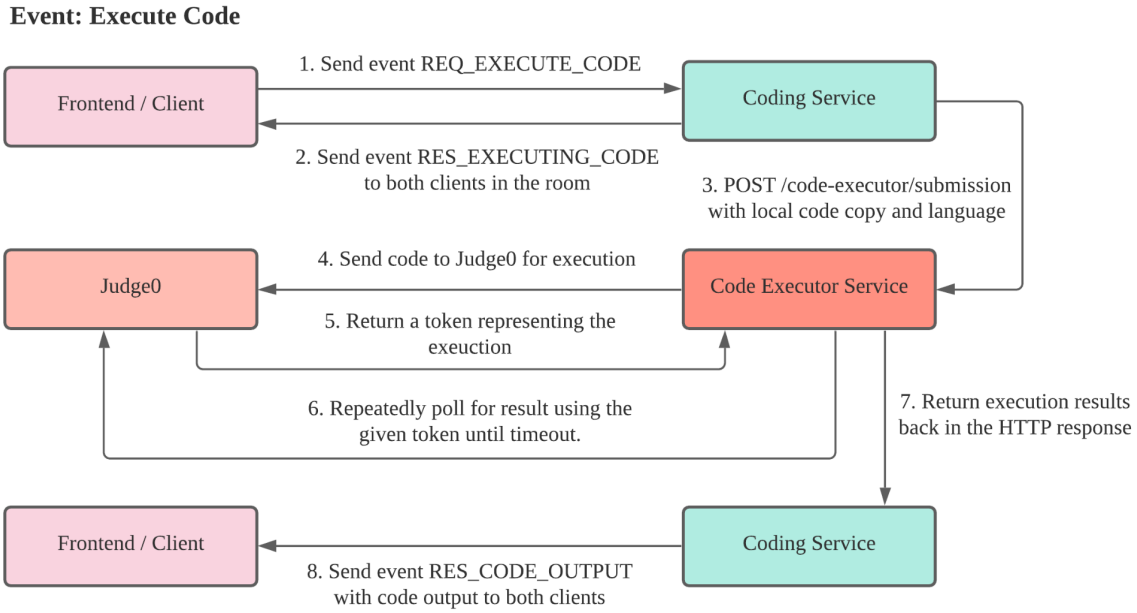

Figure 6.6.4d: Execute Code Event

6.7. Code Design

6.7.1. Database Design

The two Microservices with database implementation are Room Management MS and User History MS. Room Management MS stores the question bank of Code2Gather and persist rooms created. User History MS persists users and all meeting records in the database.

We choose to have two separate databases instead of sharing one database between microservices to clearly segregate different microservices, so that each microservice can be deployed individually. Also, we consider the questions stored in Room Management MS are more flexible that can be easily changed, while the data stored in User History is of higher significance to our application, as it stores user side information that should not be easily modified.

One thing to take note of is that there is a loose connection between the Meeting Records question field in User History MS and the questions stored in Room Management Service. However, since all information regarding the question, including its title, difficulty and code written are stored in the Meeting Record table, there is little need to enforce a strong connection between the two entities. Also, the questions in the question bank be subjected to modification and removal, however, it should affect the meeting record of an user, as the question related to a meeting record should be the one that the user attempted during the mock interview. Hence, we decided on our current database design.

6.7.2. Libraries

The following are some of the notable libraries that may be useful when implementing the above specifications:

- **React:** React is a JavaScript library for building interactive user interfaces. It is widely used for front-end development. React emphasises declarative views for ease of development and encapsulated components for reusability. React is thus good as a frontend library for its flexibility, speed, scalability and community support.

- **Redux:** Redux is a JavaScript library for managing and centralising application states. Using Redux as a state management tool allows developers to store application data that is shared across the application, as well as to cache certain data obtained via network requests for better performance.

- **Automerge:** Automerge is a JavaScript library of data structures for applications that are collaborative in nature. It supports automatic syncing and merging of application states on multiple devices to ensure that every user ends up in the same state and no changes are lost. This library can help greatly with collaborative editing inside the code editor so that both the interviewee and the interviewer are able to make changes simultaneously without causing conflicts or loss of changes.

  Automerge uses Conflict-Free Replicated Data Type (CRDT), i.e. it allows

concurrent changes on multiple edges to be merged without the need of a central server to coordinate or resolve merge conflicts. This reduces the load on the backend server by offloading the merging process to the edge devices. That being said, we would still want our server to be involved in this process as we intend to persist the interviewee's current progress on our backend. This ensures that even in the case of an Internet disruption, the users could still retrieve the code that they have written so far in their interview history.

- **Agora:** Agora is a framework that provides low-latency real-time communication services such as video and voice calls. We plan to use Agora to support voice and video conferencing for the web application to simulate more realistic interview scenarios.

  Agora promises a 99,99% uptime and 400ms average global latency. High reliability is essential in ensuring a smooth user experience when video calling his or her interview partner.

- **Judge0**: Judge0 is an open-source online code execution system that provides isolation and a friendly REST API for online code execution. We plan to use Judge0 to support code execution for the interviewees who are looking to test their code.

  Judge0 can be scaled easily, the number of workers Judge0 has can be adjusted based on the load on the system. With more workers, more submissions can be judged at any one time.

- **Protocol Buffer:** Protocol buffer is a language-neutral, platform neutral, extensible mechanism for serializing structured data. By adopting protocol buffers, we only need to define the data types once in a .proto file, which can then be compiled in multiple languages (Go and Typescript for Code2Gather).

  This allows the fast serialization of data for messages sent through WebSocket, and it also reduces code redundancy, as the message type will no longer need to be defined in frontend and multiple backend microservices, which is especially suitable for our Microservice Architecture.

6.8. Other Designs and Considerations

6.8.1. Authentication with Firebase

The team decided to adopt Firebase for authentication in the software. Firebase supports multiple providers for authentication, including Google, Facebook and GitHub. The team is thus looking at GitHub, since it's likely that the people using this platform will have some form of development-related background. This also allows them to link their behaviour on this platform with their professional/developer identity, which would likely reduce the chances of misbehaviour on the platform.

6.8.2. Video Call

As the Agora server adopts a token-exchange based approach for creating or joining audio and video channels, we have designed the following workflow to establish the call connection:

1. Once two successfully matched users are inside a room, any one party is able to initiate a video call by calling our Video Call service.
2. Our Video Call service will randomly generate a unique token and push the token to the two users in the room. The token will be used as a unique identifier for this call.
3. The first user who receives the token will be able to start the audio and video channels by passing this token to the Agora server, which then authenticates the user and starts the video streaming process.
4. The second user who receives the token can use this token to join the audio and video channels initiated by the first user.

6.8.3. Code Execution

As for code execution, we are using Judge0. Judge0 provides an API for us to send the code execution to the backend.

For each execution, the code is sandboxed within Docker with memory limit and time limit. This will prevent malicious users from sending malicious code through the code editor. Should anything malicious happen, it will only occur within the docker container which will be destroyed at the end of the execution.

It also supports authentication tokens which restrict other malicious users from contacting the API directly

6.8.4. Questions

The questions currently available on Code2Gather are authored by the team. Unfortunately, due to copyright restrictions with websites such as LeetCode and HackerRank, the team is unable to directly copy the questions that are available on these platforms.

6.8.5. Rating System

The rating system works on a basis of discrete scores from 1-5. The overall rating of a user is thus the aggregation across all ratings that a user has received. This rating is not visible to the user, it will only be used during the pairing process to improve the user experience.

However, as with most user review-driven systems, there is the possibility of inaccurate or malicious reviews, i.e. moral hazard or moral failure. The team will analyse this more carefully when implementing the software.

7. DevOps

7.1. Sprint Process

Our team has primarily adopted the Agile software development methodology, whereby our entire development process is split up into 3 major sprints, with each sprint lasting for 2 weeks. During each sprint, we complete a round of planning, building, testing and review. We have also ensured that at the end of each sprint, we have a working product that comes with the new features developed during the sprint. We conduct weekly sprint meetings to review and plan for one week's deliverables, as well as to conduct regression testing and deployments. We also conduct daily standups to resolve blocking issues as well as updating each team member's progress.

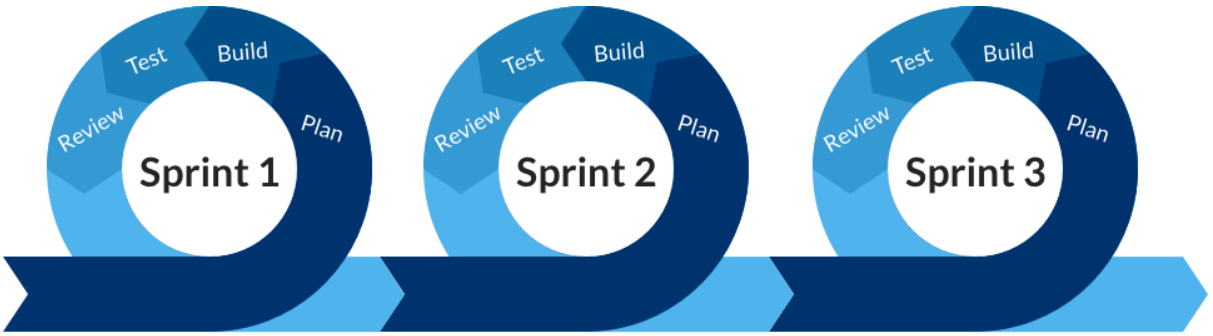

Figure 7.1a: Agile Software Development Methodology

	Sprint 1 **R. Wk - Wk 7**	**Sprint 2** **Wk 8 - Wk 9**	**Sprint 3** **Wk 10 - Wk 11**	**Sprint 4** **Wk 12 - Wk 13**
Xinyue	Set up the auth service (Firebase) and gateway Set up databases for Room Management	Implement Room Management Service	Linking up the microservices and frontend - CRDT - All WebSocket and REST API calls - Protobuf Deploying the microservices and frontend on Digital Ocean using Docker Compose Feature	Buffer Deployment to AWS using Kubernetes Measurements for Quality Attributes Finish up stretch goals Prepare for final submission
Wang Luo	Set up the frontend - Code editor - Video call Set up Firebase authentication on frontend	Implement video call features on the frontend and set up Agora token server		

Junhua	Set up the code execution functionalities	Implement Coding Service, with CRDT logic Design questions for Coding Service	refinements and implement stretch goals - Login as guest - UI/UX improvements - Better usability Test cases for frontend and backend	
Hanming	Set up the frontend - Auth screen - Home screen Set up database for User History services	Implement User History Service Implement Pairing Service Implement more frontend features: - Leaderboard - Practice history - Interview logic		

Figure7.1b: Sprint Planning

7.2. CI/CD

There are two parts to the CI/CD:

- Appropriate CI is run for all pushes and pull requests. This includes running tests, linting etc.
- **Continuous Delivery** is utilised on the staging branch, where changes will be automatically deployed to the staging server.
    - This is due to the way that the staging application is currently set-up, which requires some downtime when deployed. If continuous deployment was utilised instead, we may lead to cases where the staging app is repeatedly launched and taken down, if frequent pushes are done.

7.3. Manual Deployment

Our team has decided to make use of the Kubernetes to deploy our service. For each of the services, we have included a deployment and a service. For services which can be scaled horizontally (IE: Gateway, history, etc), we have included a horizontal pod scaler.

By using Kubernetes, it gives us more portability and flexibility as it works with any type of runtime containers and infrastructure. It also allows for easy scaling from one cloud to the next.

Deployment

1. Edit the docker-compose file and include the relevant API Keys.
2. Build all the images using `docker-compose build`.
3. Run the deployment on kubernetes using `kubectl apply -f deployment`.

8. Reflection

8.1. Possible Extensions in the future

8.1.1. User initiated practice sessions

As of the current version of Code2Gather, a user can only enter a practice session via the pairing process. Although a user can enter the guest session and send the guest room link to a friend, the guest session is publicly available to everyone who has the link. In addition, the user has no access to the video call feature in the guest mode. Nonetheless, a common use case of an interview preparation platform involves users practising with their friends instead of strangers. Hence, we plan to allow users to initiate their own practice sessions and invite a friend to practise together in the future. This could be implemented by exposing more room related API endpoints that allow invitation of users and joining rooms without being paired.

8.1.2. Text chat

As of the current version of Code2Gather, we only support video calling during a practice session. However, the user's environment (e.g. poor Internet connection, noisy environment) may not be suitable for video conferencing during the mock interview. In such cases, a text chat would be a great substitute for video calling. A text chat functionality can be easily implemented from extending the existing room socket, or from extracting text chat into an independent microservice altogether.

9. Product Screenshots

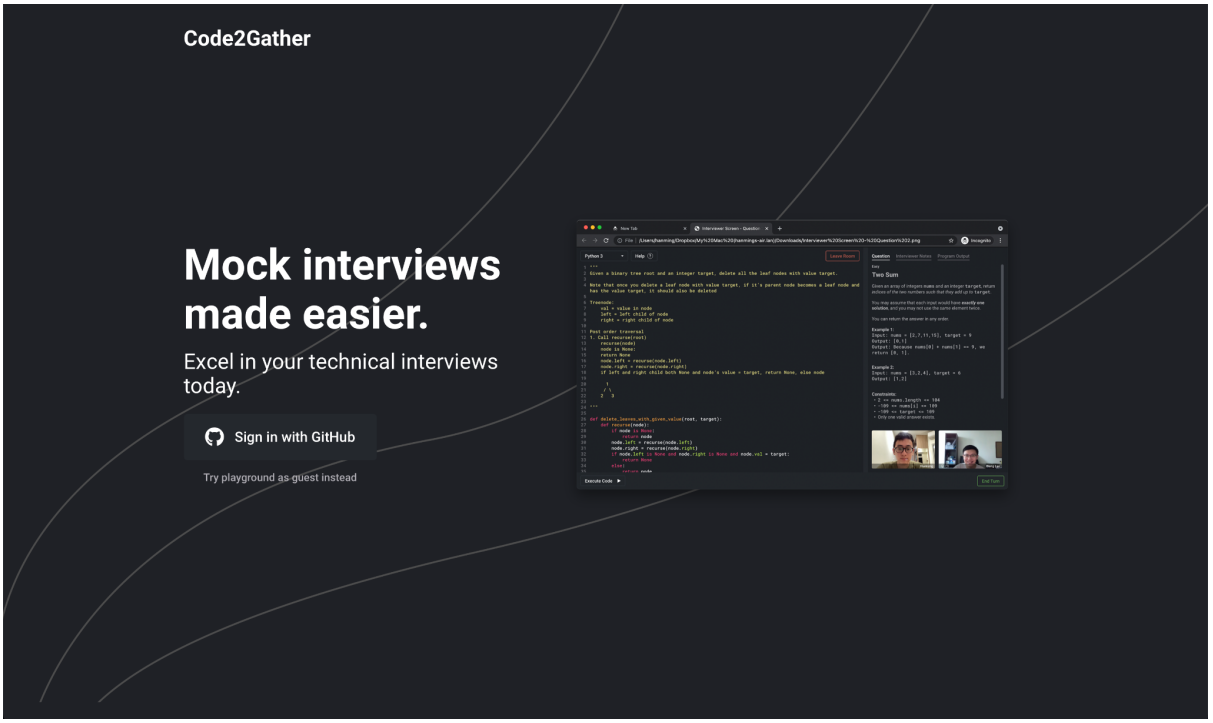

Figure 9a: Landing/authentication page

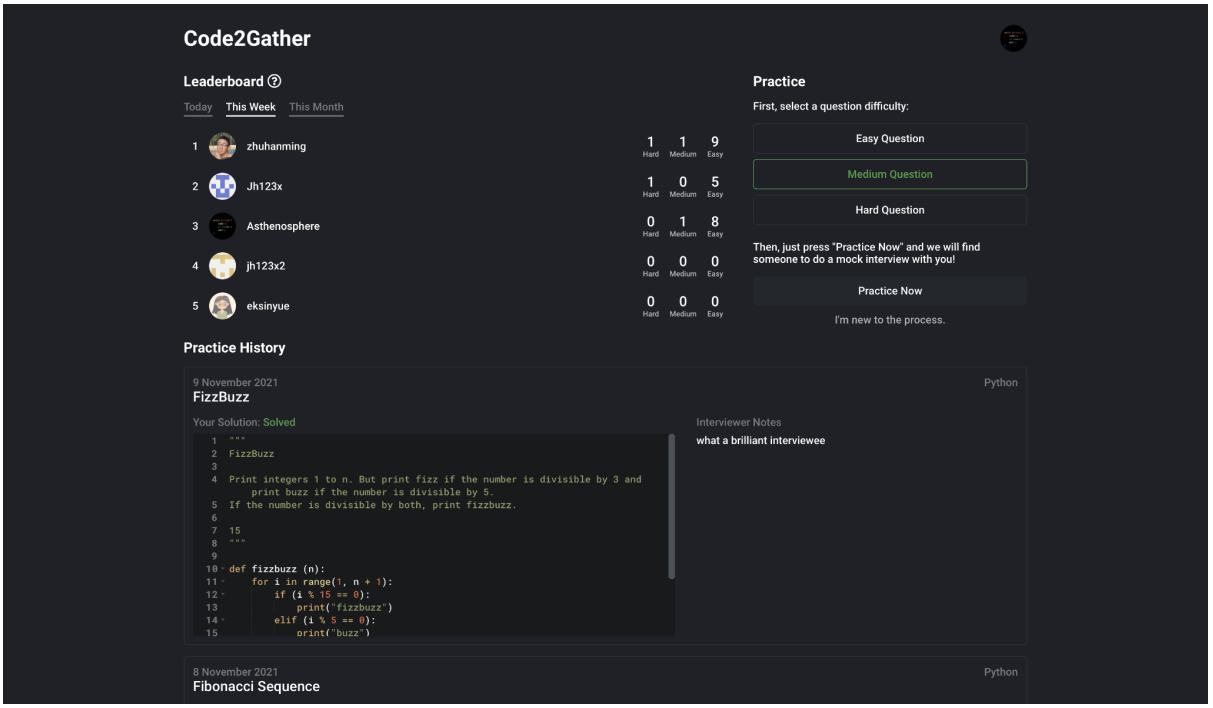

Figure 9b: Home page

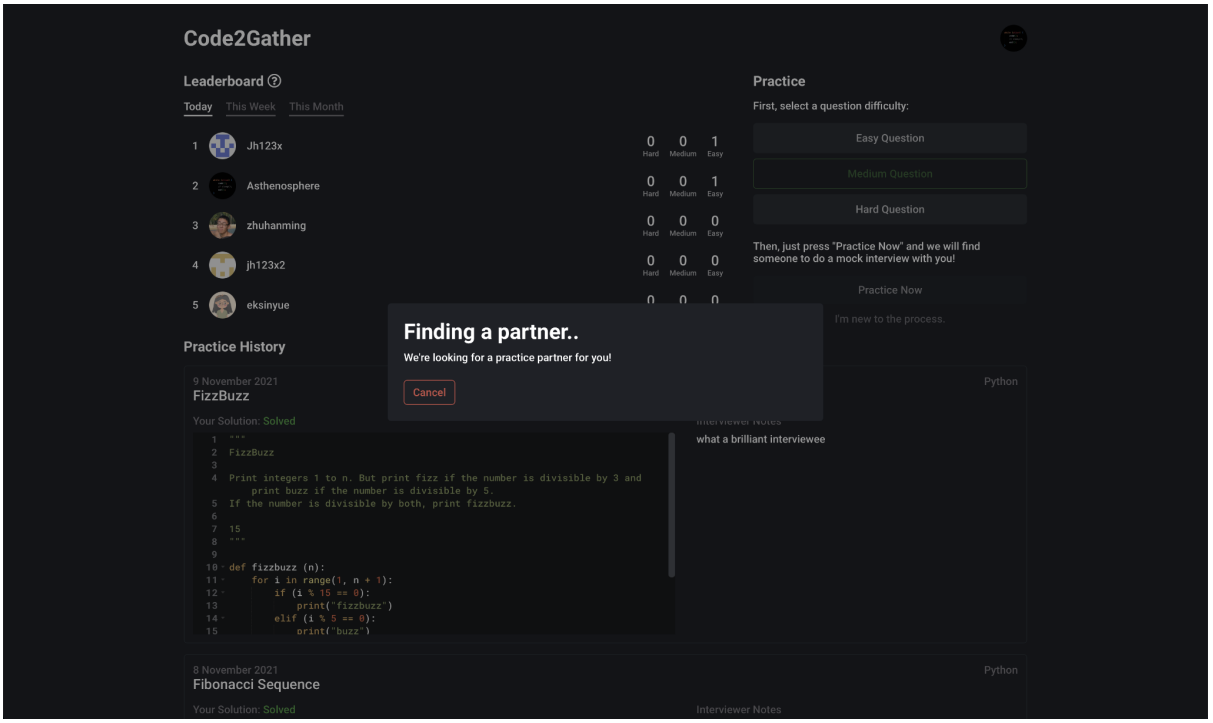

Figure 9c: Pairing modal

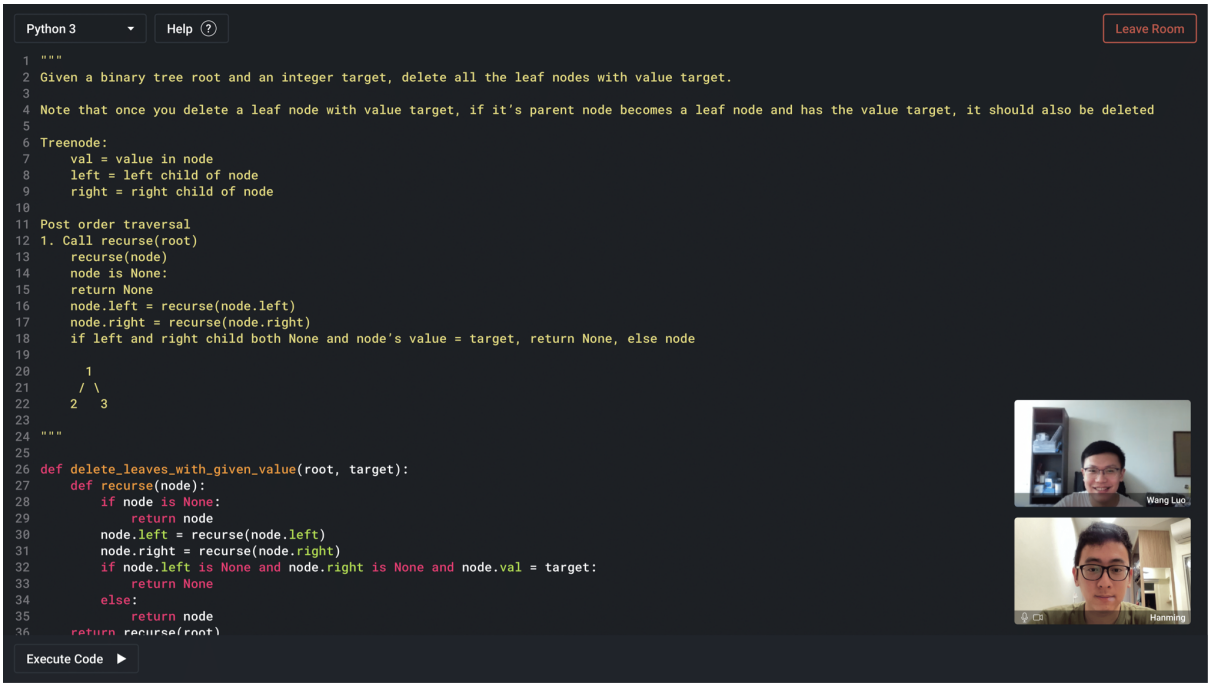

Figure 9d: Interviewee view of code editor page

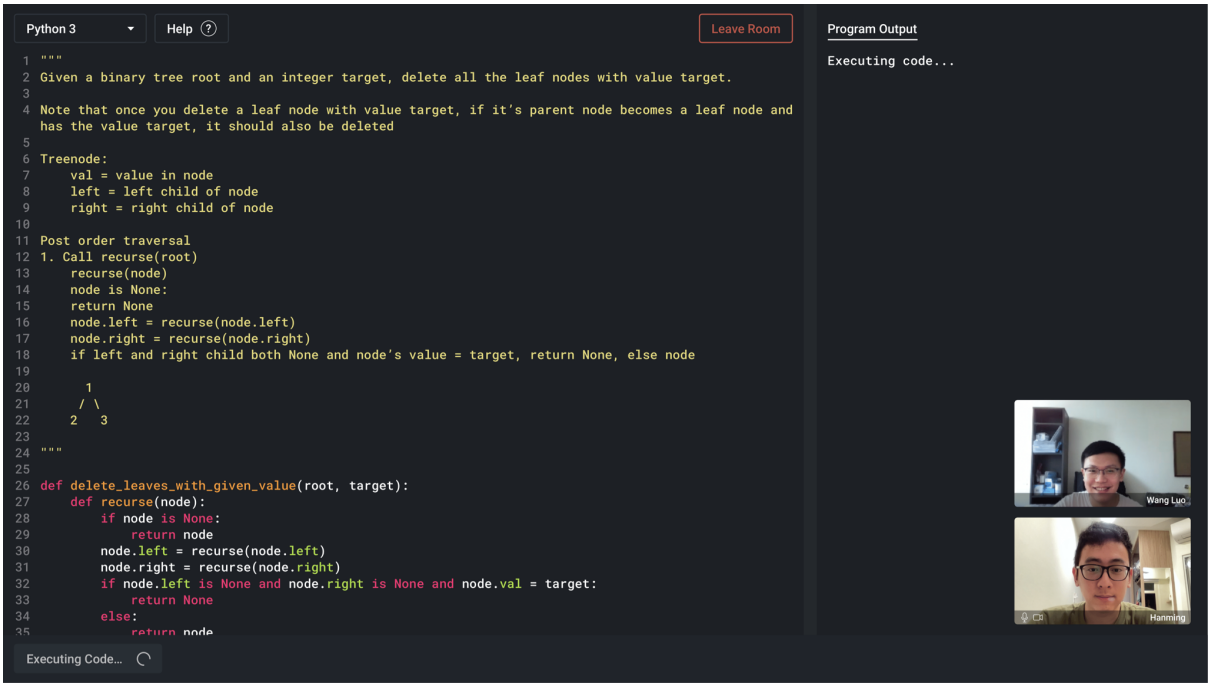

Figure 9e: Interviewee is waiting for code to execute

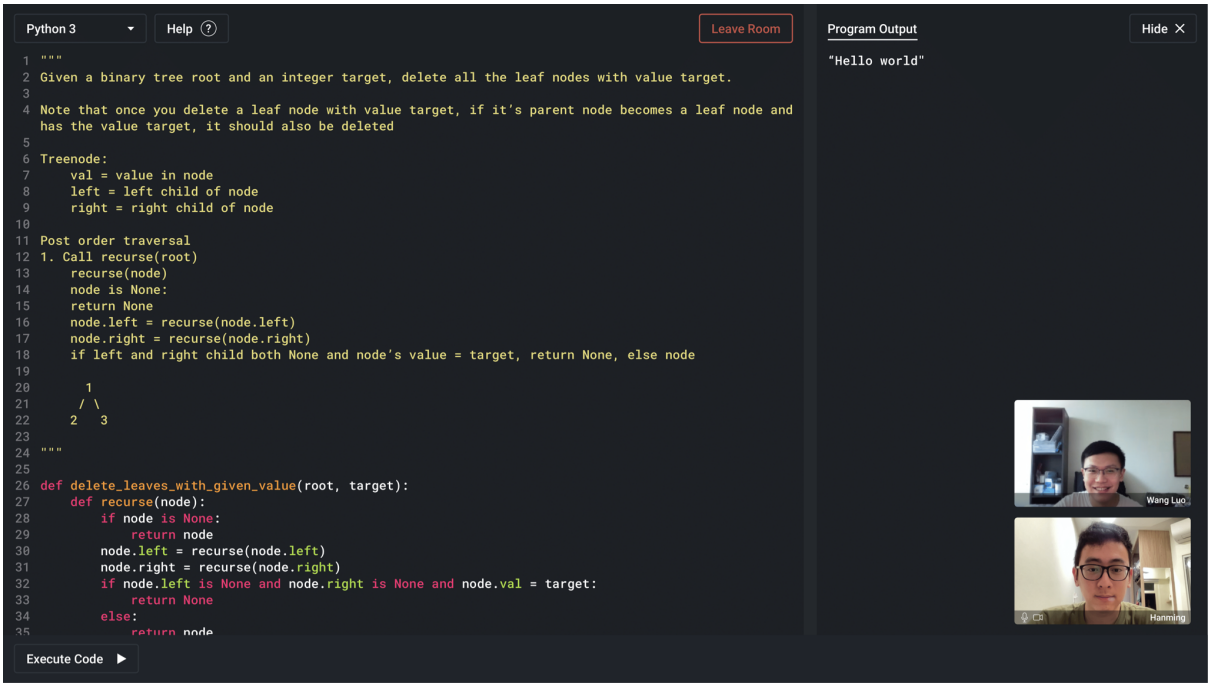

Figure 9f: Interviewee view when code has finished executing

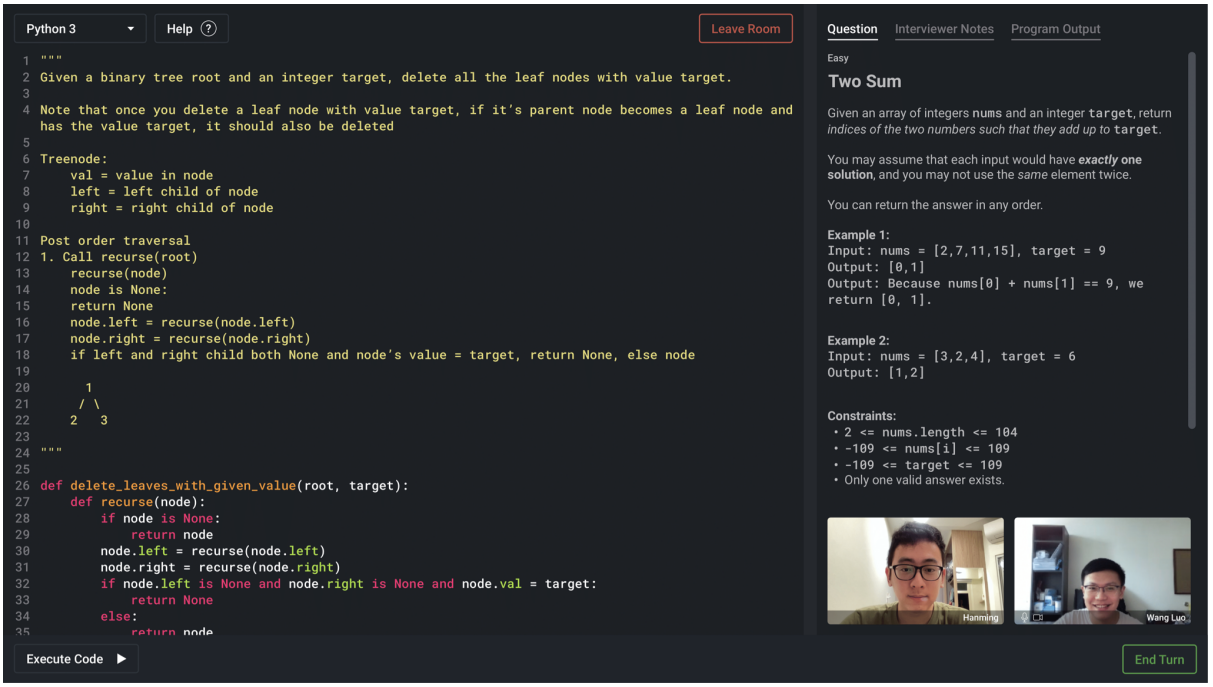

Figure 9g: Interviewer view of code editor page - question tab

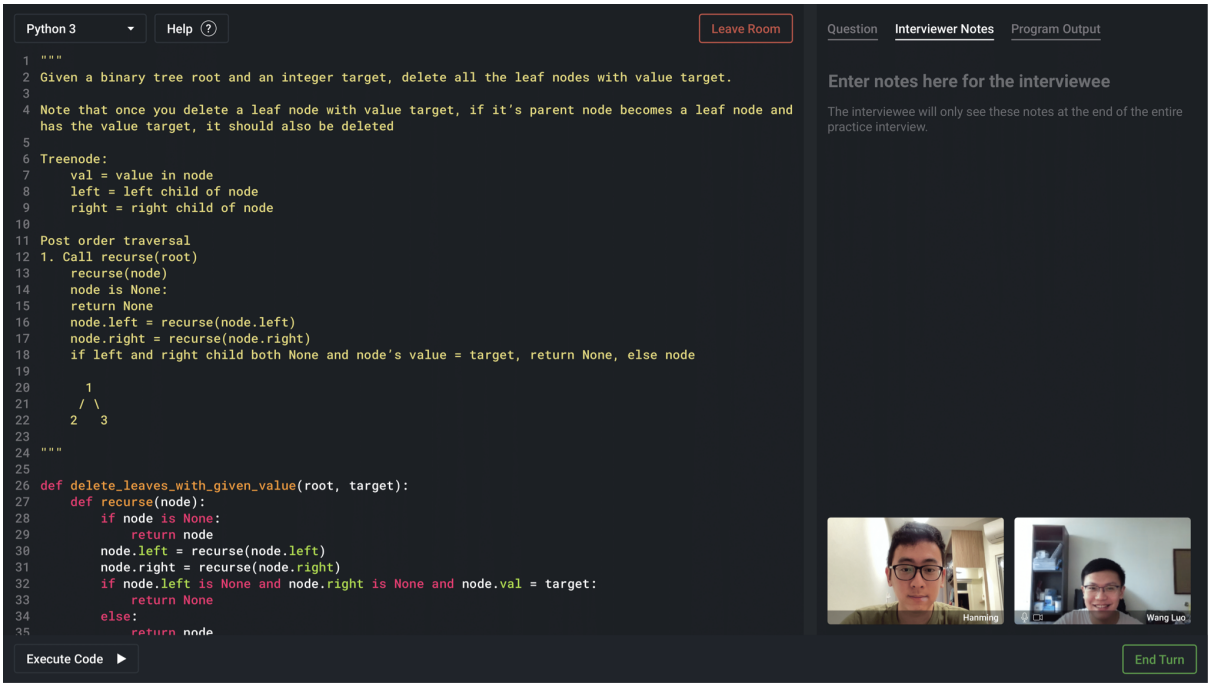

Figure 9h: Interviewer view - interviewer notes tab

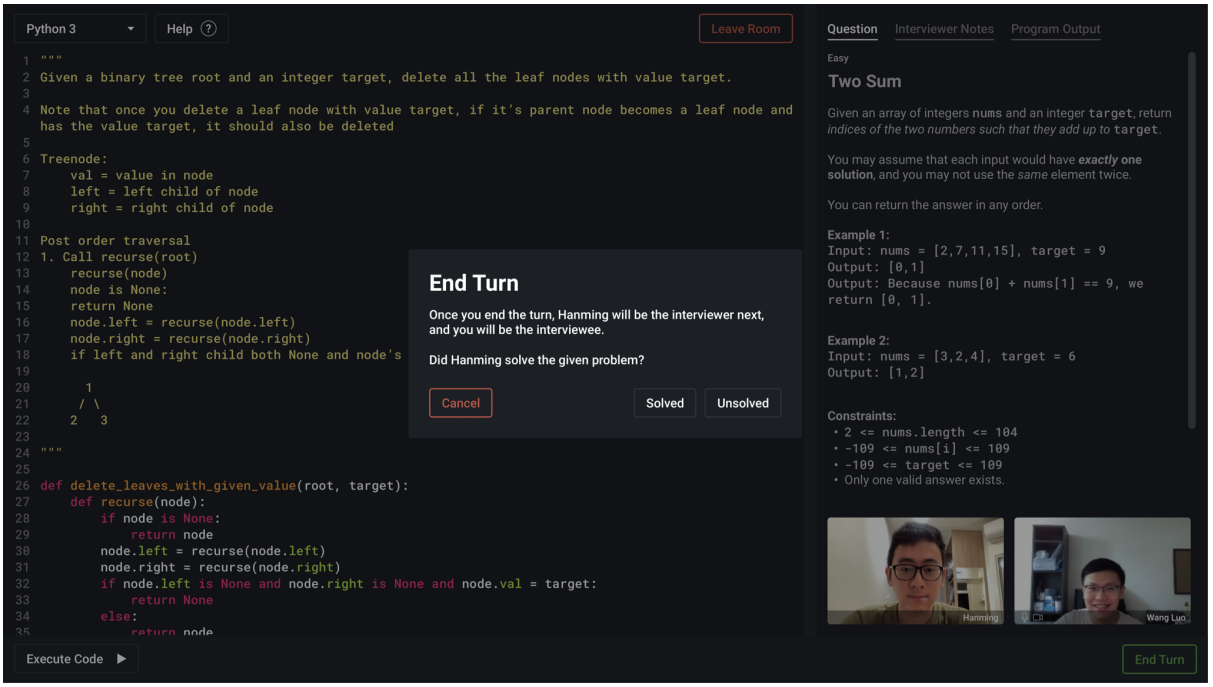

Figure 9i: Interviewer end turn modal