



**NUS**  
National University  
of Singapore

School of  
Computing

# **CS3219 Software Engineering Principles & Patterns**

## **AY22/23 Semester 1**

### **Project Report**

#### **Team 4**

<b>Team Members</b>	<b>Student No.</b>
Goh Ee Liang	A0202170B
Michaelia Tan Tong	A0177473U
Muhammad Khair Bin Abdul Rashid	A0200238W
Tan Hin Khai, Stephen	A0196608R

**Project Mentor:** Yash Sinha

# Table of Contents

<b>1. Background and Purpose of Software Interview Xchange</b>	<b>1</b>
1.1. Background	1
1.2. Purpose	1
1.3. Terminology	1
1.4. Intended Audience and Reading Suggestion	2
1.5. Project Scope	2
<b>2. Requirements</b>	<b>3</b>
2.1. User Requirements	3
2.2. Functional Requirements (FRs)	4
2.2.1. User FRs	4
2.2.2. Matching FRs	4
2.2.3. Collaboration FRs	5
2.2.4. Question FRs	5
2.3. Non-Functional Requirements (NFRs)	6
2.3.1. Security NFRs	6
2.3.2. Performance NFRs	7
2.3.3. Scalability NFRs	8
2.3.4. Integrity NFRs	8
2.3.5. Usability NFRs	9
2.3.6. Availability NFRs	9
2.3.7. Safety NFRs	9
2.3.8. Installability NFRs	9
<b>3. Developer Documentation</b>	<b>10</b>
3.1. Overall Architecture	10
3.2. Architecture Design Decision	11
3.3. Frontend	14
3.4. Backend	19
3.4.1. User Service	21
3.4.2. Matching Service	24
3.4.3. Room Service	27
3.4.4. Collaboration Service	30
3.4.5. Question Service	34
3.4.6. PubSub Service	37
3.5. Development & Deployment	39
<b>4. Project Timeline and Team Contributions</b>	<b>42</b>
4.1. Project Management	42
4.2. Project Timeline	44
4.2.1. Iteration 1	44
4.2.2. Iteration 2	44

4.2.3. Iteration 3	45
<b>5. Possible Improvements</b>	<b>46</b>
5.1. Scalability & Deployment	46
5.2. User Experience	46
5.3. Matching Selection	46
5.4. Question Variance	47
5.5. Additional Training Vectors	47
<b>6. Reflections</b>	<b>48</b>

# 1. Background and Purpose of Software Interview Xchange

## 1.1. Background

Notably, there has been an increase in companies requiring their candidates to undertake challenging algorithmic problems as part of their interview process. This has resulted in many spending time focusing on algorithms excessively, which might be detrimental as the algorithmic domain is extensive and learning too much in one sitting can cause burnout - especially when the question is of a topic that one has not heard/learnt.

At the same time, many might be able to visualise the concepts needed to solve the problem but are unable to articulate them in a concise manner. This hinders their chances of securing the interview as communication and technical skills are key requirements in such positions. This is especially true for students, as most would still need to brush up on their skills to obtain career experience and progression. As such, there needs to be a solution that provides a more enriching experience to learn algorithms and an avenue to train their communication skills at the same time.

Introducing Software Interview Xchange (SIX) the premier platform made by students for students to prepare for technical interviews.

## 1.2. Purpose

The purpose of Software Interview Xchange (SIX) is to help students better prepare themselves for technical interviews – both in terms of their technical knowledge and their communication skills. By providing a collaborative avenue for two individuals to meet virtually and work on the same technical question together, it allows the following interactions to take place:

- Revision of core algorithmic concepts through Leetcode questions
- Providing mutual feedback and discussion of (sub)optimal solutions and code quality
- Share thought processes, practice clarification and explain their codes succinctly

Additionally, being able to find somebody with a common goal to discuss Leetcode questions actively would help many students to find the motivation to practise more and break away from the mundane and lonely traditional ‘LeetCode’ grind that many of them will face when it comes to internships or job applications.

## 1.3. Terminology

Terminology	Meaning
SIX	Software Interview Xchange, the application presented in this report.
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)	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.

*Table 1.3.1: Terminology*

#### 1.4. Intended Audience and Reading Suggestion

This document is intended for developers, designers and project managers keen to implement a solution targeting the mentioned problem.

A proof of concept and iteration of this solution will be built by a team of students from NUS as part of CS3219: Software Engineering Principles and Patterns, under the guidance of professors and teaching assistants with the moniker “Software Interview Xchange”.

#### 1.5. Project Scope

The software is a web-based application to be hosted online providing a clean, user-friendly interface for users and allowing them to match with peers of similar skill levels and practise technical interviews. It should be similar to how they are conducted in reality, with one interviewer and interviewee, with each having access to different views.

The software will be targeted to anyone interested in jump-starting their career in tech and desiring to practise for technical interviews with peers, and jump into the mindset of the interviewer and understand how they think, and therefore better understand how to react to the interviewer’s responses.

## 2. Requirements

### 2.1. User Requirements

In order to draft and categorise the requirements for Software Interview Xchange, simple user stories can be used to brainstorm further:

- As a user, I would want to be able to log in
  - This shows there is a need for some user service
  - To handle both login and user particulars/preferences/history
- As a user, I would want to be matched with someone based on a selected difficulty
  - This shows that there needs to be some range of questions that users can select
  - This shows that there need to be some matching criteria
  - This also implies that there needs to be some waiting room and a timeout if no pairing is found
- As a user, I would want to collaborate with others in the same room
  - This shows that there need to be avenues for individuals in the same room to communicate

With such user stories, an overall ‘happy path’ user activity diagram can be drawn out to draft further requirements. The orange paths represent user related activities, green represent matching and pink represents collaboration.

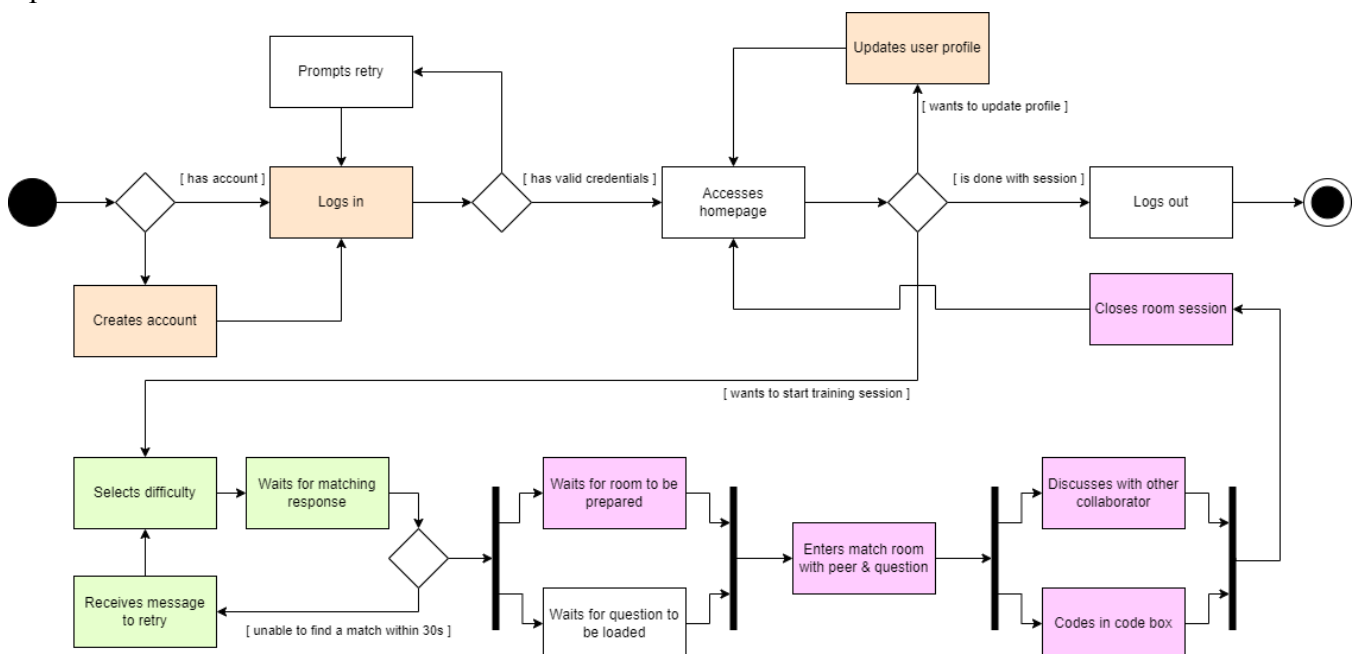


Figure 2.1.1: ‘Happy-path’ activity diagram

## 2.2. Functional Requirements (FRs)

From the happy path diagram in the previous section, the key context boundaries can be extracted - namely User, Matching, Collaboration and Question. Functional requirements for each boundary can then be crafted, as seen in the sections below.

### 2.2.1. User FRs

S/N	Nature	Description	Priority	Design Element
FR-U-1	User	The system should allow users to create an account with username and password.	High	Auth0 Signup Page
FR-U-2	User	The system should ensure that every account created has a unique username (email).	High	Users will receive an error message if they try to sign up for another account using an existing email in the User DB
FR-U-3	User	The system should allow users to log into their accounts by entering their username and password.	High	Login button at the nav bar on the landing page
FR-U-4	User	The system should allow users to log out of their accounts.	High	Logout button under the Profile navigation bar tab
FR-U-5	User	The system should allow users to delete their accounts.	High	View Profile page allows users to click “Delete My Account” button to delete their accounts
FR-U-6	User	The system should allow users to change their password.	High	View Profile page allows users to click “Change password” button to change their passwords
FR-U-7	User	The system should allow users to update their nicknames	High	View Profile page allows users to type their new nickname and click the “Update nickname” button to update their nicknames

*Table 2.2.1.1: User FRs table*

### 2.2.2. Matching FRs

S/N	Nature	Description	Priority	Design Element
FR-M-1	Match	The system should allow users to select the difficulty level of the questions they wish to attempt.	High	Train page allows users to select difficulty by clicking the “EASY”, “MEDIUM” or “HARD” buttons.
FR-M-2	Match/ Room	The system should be able to match two waiting users with similar difficulty levels and put them in the same room.	High	-

FR-M-3	Match	If there is a valid match, the system should match the users within 30s.	High	Reroute the paired users to their unique room page
FR-M-4	Match	The system should inform the users that no match is available if a match cannot be found within 30 seconds.	High	Popup alert informing users about the 30s timeout and route users back to the Train page

*Table 2.2.2.1: Matching FRs table*

### 2.2.3. Collaboration FRs

S/N	Nature	Description	Priority	Design Element
FR-C-1	Collab	The system should provide a means for the user to join a room once matched.	High	Reroute the paired users to their unique room page
FR-C-2	Collab	The system should be able to communicate peer-to-peer on a notepad environment.	High	Code box that loads & syncs data from other peer/server.
FR-C-3	Collab	The system should allow participants to quit at any time in the collaborative session.	High	-
FR-C-4	Collab/ Room	The system should allow students to indicate completion of the problem after collaboration.	Medium	A reveal solution button that determines the end of a session and unlocks the solution tab.
FR-C-5	Collab	The system should allow re-entry to disconnected collaborators.	Medium	Either by reloading the room page they were in or through the 're-join' session in the user history page.
FR-C-6	Collab	The system should be able to detect disconnection and prompt the inactive user's partner.	Medium	Prompting alerts on the browser when a user leaves/joins the room.
FR-C-7	Collab	The system should auto-close the room after disconnection from both users or termination from either user.	Medium	-
FR-C-8	Collab/ Room	The system should allow users to review past history of collaboration.	Low	A user history page loads a table with peer name, question name, start/end datetimes and a rejoin button if still ongoing.

*Table 2.2.3.1: Collaboration FRs table*

### 2.2.4. Question FRs

S/N	Nature	Description	Priority	Design Element
FR-Q-1	Qns	The system should be able to pull questions from the Question Bank.	High	-



FR-Q-2	Qns	The system should be able to display questions with descriptions and levels.	High	A question box when users are matched to a room page.
FR-Q-3	Qns	The system should provide a randomised question given a specified difficulty	High	-
FR-Q-4	Qns	The system should be able to provide the solution when a completion prompt has been indicated by either peer in the room	Medium	A solution tab in the question box.
FR-Q-5	Qns/ Match	The system should provide options (Topic/ Company) aside from difficulty to select questions and match users	Nice to Have	-

Table 2.2.4.1: Question FRs table

## 2.3. Non-Functional Requirements (NFRs)

NFRs are prioritised as per the table below. The following sections below elaborate on the specific requirements for each attribute chosen and scored.

Attribute	Score	Security	Performance	Scalability	Integrity	Usability	Availability	Safety	Installability
Security	4		^	<	<	<	<	<	<
Performance	4			<	<	<	<	<	<
Scalability	2				^	^	^	<	<
Integrity	1					<	^	<	<
Usability	1						^	<	^
Availability	1							^	^
Safety	1								^
Installability	3								

Figure 2.3.1: NFR prioritisation and scoring

### 2.3.1. Security NFRs

S/N	Description	Priority
NFR-SEC-1	Users' passwords must be hashed and salted before storing in the User DB.	Medium
NFR-SEC-2	Data stored in User DB is encrypted using TLS with at least 128-bit AES encryption.	Medium
NFR-SEC-3	All REST APIs require authentication via JWT	Medium
NFR-SEC-4	Account created must have a strong password	Low

Table 2.3.1.1: Security NFRs table

For NFR-SEC-1, NFR-SEC-2 and NFR-SEC-3, Auth0 have been utilised to provide the benefits of secure login and user management as Auth0 always hash and salt passwords using bcrypt, encrypt data at rest and in motion using TLS with at least 128-bit AES encryption and support signing JWT with both HMAC and RSA algorithms.<sup>1</sup>

<sup>1</sup> Arias, Dan. "Adding Salt to Hashing: A Better Way to Store Passwords." Auth0, February 25, 2021. <https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/>.

For NFR-SEC-4, new users need to create a new account with a strong password that fulfils the minimum complexity requirements for business organisations<sup>2</sup>:

- At least 8 characters
- At least 3 of the following:
  - Lower case letters (a-z)
  - Upper case letters (A-Z)
  - Numbers (0-9)
  - Special characters (e.g. !@#\$%^&\*)

NFR-SEC-3 is verified during the testing of room service to ensure that unauthorised users are not able to access the rooms for other users. There are notably 2 layers of authentication, the first being Auth0's verification through JWT, and the second being the JWT being analysed to determine if a user is allowed to access the requested resource.

✓ should give unauthorised 401 (181ms)

*Table 2.3.1.2: Test outputs for NFR-SEC-3*

## 2.3.2. Performance NFRs

S/N	Description	Priority
NFR-PER-1	The latency between updates on service communication (PubSub, Matching Service, Collaboration Service and Room Service) should be < 3 seconds.	High
NFR-PER-2	When the matching service is not busy, it should match 2 users who have yet to time out and pick the same difficulty within 3 seconds.	High
NFR-PER-3	Question retrieval should return the question within 0.5 seconds	High
NFR-PER-4	When the matching service is busy with match requests, it should attempt to match all of them within 10 seconds	Medium

*Table 2.3.2.1: Performance NFRs table*

For NFR-PER-2 and NFR-PER-4 (matching service is load-tested with 1,000 matching requests), these are verified during the testing of the matching service:

```

✓ Create match2 and match3 (of same difficulty) + match2 and match3 should pair together
within 3 seconds (3035ms)

✓ Create match6 and match7 (of same difficulty & within 3 seconds from match6's timeout
) + match6 and match7 should pair together (29030ms)

✓ Should be able to support 2000 match requests (40076ms)

```

*Figure 2.3.2.2: Test outputs for NFR-PER-2 & -4*

<sup>2</sup> Goyvaerts, Jan, and Steven Levithan. "Regular Expressions Cookbook, 2nd Edition." O'Reilly Online Learning. O'Reilly Media, Inc. Accessed November 3, 2022.  
<https://www.oreilly.com/library/view/regular-expressions-cookbook/9781449327453/ch04s19.html>.

### 2.3.3. Scalability NFRs

S/N	Description	Priority
NFR-SCA-1	The matching service should handle up to 2000 users at any point in time.	Medium
NFR-SCA-2	Services should be able to handle the dynamic growth of the user base	Low

*Table 2.3.3.1: Scalability NFRs table*

For NFR-SCA-1, the matching service was load-tested with 2000 matching requests from unique mock users. Using the MongoDB database, it was ensured that all of the matching requests were successfully processed which means that the matching service is capable of handling 2,000 users at any point in time.

This is verified during the testing of the matching service:

✓ Should be able to support 2000 match requests (40076ms)

*Figure 2.3.3.2: Test outputs for NFR-SCA-1*

For NFR-SCA-2, Auth0 can support up to 7000 active users<sup>3</sup> in the free version and higher in the paid versions. These services are also able to handle 7000 or more matching requests gracefully and the socket.io server should be able to handle at least 10000 concurrent connections<sup>4</sup>.

### 2.3.4. Integrity NFRs

S/N	Description	Priority
NFR-INT-1	The matching service should remove the timed-out and disconnected match requests within 3 seconds.	High

*Table 2.3.4.1: Integrity NFRs table*

For NFR-INT-1, the matching service should remove the timed-out or disconnected match request within 3 seconds to ensure the correctness of the matching service and prevent timed-out and disconnected matches from pairing with new match requests.

This is verified during the testing of the matching service:

✓ Create match1 + match1 should be removed within 3 seconds after the 30 seconds timeout (33020ms)  
✓ Create match4 and match5 (of different difficulty) + match4 and match5 should time out (32048ms)  
✓ Create match8 and match9 (of same difficulty & 3 seconds after match8's timeout) + match8 and match9 should not be paired together (64064ms)

*Figure 2.3.4.2: Test outputs for NFR-INT-1*

<sup>3</sup> "Pricing." Auth0. Auth0. Accessed November 3, 2022. <https://auth0.com/pricing>.

<sup>4</sup> "Performance Tuning." SocketIO RSS, October 8, 2022. <https://socket.io/docs/v4/performance-tuning/#at-the-socketio-level>.

### 2.3.5. Usability NFRs

S/N	Description	Priority
NFR-USE-1	Users should be able to navigate and use the application with ease.	Low

*Table 2.3.5.1: Usability NFRs table*

### 2.3.6. Availability NFRs

S/N	Description	Priority
NFR-AVA-1	All services (login, user, matching, collab, room, PubSub) must have an uptime of at least 99%.	Medium

*Table 2.3.6.1: Availability NFRs table*

For NFR-AVA-1, according to Auth0<sup>5</sup>, the average availability of certain core services for each month will be at least 99.99%. For other services such as matching, collab, room and PubSub, the socket instance (client-side) will always try to reconnect as long as the browser is opened and when a low-level connection cannot be established. In cases where some docker containers fail due to unexpected errors, it will always try to re-run. This is to ensure the availability of all services as long as the docker is running.

### 2.3.7. Safety NFRs

S/N	Description	Priority
NFR-SAF-1	Application should safeguard against performance anxiety if users are unable to finish the question by providing encouragement.	Medium

*Table 2.3.7.1: Safety NFRs table*

For NFR-SAF-1, we have included encouraging messages during the loading screen, as seen in [Figure 3.3.4](#), to motivate our users and ensure their wellness.

### 2.3.8. Installability NFRs

S/N	Description	Priority
NFR-INS-1	Services should be easy to update and test during development.	Medium
NFR-INS-2	Services should be containerised and easy to deploy.	Medium
NFR-INS-3	Developers should be able to install and access the application easily with macOS, Windows or Linux.	Medium

*Table 2.3.8.1: Installability NFRs table*

---

<sup>5</sup> Auth0. "Service Levels." Auth0 Docs. Accessed November 8, 2022.  
<https://auth0.com/docs/troubleshoot/customer-support/services-level-descriptions>.

For this group of NFRs, Docker Compose was utilised to ensure that each developer will have the same hosting medium to develop and test their code on. In addition, GitHub Actions CI is utilised with docker to ensure that there is no breaking functionality when a feature branch is to be integrated.

### 3. Developer Documentation

#### 3.1. Overall Architecture

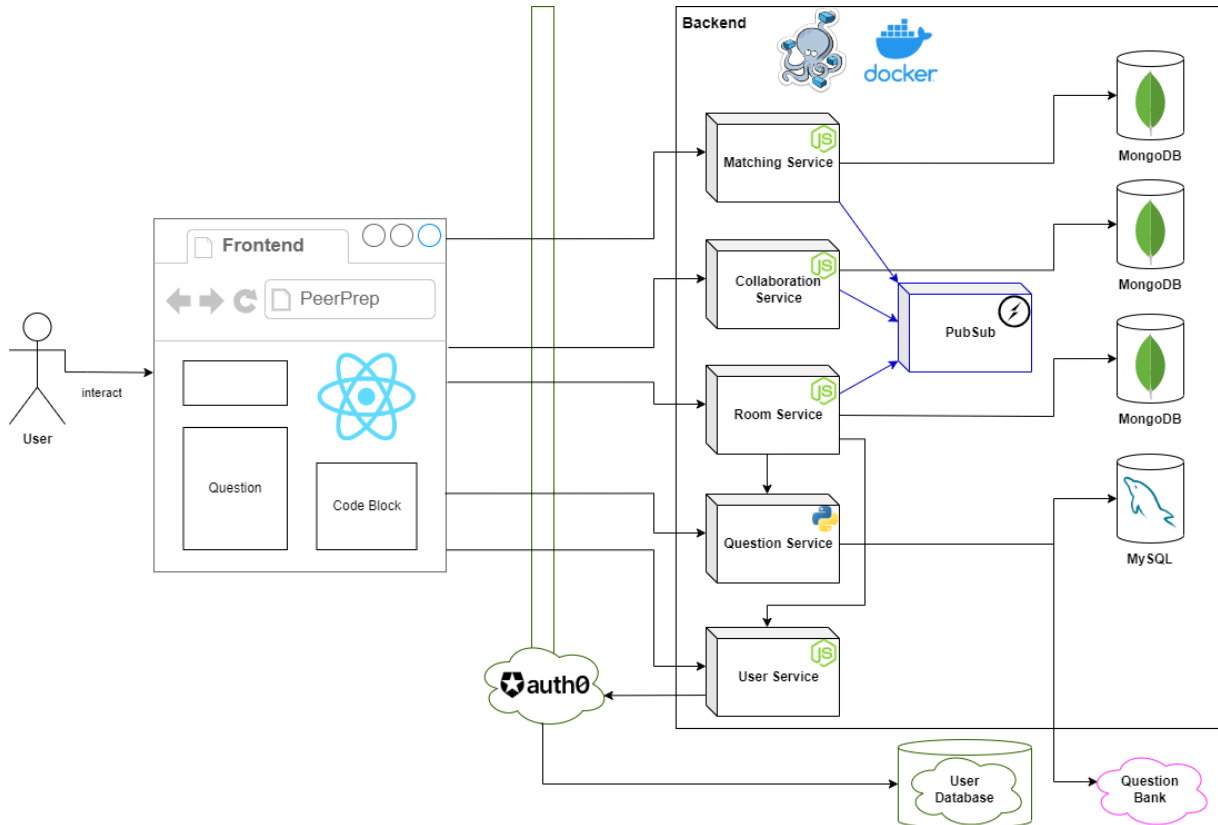


Figure 3.1.1: Overall architecture diagram

Components	Tech Stack
Frontend	React Framework with Axios API calls
Backend	Express.js/Node.js
Database	MongoDB, MySQL
Pub-Sub Messaging	Socket.io
CI/CD	GitHub Actions
Orchestration Service	Docker-Compose
Project Management Tools	GitHub Issues
Authentication Service	Auth0

Table 3.1.2: Tech stack used for each component

### 3.2. Architecture Design Decision

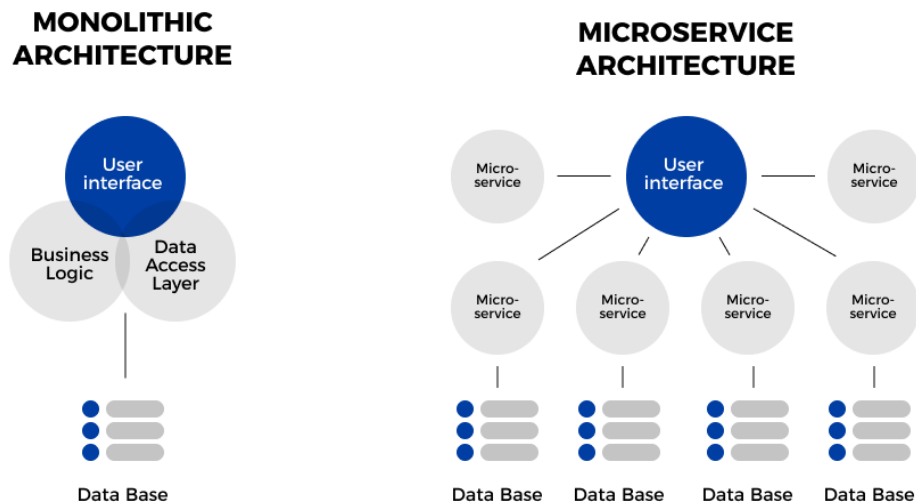


Figure 3.2.1: Monolithic vs Microservices Architecture<sup>6</sup>

Prior to deciding the architecture (Monolithic vs Microservices) of Software Interview Xchange, all functional and non-functional requirements (in [Section 2.2](#) and [Section 2.3](#)) are mapped out. Based on these requirements, potential implications that might be anticipated are drafted below:

Possible implications based on the requirements:
<ol style="list-style-type: none"><li><b>1. Slow development cycle with monolithic architecture</b><ul style="list-style-type: none"><li>○ Everybody in the team needs to coordinate their development and deployment efforts when working on a single large codebase together as it is difficult for multiple teams to work in code simultaneously.</li></ul></li><li><b>2. Frequent deployments get difficult with monolithic architecture</b><ul style="list-style-type: none"><li>○ In order to update one component, we might need to redeploy the entire application.</li></ul></li><li><b>3. Scaling the application can be difficult with monolithic architecture</b><ul style="list-style-type: none"><li>○ Different components have different requirements. With a monolithic architecture, we cannot scale each component independently.</li></ul></li><li><b>4. Some components might be tightly coupled</b><ul style="list-style-type: none"><li>○ Tightly coupled components will be challenging to isolate and test.</li></ul></li><li><b>5. Additional complexity with microservices</b><ul style="list-style-type: none"><li>○ Need to implement inter-process communication mechanism between microservices</li><li>○ Need to implement DevOps tools to automate the processes</li><li>○ Need to manage multiple databases and their transactions</li><li>○ Writing tests involving multiple services will be challenging</li></ul></li><li><b>6. High upfront cost for cloud deployment and using external tools or cloud features</b></li></ol>

Figure 3.2.2: Possible implications based on requirements

<sup>6</sup> Karwata, 2020. "Monolithic architecture vs microservices". Accessed 8 November 2022.  
<https://www.divante.com/blog/monolithic-architecture-vs-microservices>

Architecture styles (mainly Monolithic and Microservices) are then analysed to determine the optimal design to be used for implementation. Below shows a comparison between the mentioned architectural styles.

Architecture Styles	Advantages	Disadvantages
Microservice	<p><b>Loose coupling</b></p> <ul style="list-style-type: none"> <li>● Reduce Implementation coupling</li> <li>● Reduce Domain coupling</li> </ul> <p><b>High system stability and better fault isolation</b></p> <ul style="list-style-type: none"> <li>● Error in one place does not paralyse the entire system</li> <li>● Relatively easy to fix bugs</li> </ul> <p><b>Freedom to choose technology</b></p> <ul style="list-style-type: none"> <li>● Each microservice acts like an application on its own, allowing flexibility to choose technology that is best suited for each component.</li> </ul> <p><b>Modularisation</b></p> <ul style="list-style-type: none"> <li>● Independent microservices are easier to develop, test and maintain due to high cohesion and loose coupling</li> <li>● Teams can work autonomously on their assigned microservice without worrying about what's going on with the rest of the app.</li> </ul> <p><b>Bounded contexts</b></p> <ul style="list-style-type: none"> <li>● Microservices are bounded contexts.</li> <li>● Applies Separation of Concerns (SoC) and Single Responsibility Principle (SRP)</li> </ul>	<p><b>Integrated testing is difficult</b></p> <ul style="list-style-type: none"> <li>● Writing automated tests involving multiple services is challenging.</li> </ul> <p><b>Higher upfront cost</b></p> <p><b>Code duplication due to limited reuse</b></p> <ul style="list-style-type: none"> <li>● It is difficult to share code between loosely coupled microservices.</li> </ul> <p><b>Higher complexity</b></p> <ul style="list-style-type: none"> <li>● Configuration of microservice is usually more complex</li> <li>● Communication between multiple services can be complex</li> <li>● Need to be good in DevOps tools</li> </ul>
Monolith	<p><b>Simple to develop</b></p> <ul style="list-style-type: none"> <li>● Many development tools and IDEs support the development of monolithic applications</li> </ul> <p><b>Simple to deploy</b></p> <ul style="list-style-type: none"> <li>● Simply deploy the WAR file (or directory hierarchy)</li> </ul> <p><b>Simple to scale</b></p> <ul style="list-style-type: none"> <li>● By running multiple copies of the application behind a load balancer</li> </ul>	<p><b>Single large codebase</b></p> <ul style="list-style-type: none"> <li>● The large monolithic code base intimidates developers, especially ones who are new to the team.</li> <li>● The teams must coordinate their development efforts and redeployments</li> </ul> <p><b>Frequent deployments get difficult</b></p> <ul style="list-style-type: none"> <li>● In order to update one component, you have to redeploy the entire application. The risk associated with redeployment increases, which discourages frequent updates.</li> </ul>

		<p><b>Scaling the application can be difficult</b></p> <ul style="list-style-type: none"> <li>Each copy of the application instance will access all of the data, which makes caching less effective and increases memory consumption and I/O traffic.</li> <li>Also, different application components have different resource requirements. With a monolithic architecture, we cannot scale each component independently.</li> </ul> <p><b>Tight coupling</b></p> <ul style="list-style-type: none"> <li>Maintenance, integration, testing and reuse of modules can be difficult.</li> </ul>
--	--	--

*Table 3.2.3: Advantages and disadvantages of Monolithic and Microservices architectures*

Based on the implications and comparison of the advantages and the disadvantages of the two architectures, we decided to go with the Microservices architecture. As a team, it has been decided that it is essential for each member to focus on their assigned components without having to worry about what's going on with the rest of the application so that the team can speed up the development process.

The table below also illustrates several design patterns and principles utilised in the design and implementation of the application.

<b>Design Principles &amp; Patterns</b>	<b>Nature</b>	<b>Rationale &amp; Implementation</b>
Separation of Concerns	Domain management	By considering the business requirements, domains can be extracted and managed by relevant services.
Decentralised governance	Design management	<p>Separating domains required by the application also allows independent management and design of services - with the added benefit of parallel development.</p> <p>This allows more specialised tools to be utilised to meet their respective requirements. An example would be utilising BeautifulSoup in the question service for question scraping and storage.</p> <p>Additionally, to guarantee correctness and smooth operations, test cases are to be devised. Hence, having modular services improves testing implementation and service refinement as compared to a highly coupled system/monolith.</p>
Per service database	Data management	Allowing separate database management per service improves overall performance and reliability.



Facade	Structural management	To prevent heavy coupling between services, a top layer service is provided for each microservice to allow external services to communicate through a centralised interface.
Observer (push model)	Behavioural management	Using the PubSub mechanism allows different services to update each other regarding the state of ongoing events.
Usage of DTOs	Data object management	After performing API discovery, designing abstractions through DTOs allows services to be coupled to their own explicit interpretation of objects, but rather to extract relevant information from an agreed shared data template.
Docker Compose	Deployment & Orchestration	As the number of services and codebase significantly increases due to the chosen microservice architecture, a robust deployment tool should be utilised to smoothen development and system testing. As such, docker-compose is the first step towards working towards a larger goal to deploy through Kubernetes and NGINX (elaborated in Sections 3.5 and 5.2) to increase the scalability of the application.

*Table 3.2.4: Design principles & patterns table*

### 3.3. Frontend

The frontend application utilises React.js to provide a simplistic Single Page Application (SPA). Using React.js has its benefits as it has wide documentation for hooks/useEffect/useState/route and is easily extensible with other Javascript libraries. With the additional Material UI components and bootstrap, the application is able to be presented in a sleek and clean manner.

Another reason as to why React.js was chosen is due to its Web-MVC design pattern. As the application will be heavily driven with numerous data sources/services from the backend, there needs to be a middle layer to process and represent the data in a meaningful manner to the user. React.js utilises the MVC pattern that modularised state from processing user views, making it capable of working seamlessly with backend server applications.

The overall frontend application provides the following pages:

- Welcome/Dashboard page
- Login/Account creation page
- User profile page
- Training selection page
- Loading page
- Room page
- History page

A typical user flow on SIX is as such. A new user will be first greeted by the welcome screen.

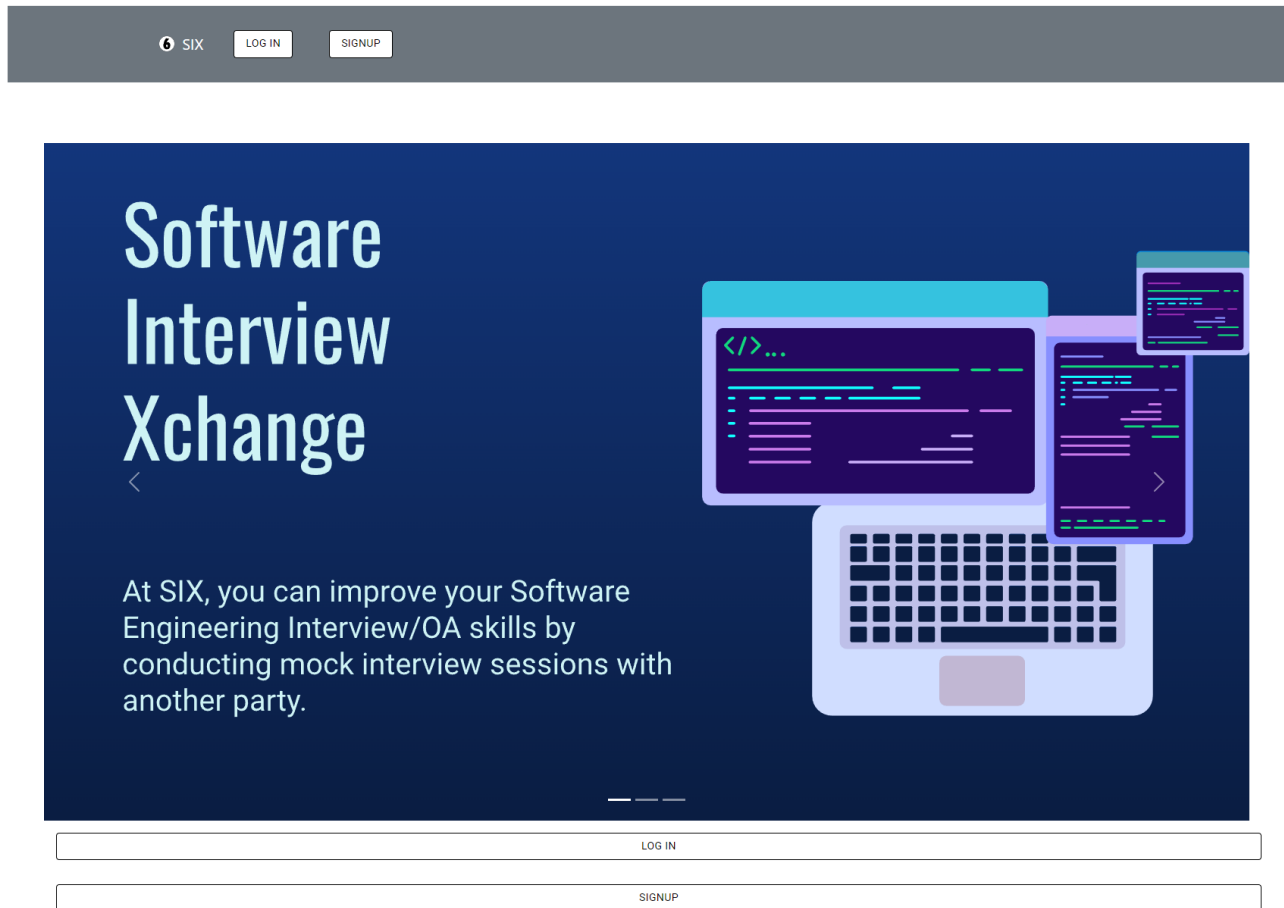


Figure 3.3.1: Welcome page of SIX

The user then creates an account to log in.

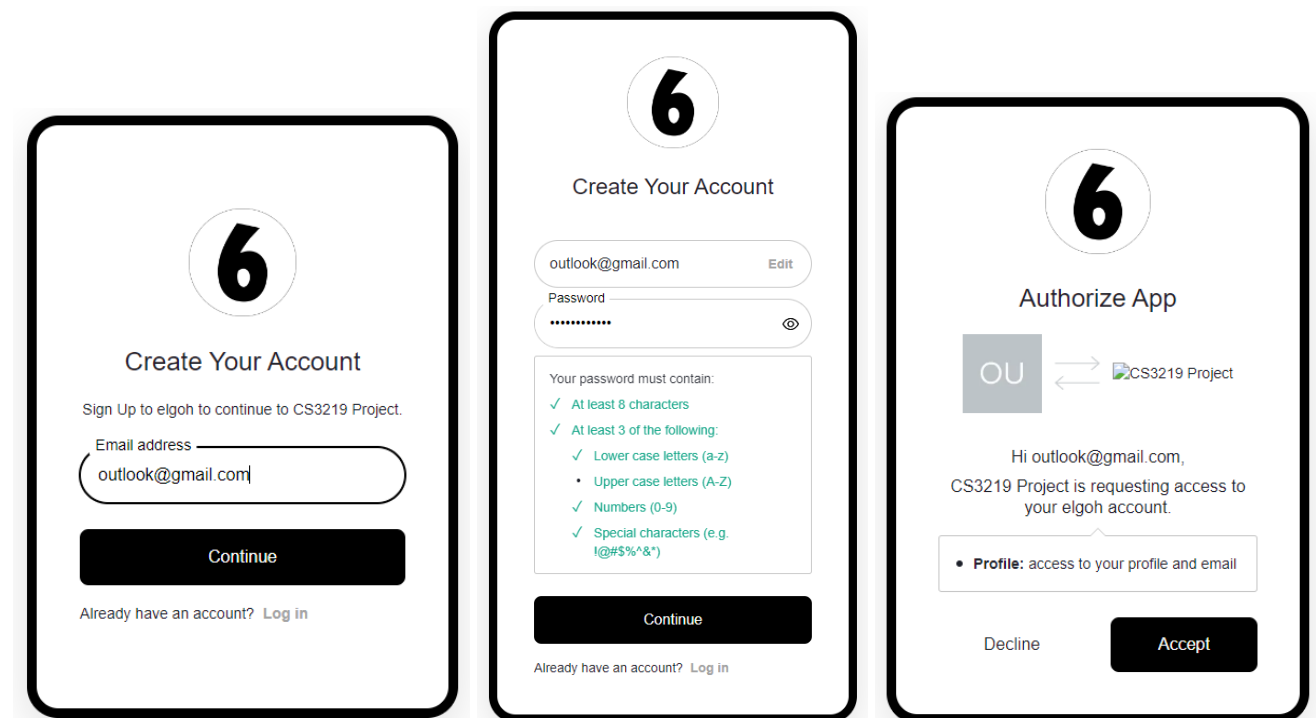
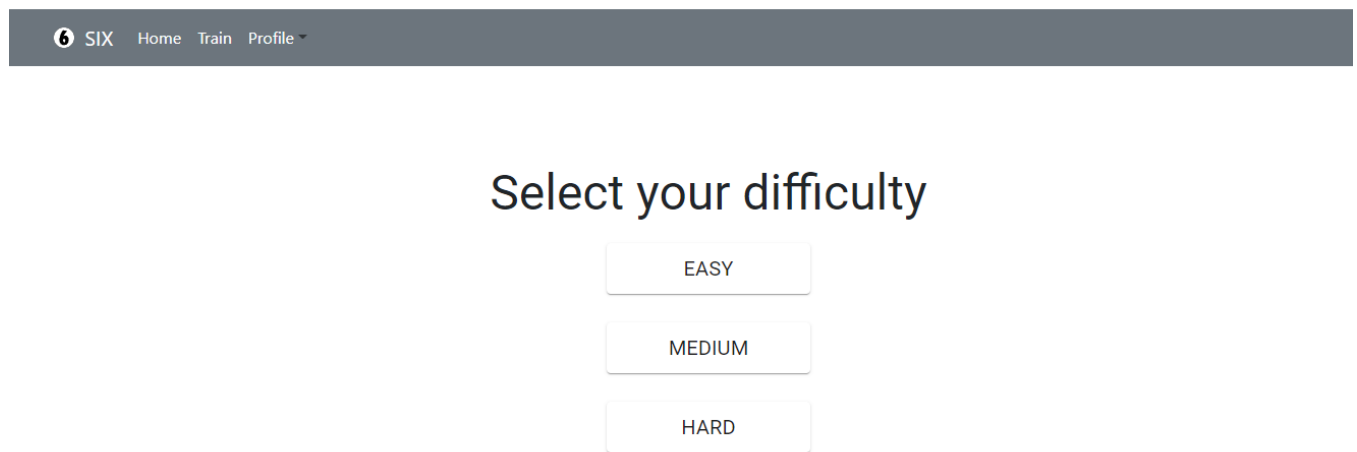


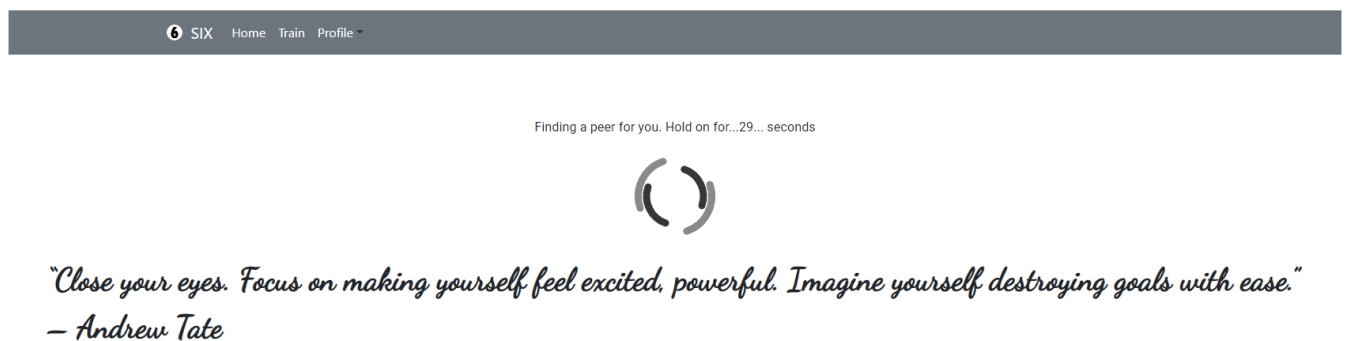
Figure 3.3.2: Account creation

The user can start training by selecting a difficulty.



*Figure 3.3.3: Difficulty selection*

The user then waits for a match and then gets sent to a room with a peer and a question of selected difficulty.



*Figure 3.3.4: Loading page*

Coding session with outlook

# Same Tree

[PROBLEM](#) [SOLUTION](#)

Given two binary trees, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical and the nodes have the same value.

**Example 1:**

```

**Input:**      1      1
               / \   / \
              2  3 2  3
             [1,2,3], [1,2,3]

**Output:** true

```

**Example 2:**

```

**Input:**      1      1
               /       \
              2         2
             [1,2],    [1,null,2]

**Output:** false

```

**Example 3:**

```

**Input:**      1      1
               / \   / \
              2  1 1  2
             [1,2,1], [1,1,2]

**Output:** false

```

```

Btree() {
    private Btree parent;
    private Btree leftChild;
    private Btree rightChild;

    Btree(){

```

Figure 3.3.5: Room page

The user and peer can type collaboratively, and can request to reveal the solutions tab.

Coding session with outlook

# Same Tree

[PROBLEM](#) [SOLUTION](#)

Solution locked. Try the question first before unlocking this tab!

```

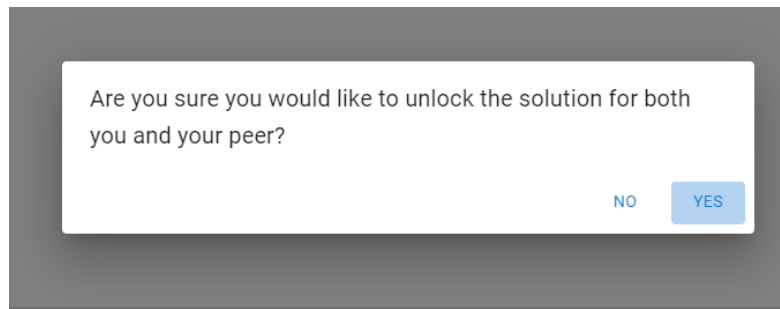
Btree() {
    private Btree parent;
    private Btree leftChild;
    private Btree rightChild;

    Btree(){

```



REVEAL SOLUTION



Coding session with outlook

# Same Tree

PROBLEM SOLUTION

## Question

Formatted question description: <https://leetcode.ca/all/100.html>

100 Same Tree

Given two binary trees, write a function to check if they are equal or not.

Two binary trees are considered equal if they are structurally identical and the nodes have the same value.

@tag-tree

## Algorithm

Depth first search DFS to recurse.

## Code

Java

- [Java](#)
- [C++](#)
- [Python](#)
- 

```
public class Same_Tree {
    /\^*\*
    * Definition for a binary tree node. * public class TreeNode { * int val; * TreeNode left; * TreeNode right; * TreeNode(int x) { val = x; } * } */
    public class Solution\iteration {
        public boolean isSameTree(TreeNode p, TreeNode q) {
            if (p == null) {
                return q == null;
            }
            if (q == null) {
                return p == null;
            }
            Stack<TreeNode> sk1 = new Stack<TreeNode>();
            Stack<TreeNode> sk2 = new Stack<TreeNode>();
```

Figure 3.3.6: Reveal solution prompt and solutions tab

Upon both users leaving the room, the session will then be closed. A user can then review their previous sessions under the history page.

SIX Home Train Profile

View Profile

User History

Change Password

LOG OUT

History

Date	Peer name	Question title	Ongoing	Link
Wed, 09 Nov 2022 06:41:46 GMT	elgoh2	Same Tree	Yes	<a href="#">Question link</a>

Figure 3.3.7: User history page

Finally, a user can choose to edit their username, change their password, or close their account in the profile page.

The image shows two side-by-side screenshots of a web application interface. The left screenshot is the 'User profile page' and the right is the 'change password page'. Both pages have a dark grey header with a logo and navigation links: 'SIX', 'Home', 'Train', and 'Profile'. The profile page on the left features a grey square with 'OU' in white, the nickname 'outlook', an email input field with 'outlook@gmail.com', and buttons for 'Update nickname', 'Change password', and 'Delete My Account'. The change password page on the right has a light green box for 'Password requirements' (labeled 'Must fulfill') listing rules like 'At least 8 characters' and 'At least 3 of the following: Lower case letters, Upper case letters, Numbers, Special characters'. Below this are input fields for 'New password' and 'Confirm new password', and a 'Change password' button.

Figure 3.3.8: User profile page and change password page

### 3.4. Backend

For the backend service, each context is bound to its own service. The table below describes each unit and its purpose in SIX.

Backend Service	Description	Tech Stack
User	Performs user-specific operations - such as changing users' nicknames and passwords	<ul style="list-style-type: none"> <li>Express.js/Node.js</li> <li>Auth0</li> </ul>
Matching	Performs matching operation between 2 users with the same difficulty preference and provides redirection upon success.	<ul style="list-style-type: none"> <li>Express.js/Node.js</li> <li>Socket.io</li> <li>MongoDB</li> </ul>
Room	Performs room creation for a given pair of users and difficulty and provides an avenue to load details of a given room and its history.	<ul style="list-style-type: none"> <li>Express.js/Node.js</li> <li>MongoDB</li> </ul>
Collaboration	Hosts rooms for the users' pairings by maintaining communications and code between the users in the same room.	<ul style="list-style-type: none"> <li>Express.js/Node.js</li> <li>Socket.io</li> <li>MongoDB</li> </ul>
Question	Extracts questions and solutions from a question source and provides endpoints for random question selection and data retrieval.	<ul style="list-style-type: none"> <li>FastAPI/Python</li> <li>MySQL</li> </ul>
PubSub	Provides an avenue for external services to subscribe to the desired topic and publish data with a chosen topic.	<ul style="list-style-type: none"> <li>Express.js/Node.js</li> <li>Socket.io</li> </ul>

Table 3.4.1: Backend services summary table

Throughout implementation, several design decisions that affect multiple backend services were made. These are elaborated in Table 3.4.2.

Design Decision	Related Systems	Justification	Alternatives & rationale for not choosing them
Splitting matching and room-related logic	Matching, Room	After reviewing the boundary conditions for matching and room operations, it would be more reasonable to split these services to declutter the code base. Additionally, as matches require to be created and deleted, it would incur more database operations as compared to just creating a room pairing. This would allow extensibility on scalability and potential features down the line.	<ul style="list-style-type: none"> <li>• <i>Combining said services with multiple database schemas:</i> Would be easier to implement, but is not able to scale with respect to deliverable features and scalability</li> </ul>
Splitting room and collaboration related logic further	Room, Collaboration	The desired operations for a room could be split further to creation/loading and updating. Creation and loading of rooms could be done by a separate entity, which would not be called as frequently as an entity that would host collaboration services with users linked with socket communications. Such collaboration requires frequent database updates and bilateral communication. This separation of concerns will allow better modularity and allocation of resources when scaling.	<ul style="list-style-type: none"> <li>• <i>Combining said services:</i> Easier implementation, but could have reduced performance overall due to the system requiring to perform database updates and creation/endpoint management.</li> </ul>
Utilising a publisher-subscriber (PubSub) messenger mechanism	Matching, Room, Collaboration, PubSub	As there is a need to pass states of pairings and rooms between services and such services must operate asynchronously, a messaging system is required. At the same time, there could be multiple listeners required for a desired state nature.	<ul style="list-style-type: none"> <li>• <i>RESTful API endpoints:</i> Easier to implement but that would require wrapping state in stateless operations. Additionally, having services expose endpoints for communication can lead to heavy coupling and exploitations from external users.</li> </ul>

Decomposition using ports and adapters & slicing	All	When implementing the codebase, it was observed that the responsibilities of specific classes could be classified in a manner that separates domain-related code from technology. Additionally, it can also be divided by features required by the service - such as model operations and socket communications. Facade interfaces are also employed at each service endpoint to prevent unwanted coupling of control. This increases overall cohesion within the codebase.	<ul style="list-style-type: none"> <li><i>Lump codebase without hierarchy foldering:</i> Would be easier to implement in initial phases, but would not scale well and can cause the ‘Big-bang’ scenario when merging features with other developers.</li> </ul>
--	-----	---	---

Table 3.4.2: Design decisions table

The following sections describe the architecture, design and flow of operations for each backend service.

### 3.4.1. User Service

For the user service, the team has decided to utilise Auth0 as the user management rather than an in-house internal service. The table below provides a rationale for the usage of Auth0 by comparing it with an in-house solution.

Criteria	Auth0 (third-party)	In-house
<b>User creation</b>	Service provided by Auth0, only requires extension through frontend elements. Email verification is also sent to the user upon account creation.	Manual verification is needed for account uniqueness and email validation. Additionally, password storage would require manual salting and hashing.
<b>User management</b>	Auth0 client manager and the library are available and have extensive documentation to provide user updates.	Manual logic is required to manage users.
<b>User extensibility</b>	Auth0’s user database allows extended fields for a user to hold onto. Can also utilise a local database with an Auth0 listener to tap into for a larger user definition (user profile picture etc.)	User details are held in a local/hosted database. Requires updates in schemas to implement various extensions for user definition.
<b>Authentication</b>	JSON Web Tokens (JWT) generated by Auth0 using RSA256. Auth0’s client manager is able to verify such tokens, and available libraries are able to decode it and perform other verifications (e.g. user id checks for a given resource).	Token, signature and verification are to be done with local libraries.



<b>User scalability</b>	Auth0 handles the scaling of User service such as database management and server availability	To scale the system, a lot of manual work has to be done.
<b>Decision: Auth0</b>	On the security front, passwords are able to be salted and hashed automatically using Auth0's Infrastructure as a Service platform, keeping the internal codebase more maintainable by working with Auth0 service abstraction. User Service can then be easily extended with the frontend by bouncing pages with Auth0 that manages the JWT token sessions.	

Table 3.4.1.1: Comparison between Auth0 and an in-house user manager

Additionally, with Auth0, the login service can be easily extended to include SSO login and third-party login using Google, Facebook, etc. users' information is stored and secured by Auth0. Below shows a sample JWT created by Auth0 with a test user, as well as its encoded information. Note the sub field (representing subject) which holds the user id, which can be utilised by other services for resource permissions.

The screenshot shows the JWT.io website interface. At the top, there's a navigation bar with links: Debugger, Libraries, Introduction, Ask, and a 'Crafted by auth0 by Okta' logo. The main content area is split into two panels: 'Encoded' on the left and 'Decoded' on the right. The 'Encoded' panel has a text input field containing a long string of characters. The 'Decoded' panel shows the token's structure with a header, payload, and secret. The header is 'ALGORITHM & TOKEN TYPE' and the payload is 'DATA'. The payload contains a JSON object with fields: 'alg' (RS256), 'typ' (JWT), 'kid' (ilpocS1mCKW69xZZmqcI3), 'iss' (https://elgoh.us.auth0.com/), 'sub' (auth0|634d6b07ec3310e32ea015b), 'aud' (https://user-service.com, https://elgoh.us.auth0.com/userinfo), 'iat' (1667915421), 'exp' (1668001821), 'azp' (3CVQjahdc8e1z1cUdqsZMw15FfhqylxW), 'scope' (openid profile email), and 'permissions' ( []).

Figure 3.4.1.2: Sample decoded JWT generated by Auth0<sup>7</sup>

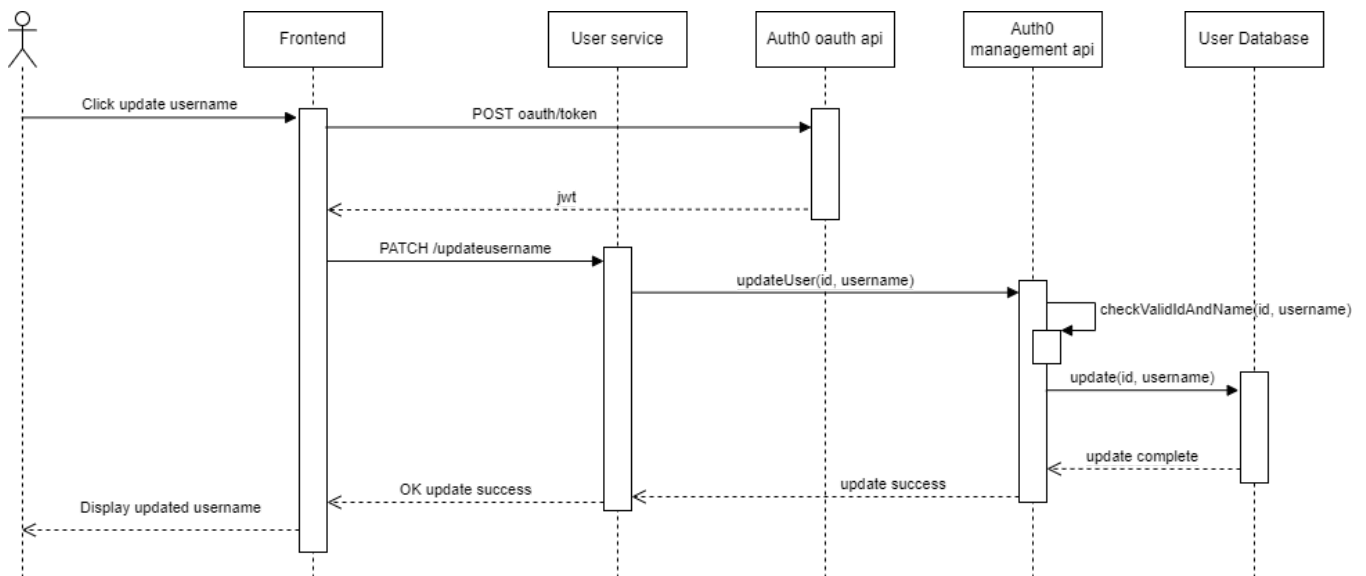
Within the frontend service, Auth0 is integrated to provide the (1) user creation, (2) user login, (3) user password reset option, (4) user logout and JWT generation for authentication with other services. Apart from those operations, the following shows the operations/APIs exposed from the user service for other services to utilise.

<sup>7</sup> Auth0. "JWT.io" Accessed November 8, 2022. <https://jwt.io/>

HTTP		Purpose	DTOs	
Method	Path		Input	Output
GET	/api/user/delete	To delete a user from the application	-	[raw] true/error
PATCH	/api/user/updateusername	To update the username/nickname of a user	[json] id: UUID, nickname: str	[raw] true/error
PATCH	/api/user/changepassword	To update the password of a user	[json] id: UUID, password: str	[raw] true/error
GET	/api/user/username/:user_id	To get the user particulars based of the id	-	nickname: str

*Table 3.4.1.3: User service endpoints*

The sequence diagram below describes the flow of events during a user-specific update.



*Figure 3.4.1.4: User update sequence diagram*

The following table shows the software requirement specification for this component.

<b>Name</b>	User Service		
<b>Description</b>	Receive user request to change particulars and perform it through Auth0 management API		
<b>Capabilities</b>			
Allow users to change their nickname Allow users to change their password			
<b>Service API</b>			
<i>Send Events (to Auth0 management API)</i>		<i>Receive Events</i>	
<ul style="list-style-type: none"><li>• DELETE_USER(userId: string)</li><li>• UPDATE_USER(userId: string, newUserName: string)</li><li>• UPDATE_USER(userId: string, newPassword: string)</li></ul>		<ul style="list-style-type: none"><li>• REMOVE_USER(userId: string)</li><li>• UPDATE_USER_NAME(userId: string, newUserName: string)</li><li>• UPDATE_USER_PASSWORD(userId: string, newPassword: string)</li></ul>	
<b>Non-functional requirements</b>	<ul style="list-style-type: none"><li>• User must be authenticated to make a request to the User Service</li><li>• Communication with User Service must contain valid JWT token</li><li>• User Service must be able to handle 100 requests at once</li></ul>		

*Table 3.4.1.5: User service SRS Table*

### 3.4.2. Matching Service

For the matching service, we decided to implement the database using MongoDB. We decided not to use the recommended SQLite because we find it much easier to set up the ORM using MongoDB as compared to Sequelize. On top of that, all of us have prior experience working with MongoDB and it is much easier and faster for us to work with MongoDB given the time constraint. However, SQLite has slightly better performance than MongoDB when it comes to real-time matching. Even so, we think that the performance of MongoDB should be sufficient to handle the non-functional requirements that we have set out for the matching service.

Also, since our use cases for the matching service involve creating a match within thirty seconds and informing both parties that a match has been made in real time, we require a Pub-Sub mechanism. For this, we chose to implement the mechanism using socket.io. The main reason why we chose socket.io, other than that this is the recommended tech stack, is the Rooms feature which allows us to broadcast relevant events to the two paired users only.

The following diagram describes the architecture of the service.

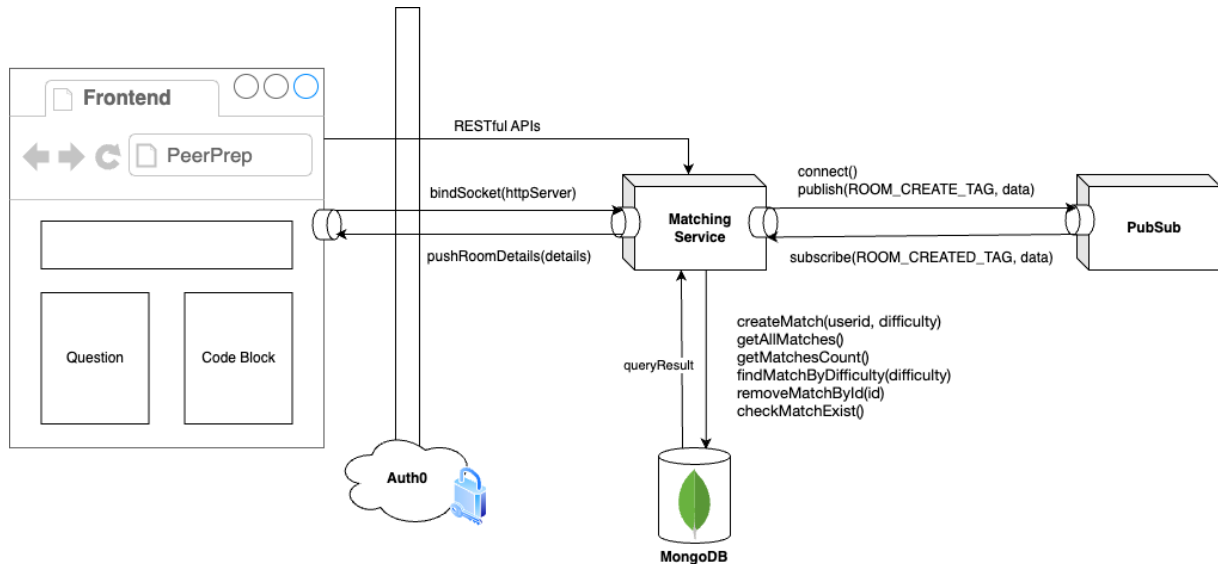


Figure 3.4.2.1: Matching service architecture diagram

The activity diagram below shows the flow of events provided by the matching service.

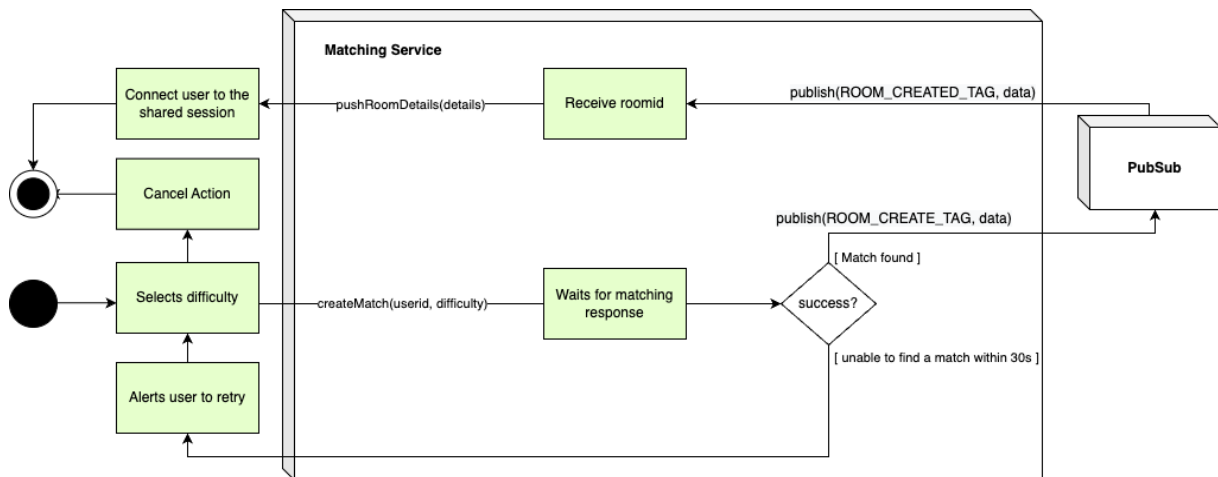


Figure 3.4.2.2: Matching service activity diagram

The matching service utilises the following schema in MongoDB, paired with the mongoose library available for node.js. The socketid field is present so as to allow lookup of a user's waiting socket room to forward a successful pairing message.

```
Match {
  userid: UUID,
  difficulty: ENUM(easy, medium, hard),
  socketid: UUID
}
```

Figure 3.4.2.3: Match model schema on MongoDB

The following shows the endpoints available for matching service.

HTTP		Purpose	DTOs	
Method	Path		Input	Output
POST	/api/match	Perform a match given options specified	[json] difficulty: ENUM(easy, medium, hard)	[json] matchId: UUID

*Table 3.4.2.4: Matching service endpoints*

The following shows the socket and PubSub operations present for matching service.

Operation	Room	Purpose	Data
pushRoomDetails	2 waiting users based of waiting room id	To redirect user sockets to their collaboration sockets	[raw] roomId: UUID

*Table 3.4.2.5: Matching service socket details*

Subscriptions		Publications	
Event Tag	Handler	Event Tag	Data
ROOM_CREATED	Publish room details to waiting socket rooms	ROOM_CREATE	[json] userId1: UUID, userId2: UUID, difficulty: ENUM(easy, medium, hard)

*Table 3.4.2.6: Matching service PubSub details*

The following table shows the software requirement specification for this component.

<b>Name</b>	Matching Service	
<b>Description</b>	The Matching Service tries to find two separate users with the same difficulty preference within the time limit of 30 seconds and redirects them to the room if successful.	
<b>Capabilities</b>		
Matching users who chose the same question difficulty to the same room.		
<b>Service API</b>		
<i>Send Events</i>		<i>Receive Events</i>
<ul style="list-style-type: none"><li>PUBLISH_ROOM_CREATE(user1:UUID, user2: UUID)</li></ul>		<ul style="list-style-type: none"><li>PUSH_ROOM_DETAILS(room_id: UUID)</li></ul>

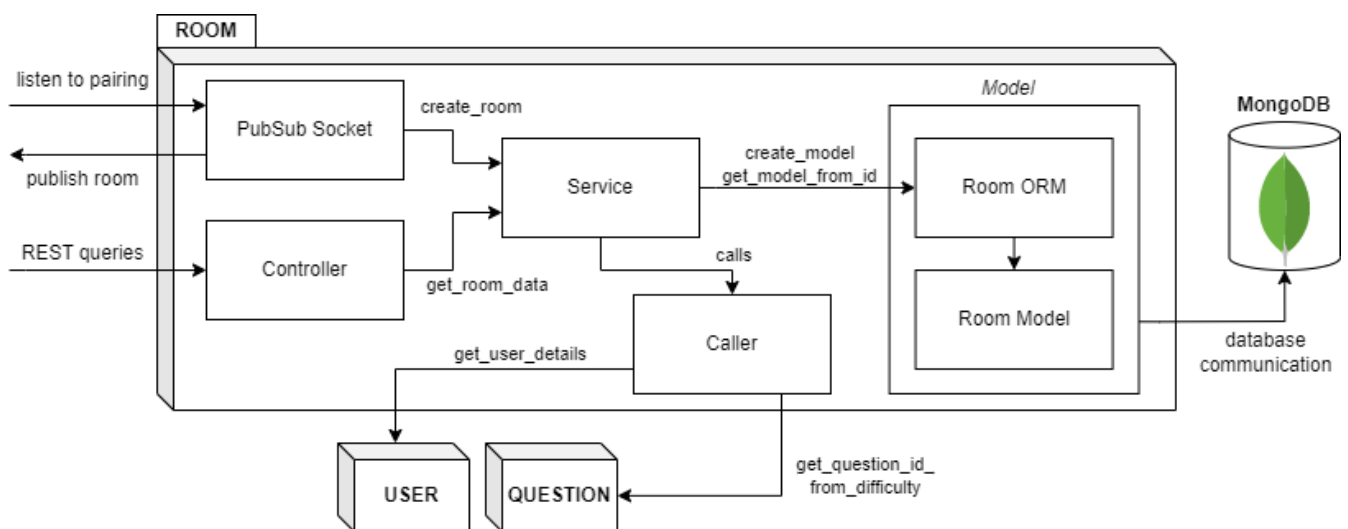
<b>Non-functional requirements</b>	<ul style="list-style-type: none"> <li>• The service should have at least 99.9% uptime/availability.</li> <li>• The service should support up to 1,000 private sessions at any point in time.</li> <li>• When the service is busy with match requests, it should match all of them within 10 seconds.</li> <li>• When the service is not busy, it should match 2 users who have yet to time out and pick the same difficulty within 3 seconds.</li> <li>• The service should remove the timed-out match within 3 seconds.</li> </ul>
------------------------------------	--

*Table 3.4.2.7: Matching service SRS table*

### 3.4.3. Room Service

A room service is designed to ensure room creation and room maintenance are met when a user gets matched and interacts with others. For implementation details, MongoDB is used for entity management, socket.io is used for pub-sub communication with matching-service, and RESTful API endpoints are used for users to load initial room details (such as peer and question details). This design allows the separation of concerns between the different microservices.

The following describes the architecture of the service.



*Figure 3.4.3.1: Room service class diagram*

The sequence diagrams show the flow of events provided by the service. The first sequence diagram shows the flow when it receives a CREATE\_ROOM event (broadcasted from PubSub). Note that a CREATED\_ROOM event will be published out through PubSub - this will allow the matching service and collab service to act accordingly. The caller service then calls the question service to obtain a random question ID based on the difficulty selected.

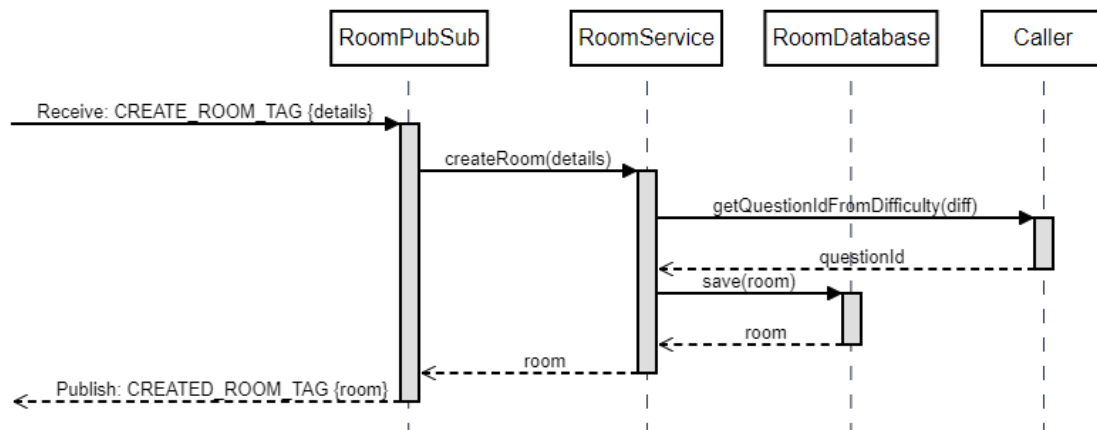


Figure 3.4.3.2: Create room sequence diagram

A GET request to obtain the room details based on a room ID. This is normally called by the Frontend to help render the various components in the collaboration page (such as user/peer particulars and question ID).

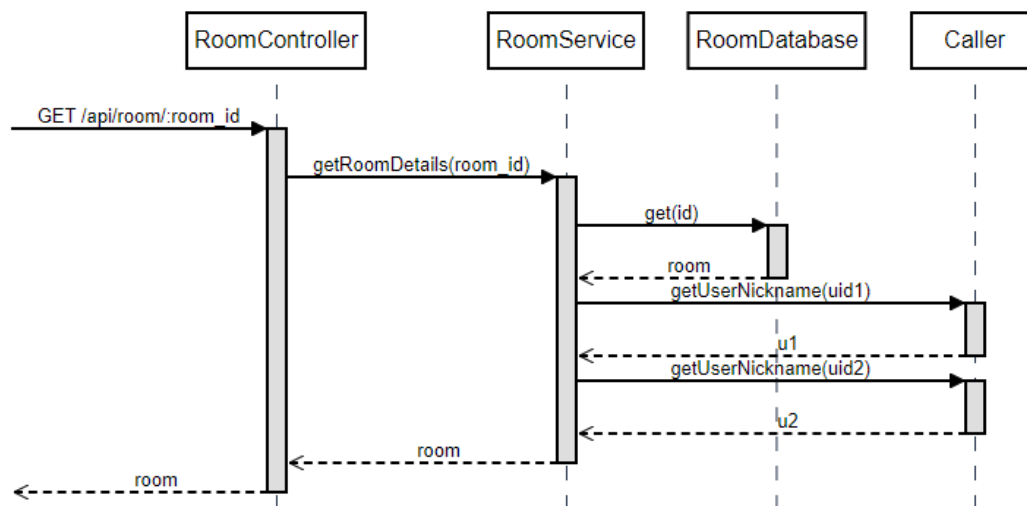


Figure 3.4.3.3: Load room details sequence diagram

Another GET request can be done to obtain the room details a specific user has participated in. Internally the service calls other services through endpoint calls to craft a detailed response back to the user. This endpoint is used by the front end to load the user's history.

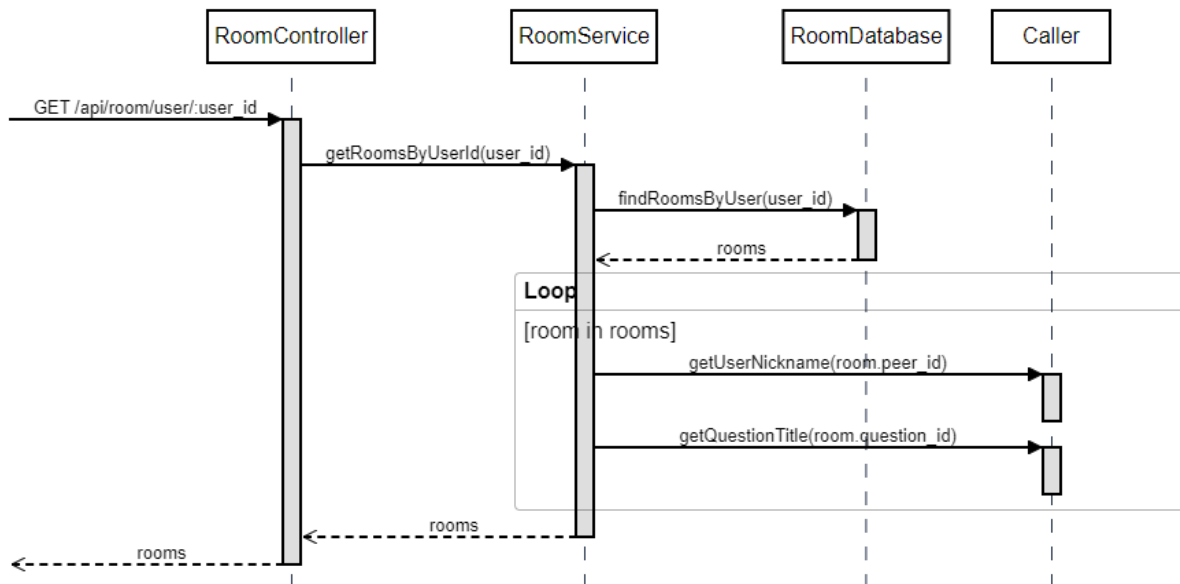


Figure 3.4.3.4: Load user room history sequence diagram

The room service utilises the following schema in MongoDB, paired with the mongoose library available for node.js. The question id field is obtained from calling the question service for a random id based on difficulty, and is utilised for user history features.

```

Room {
  userid1: UUID,
  userid2: UUID,
  questionid: UUID
}
  
```

Figure 3.4.3.5: Room model schema on MongoDB

The following shows the endpoints available for room service.

HTTP		Purpose	DTOs	
Method	Path		Input	Output
GET	/api/room/:room_id	Obtain immediate details regarding a room.	-	[json] userid1: UUID, userid2: UUID, questionid: UUID
GET	/api/room/user/:user_id	Obtain all details of a user's past and ongoing room/collaborations.	-	[json] rooms: list( roomId: UUID, startDateTime: time, endDateTime: time?, peerNickname: str, questionTitle: str )

Table 3.4.3.6: Room service endpoints



The following shows the socket and PubSub operations present for room service.

Subscriptions		Publications	
Event Tag	Handler	Event Tag	Data
ROOM_CREATE	Create a room instance based on supplied parameters.	ROOM_CREATED	[json] roomId: UUID, userId1: UUID, userId2: UUID

*Table 3.4.3.7: Room service PubSub details*

The following shows the software requirement specification for this component.

<b>Name</b>	Room Service		
<b>Description</b>	Creates a collaboration room & maintains communication between users in the room.		
<b>Capabilities</b>			
Creating a room for a given pair of users and difficulty. Also provides an avenue to load details of a given room and its history.			
<b>Service API</b>			
<i>Send Events</i>		<i>Receive Events</i>	
<ul style="list-style-type: none"><li>• PUBLISH_ROOM(user1: UUID, user2: UUID, roomId: UUID)</li><li>• REQUEST_QUESTION_ID(difficulty: ENUM)</li><li>• REQUEST_QUESTION_DETAILS(question Id: UUID)</li><li>• REQUEST_USER_DETAILS(userId: UUID)</li></ul>		<ul style="list-style-type: none"><li>• CREATE_ROOM(user1: UUID, user2: UUID, difficulty: ENUM)</li><li>• GET_ROOM_DETAILS(roomId: UUID)</li><li>• GET_USER_ROOMS(userId: UUID)</li></ul>	
<b>Non-functional requirements</b>	<ul style="list-style-type: none"><li>• The service should have at least 99.9% uptime/availability.</li><li>• The service should be able to receive and respond to PubSub events in under 3 seconds.</li><li>• The service should be able to respond to API endpoints within 3 seconds.</li></ul>		

*Figure 3.4.3.8: Room service SRS table*

### 3.4.4. Collaboration Service

A collaboration service is designed to ensure paired users are able to interact with each other. For implementation details, MongoDB is used for entity management, socket.io is used for pub-sub communication with matching-service as well as user-room interaction, and RESTful API endpoints are used to retrieve documents for attempt history. This design allows the separation of concerns between the different microservices.

The following describes the architecture of the service.

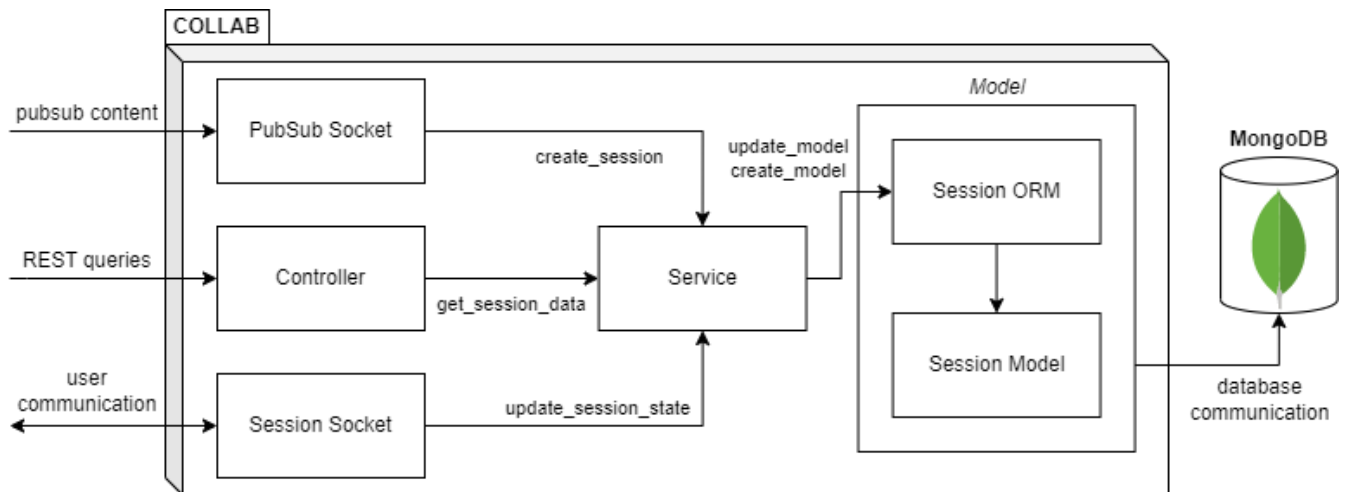


Figure 3.4.4.1: Collaboration service class diagram

A session is created once a room has been created and published under the CREATED\_ROOM event. This then triggers the service to create a new session, which can then be loaded by the users for collaboration. A session includes open and close timestamps as well as the code text inputted by users.

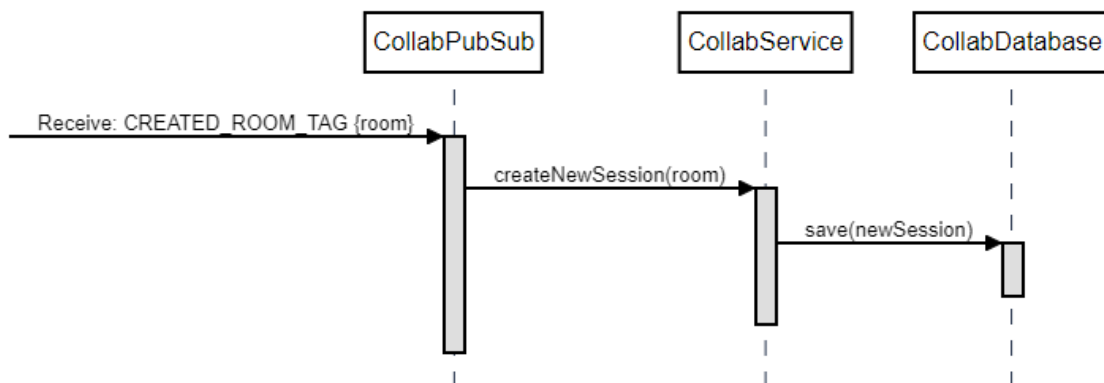


Figure 3.4.4.2: Session creation sequence diagram

When a user joins through socket communications, the collab socket service will redirect the user socket into its specific room id, and load the current code/data status of the session. From which when a user codes in the code box, the document gets updated and saved. Once the collab socket detects that both users have left the session, it automatically closes the session - preventing it from being updated anymore. This updated status can then be called by other services to load its state, with the user history being one such example.

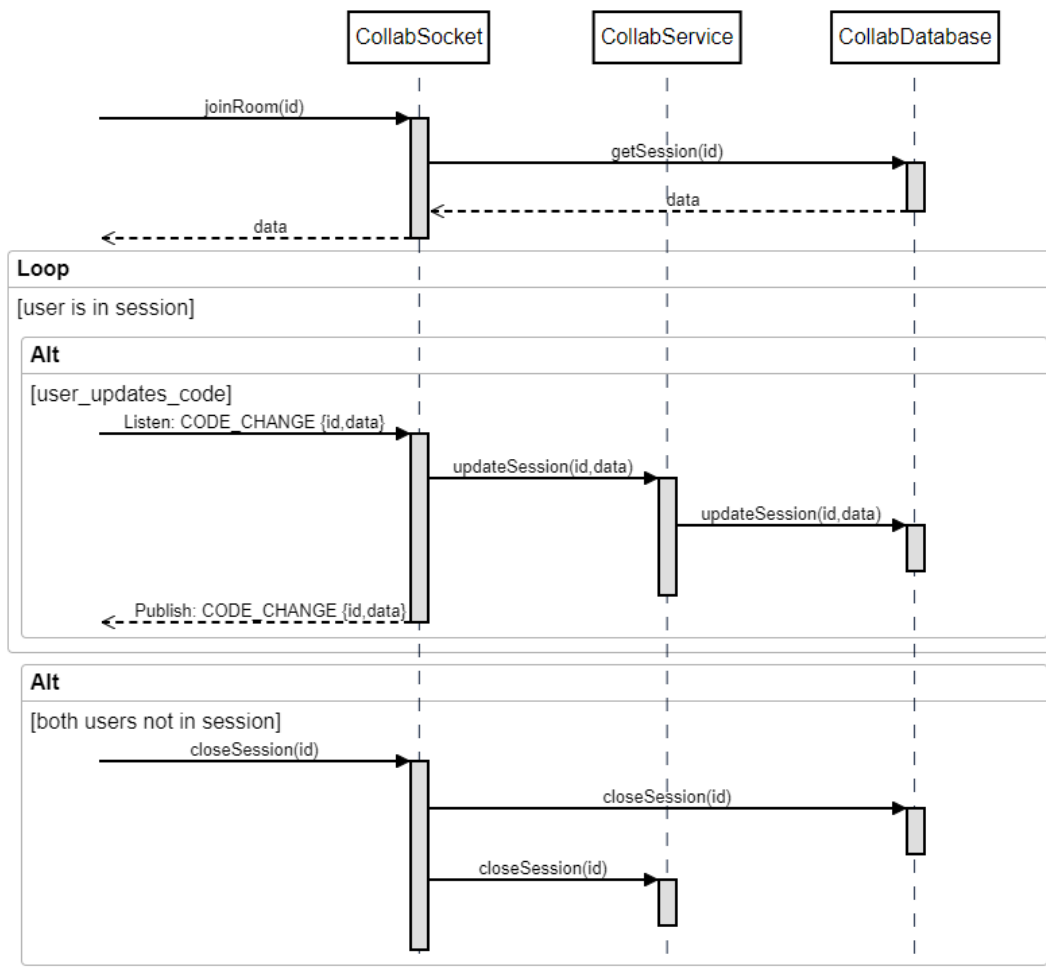


Figure 3.4.4.3: Collaboration service sequence diagrams

The collaboration service utilises the following session schema in MongoDB, paired with the mongoose library available for node.js. Timestamps are also enabled for this schema to track the history of operations performed.

```

Session {
  roomid: UUID,
  document: str,
  isOpen: boolean
}
  
```

Figure 3.4.4.4: Session model schema on MongoDB

The following shows the endpoints available for collaboration service.

HTTP		Purpose	DTOs	
Method	Path		Input	Output
GET	/api/session/:room_id	Obtain the session present for a given room.	-	[json] roomid: UUID, document: str, isOpen: boolean

PUT	/api/session/:room_id	Forcefully closes the session.	-	-/error
-----	-----------------------	--------------------------------	---	---------

*Table 3.4.4.5: Collab service endpoints*

The following shows the socket and PubSub operations present for collaboration service.

Operation	Room	Purpose	Data
get-code	Users in the same collaboration room.	To load code from a given session	[raw] document: str
send-code		To update code inputted by a user socket, and broadcast changes to the other peers present	[json] roomId: UUID
save-code		To save code periodically based on a given session	[json] roomId: UUID

*Table 3.4.4.6: Collab service socket details*

Subscriptions		Publications	
Event Tag	Handler	Event Tag	Data
ROOM_CREATED	Creates a session document based on the room id to allow collaboration between incoming peers.	-	-

*Table 3.4.4.7: Collab service PubSub details*

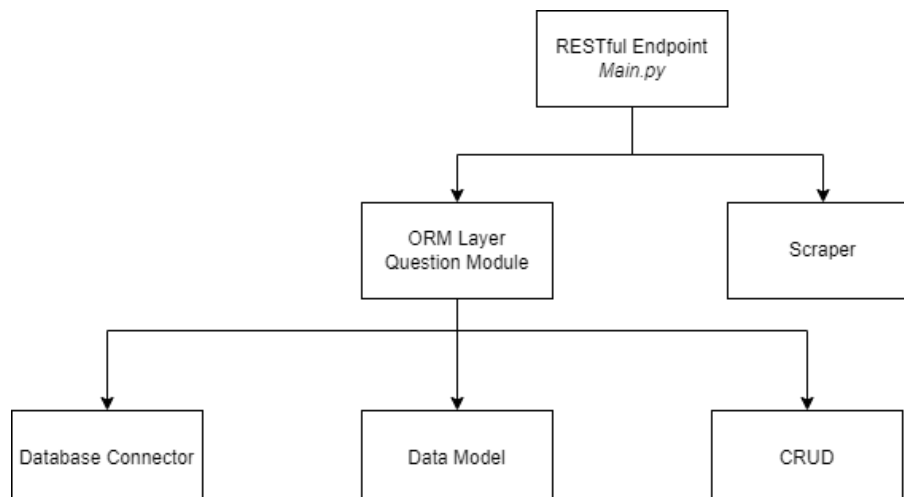
The following shows the software requirement specification for this component.

<b>Name</b>	Collaboration Service		
<b>Description</b>	Creates a collaboration room & maintains communication between users in the room.		
<b>Capabilities</b>			
Hosting the said room for the users. Maintaining communications between the users in the same room.			
<b>Service API</b>			
<i>Send Events</i>		<i>Receive Events</i>	
<ul style="list-style-type: none"><li>• BROADCAST_CHAT(roomId: UUID)</li><li>• BROADCAST_CODE(roomId: UUID)</li></ul>		<ul style="list-style-type: none"><li>• CREATE_SESSION(roomId: UUID)</li><li>• LOAD_SESSION(roomId: UUID)</li><li>• CODE(roomId: UUID, content: string, timestamp: datetime)</li><li>• CLOSE_SESSION(roomId: UUID)</li></ul>	
<b>Non-functional requirements</b>	<ul style="list-style-type: none"><li>• Communications should be secure between users in the same room</li><li>• Communications between users should be 99.9% reliable</li><li>• Users should be authenticated before joining their assigned room</li><li>• Service should be able to support with 1000 concurrent communications/rooms</li><li>• Users should be able to join back their current unclosed session in the event of disconnection</li></ul>		

*Table 3.4.4.8: Room service SRS table*

### 3.4.5. Question Service

A question service has been designed to ensure users are able to work on problems that they want to collaborate on at their desired level of difficulty. For implementation details, Python is used to scrape question data while MySQL is used for storing the questions as well as the relations of related difficulty as well as other tags. SQLAlchemy with FastAPI is then used to host the Python service for endpoint calls/data retrieval.



*Figure 3.4.5.1: Question service architecture*

Compared to other services using Node.js, Python is utilised for its ubiquitous functionality and ease of use for Web Scrapping. Python also has a very simple API for its middleware due to FastAPI allowing quick and easy development.

RESTful API endpoints are used to retrieve records. Tapping on the benefits of Microservices architecture, we were able to use the most appropriate tools for the question service such as BeautifulSoup4 and FastAPI for web scraping as well as deployment of the RESTful API endpoint.

The scraping process generally follows this sequence diagram:

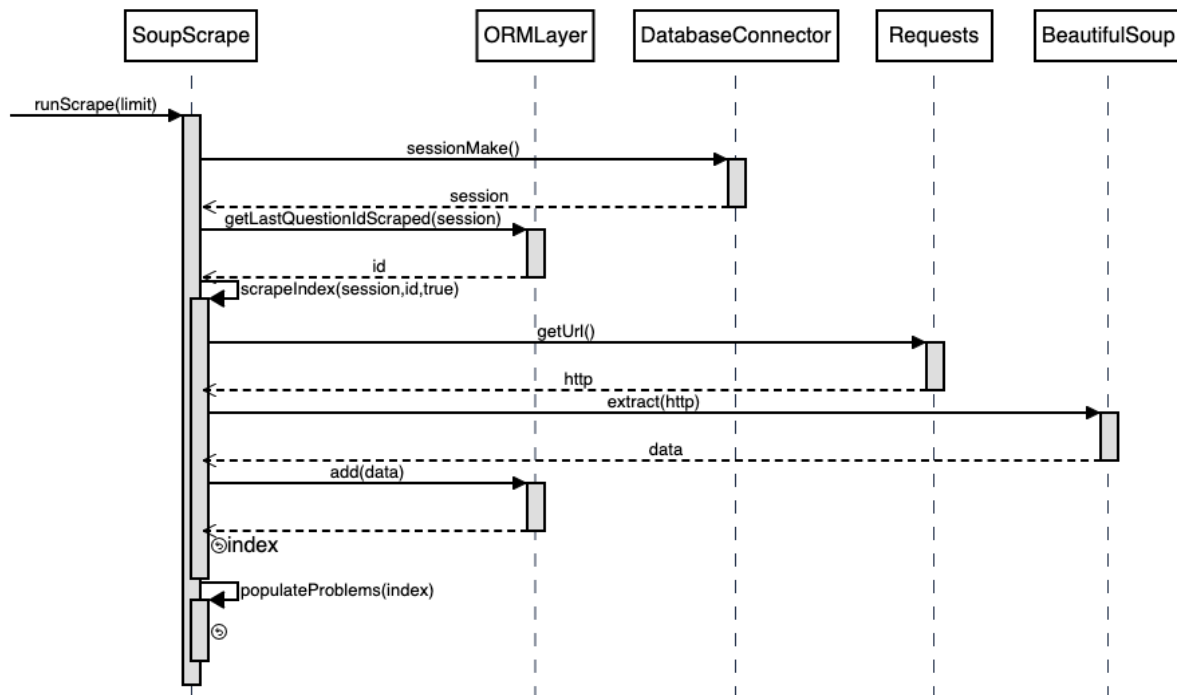
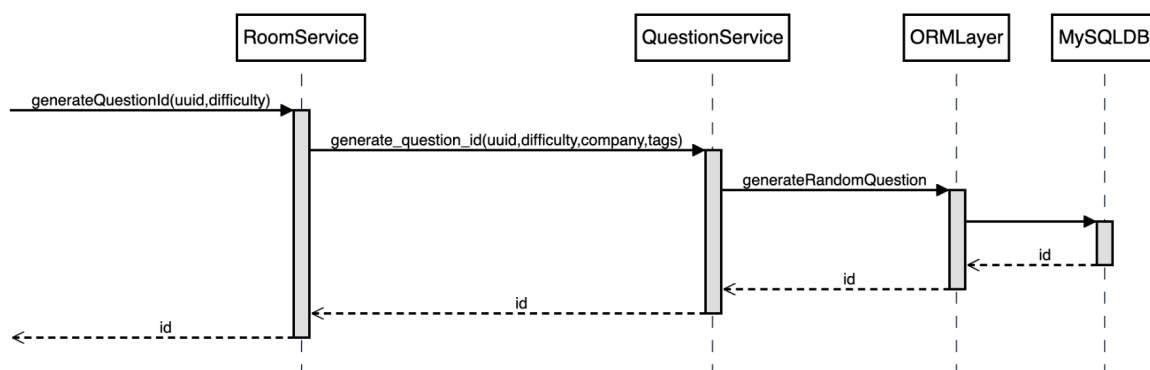


Figure 3.4.5.2: Question scraping sequence diagram

The database used is also MySQL rather than MongoDB due to the use of question id as a Foreign key that allows for Joins. Data in the MySQL Database is also unlikely to change over time and hence is optimised for fast reads and queries utilising the index for fast queries and joins, compared to MongoDB which benefits most from fast moving and real-time data writes.<sup>8</sup>



<sup>8</sup> <https://www.simplilearn.com/tutorials/mongodb-tutorial/mongodb-vs-mysql>

Figure 3.4.5.3: Question generation sequence diagram

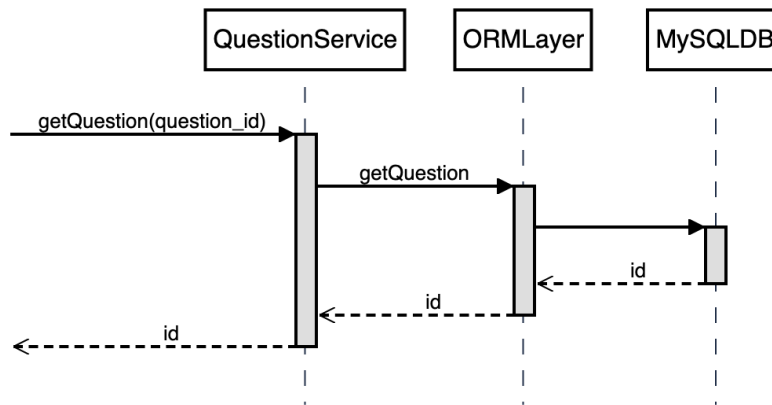


Figure 3.4.5.4: Question load sequence diagram

The question service utilises the following schema in MYSQL, paired with the sqlalchemy library available for python. This database is segmented into 3 main portions in order to segment only the parts that are required for each type of query on the database. In particular the question table which contains the actual long text of explanations and problem description is only queried when it is required, with a 1-1 mapping with questionindex.id. Therefore we can leverage on indexing to speed up query time.

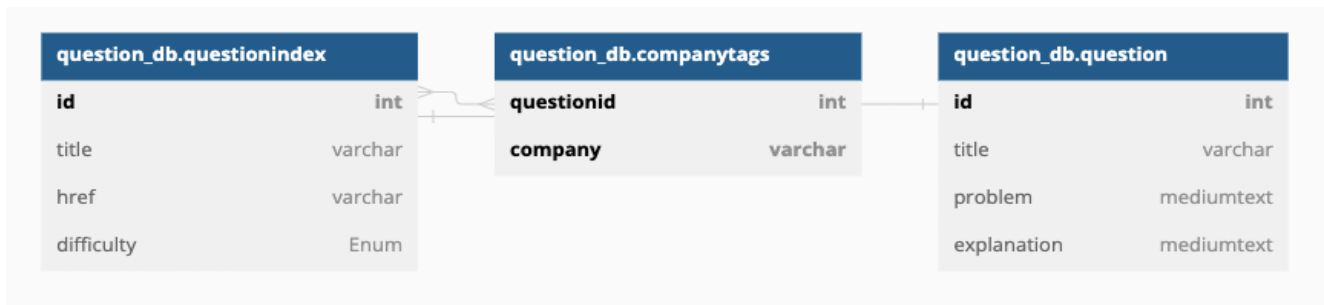


Figure 3.4.5.5: Question model schema on MySQL

The following shows the endpoints available for question service.

HTTP		Purpose	DTOs	
Method	Path		Input	Output
GET	api/question/:id ?solution=<bool>	Get a specified question ID with or without solution	-	[json] question_id: UUID, title: str, question: str, solution: str?
POST	api/question/ :question_id	Generate a question ID that the user has not seen before	[json]	[json] id: UUID

Table 3.4.5.6: Question service endpoints

The following shows the software requirement specification for this component.

<b>Name</b>	Question Service	
<b>Description</b>	Provide questions from a question bank based on difficulty and other tags	
<b>Capabilities</b>	Retrieving a specific question problem and solution from a known id Generating a random question id based on user input	
<b>Service API</b>		
<i>Send Events</i>	<i>Receive Events</i>	
NA	<ul style="list-style-type: none"> <li>• SCRAPE_QUESTIONS(void)</li> <li>• GENERATE_QUESTION(user: UUID, difficulty: string, tags?: list, company?: string)</li> <li>• GET_QUESTION(question: int)</li> </ul>	
<b>Non-functional requirements</b>	<ul style="list-style-type: none"> <li>• Interoperability is built into the service due to the microservices interactions only being based on the API exposed</li> <li>• Performance of the internal service is fast, as MySQL indexes reduce join time</li> <li>• Robust to invalid inputs and requests</li> </ul>	

Table 3.4.5.7: Question service SRS table

### 3.4.6. PubSub Service

Given the design decision to have a microservice architecture, there needs to be some form of inter-service communication. With stateless communications being done with RESTful endpoints, the use of a publisher-subscriber setup helps reduce the coupling present for communication that mutates state. As such, a pub-sub mechanism with socket.io is utilised to allow this communication between services - primarily between matching and collaboration services. Below shows the basic architecture diagram for the service and its usages respectively.

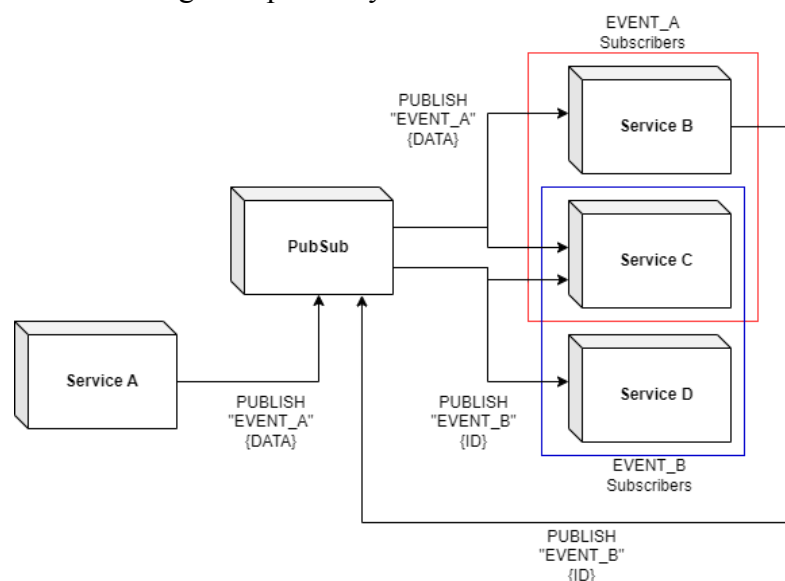


Figure 3.4.6.1: PubSub architecture diagram



The following illustrates a simple example on how PubSub operates. A given Service A subscribes to an EVENT once connected with the PubSub service. Once a service publishes data on that EVENT (could be from A or other services), A will receive the data and handle it accordingly.

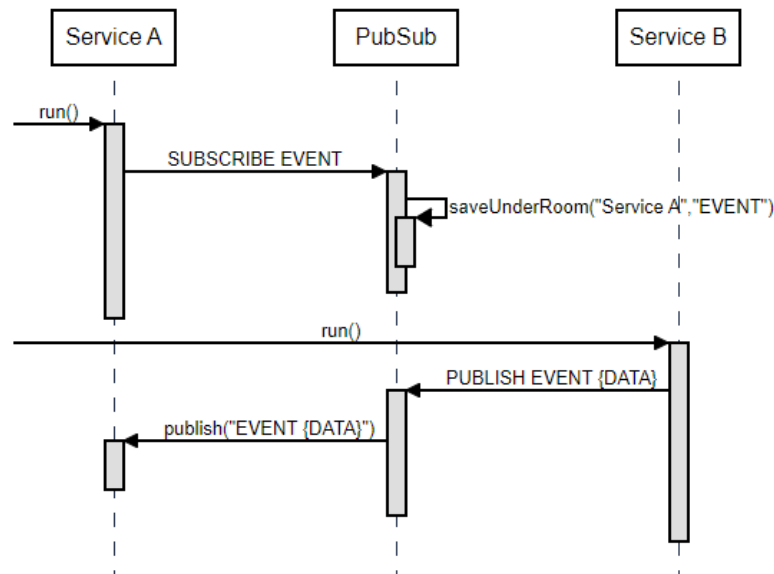


Figure 3.4.6.2: PubSub sequence diagram

Below shows the design decisions made for this service. With such considerations made, the most suitable tool for the required communication needs is socket.io - with their rooms capability to allow services to register under subscription event topics. The team has decided to set up this service as a temporal message forwarder due to reduced latency between operations (from saving and loading). At the same time, the use of socket io rooms allows broadcasting events to multiple listeners - allowing a transient, asynchronous and multi-receiver messaging system.

Persistence Type	Persistent	Transient
<b>Description</b>	<ul style="list-style-type: none"> <li>Message is stored and history can be reloaded upon new listener</li> </ul>	<ul style="list-style-type: none"> <li>Message is buffered for small periods of time</li> </ul>
<b>Decision: Transient</b>	As the system is required to operate and respond timely, there is no need for persistent storage and recovery of messages. Hence, a transient messaging system would be sufficient for this use case. Additionally, having records stored would mean additional handlers for past data for other services, which might conflict with other ongoing operations within the application.	
Synchronicity	Asynchronous	Synchronous
<b>Description</b>	<ul style="list-style-type: none"> <li>Allows independent functioning of the sender and receiver</li> </ul>	<ul style="list-style-type: none"> <li>Sender waits for a response/ acknowledgement from the receiver after messaging (blocking)</li> </ul>

<b>Decision: Asynchronous</b>	Due to the needs of the system to respond quickly, running the system asynchronously is required. This brings about the advantages of having a PubSub system, providing external systems an avenue to forward data events over while listening to other events.
-------------------------------	---

*Table 3.4.6.3: Determination of messaging nature of PubSub needed*

The following shows the software requirement specification for this component.

<b>Name</b>	PubSub Service		
<b>Description</b>	Acts as an inter-service communication tool		
<b>Capabilities</b>			
Allowing external services to subscribe to a desired topic. Allowing external services to publish with a chosen topic, and to have published messages forwarded to others services subscribed to it.			
<b>Service API</b>			
<i>Send Events</i>		<i>Receive Events</i>	
<ul style="list-style-type: none"><li>PUBLISH(event: String, data: JSON)</li></ul>		<ul style="list-style-type: none"><li>CONNECT(service: socket)</li><li>SUBSCRIBE(event: String: service: socket)</li></ul>	
<b>Non-functional requirements</b>	<ul style="list-style-type: none"><li>Communications between services should be 99.9% reliable</li></ul>		

*Table 3.4.6.4: PubSub service SRS table*

### 3.5. Development & Deployment

Apart from the services presented in the previous sections, deployment is also a crucial step in the development process. For this application, docker-compose is used as the orchestration medium. The rationale for such is that with the number of microservices present, it would not be feasible to perform integration and system tests when running each service individually. Hence, by utilising both unit testing to test the correctness of individual services and docker-compose to perform system wide level tests and deployment, this will allow smoother development across the team.

System	Testing Scheme
matching	<ul style="list-style-type: none"> <li>POST endpoint testing</li> <li>Successful match pairing within 30s</li> <li>Different difficulty resulting in no pairing</li> <li>PubSub ROOM_CREATED handling</li> </ul>
room	<ul style="list-style-type: none"> <li>Authorisation testing</li> <li>GET endpoint testing</li> <li>PubSub ROOM_CREATE handling</li> </ul>

collab	<ul style="list-style-type: none"> <li>• GET endpoint testing</li> <li>• PUT endpoint testing</li> <li>• PubSub ROOM_CREATED handling</li> </ul>
pubsub	<ul style="list-style-type: none"> <li>• Event subscription</li> <li>• Event publication</li> </ul>
question	<ul style="list-style-type: none"> <li>• Database connection testing</li> <li>• GET endpoint testing</li> <li>• POST endpoint testing</li> </ul>

*Table 3.5.1: Overall testing schematics*

Bash scripts have been made to smoothen the process further, reducing the need to depend on docker compose functions, but rather on the bash scripts and their arguments. There are scripts to (1) run test automation for each service through docker/docker compose/local build and (2) deploy all services through docker compose and provide additional options to view status and purge previous images.

```
e0407219@DESKTOP-T0TDT75:/mnt/c/Users/khair/Documents/GitHub/repositories/cs3219-project-ay2223s1-g4$ ./run-docker.sh
+ stop
+ docker-compose down
+ start
+ docker-compose up -d --force-recreate --renew-anon-volumes
[+] Running 12/12
  :: Network cs3219-project-ay2223s1-g4_service-net      Created
  :: Container collab-db                                Started
  :: Container question-db                              Started
  :: Container cs3219-project-ay2223s1-g4-user-service-1 Started
  :: Container room-db                                  Started
  :: Container matching-db                              Started
  :: Container cs3219-project-ay2223s1-g4-pubsub-service-1 Started
  :: Container cs3219-project-ay2223s1-g4-matching-service-1 Started
  :: Container cs3219-project-ay2223s1-g4-room-service-1 Started
  :: Container cs3219-project-ay2223s1-g4-collab-service-1 Started
  :: Container cs3219-project-ay2223s1-g4-question-service-1 Started
  :: Container cs3219-project-ay2223s1-g4-frontend-1   Started
+ exit
e0407219@DESKTOP-T0TDT75:/mnt/c/Users/khair/Documents/GitHub/repositories/cs3219-project-ay2223s1-g4$ docker-compose ps
NAME                                COMMAND                                SERVICE    STATUS    PORTS
collab-db                          "docker-entrypoint.s..."            collab-db  running   27017/tcp, 27021/tcp
cs3219-project-ay2223s1-g4-collab-service-1 "docker-entrypoint.s..."            collab-service running   0.0.0.0:8005->8005/tcp
cs3219-project-ay2223s1-g4-frontend-1 "docker-entrypoint.s..."            frontend  running   0.0.0.0:3000->3000/tcp
cs3219-project-ay2223s1-g4-matching-service-1 "docker-entrypoint.s..."            matching-service running   0.0.0.0:8001->8001/tcp
cs3219-project-ay2223s1-g4-pubsub-service-1 "docker-entrypoint.s..."            pubsub-service running   0.0.0.0:8004->8004/tcp
cs3219-project-ay2223s1-g4-question-service-1 "poetry run python3 ..."            question-service running   0.0.0.0:8003->8003/tcp
cs3219-project-ay2223s1-g4-room-service-1 "docker-entrypoint.s..."            room-service running   0.0.0.0:8002->8002/tcp
cs3219-project-ay2223s1-g4-user-service-1 "docker-entrypoint.s..."            user-service running   0.0.0.0:8393->8393/tcp
matching-db                        "docker-entrypoint.s..."            matching-db running   27017/tcp, 27022/tcp
question-db                        "docker-entrypoint.s..."            question-db running   0.0.0.0:3306->3306/tcp, 33060/tcp
room-db                            "docker-entrypoint.s..."            room-db    running   27017/tcp, 27023/tcp
```

*Figure 3.5.2: Deploying locally with Docker Compose*

In addition, CI using GitHub Actions also utilises Docker to run tests to ensure all dependencies of a given service is present - easing overall development and feature branch review.

The screenshot shows the GitHub Actions interface for a workflow named 'test'. The workflow is triggered by a pull request and is currently in progress. The left sidebar shows the workflow's summary, jobs, and details. The main panel displays the workflow's steps and their execution logs.

**Workflow Summary:**

- Workflow: test
- Status: succeeded 7 hours ago in 6m 56s
- Jobs: 1 (test)

**Workflow Steps:**

- Set up job (3s)
- Pull repository (1s)
- Use Node.js (1s)
- Run tests (6m 49s)

**Run tests step logs:**

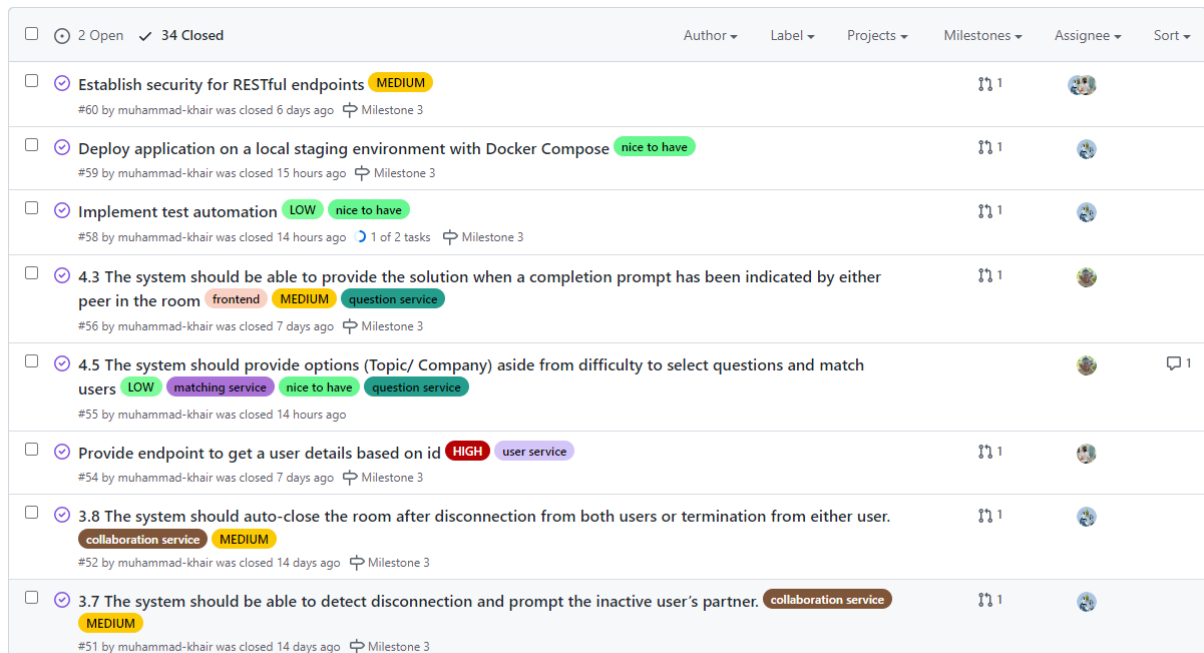
```
1 ▶ Run nick-fields/retry@v2
25 + test_backend
26 + for service in "${BACKEND_SERVICES[@]}"
27 + cd backend/pubsub-service
28 + ./run.sh t
29 + verify
30 + set -e
31 ++ docker build -q -f Dockerfile.dev .
32 + docker run --rm sha256:e9b7a8538a9bc30030839abdbe9954e5f24af0d5cf7e3b22d728d74f94c66f38
33
34 > pubsub-service@1.0.0 test
35 > mocha --timeout 5000 --exit
```

Figure 3.5.3: GitHub Actions running test script

## 4. Project Timeline and Team Contributions

### 4.1. Project Management

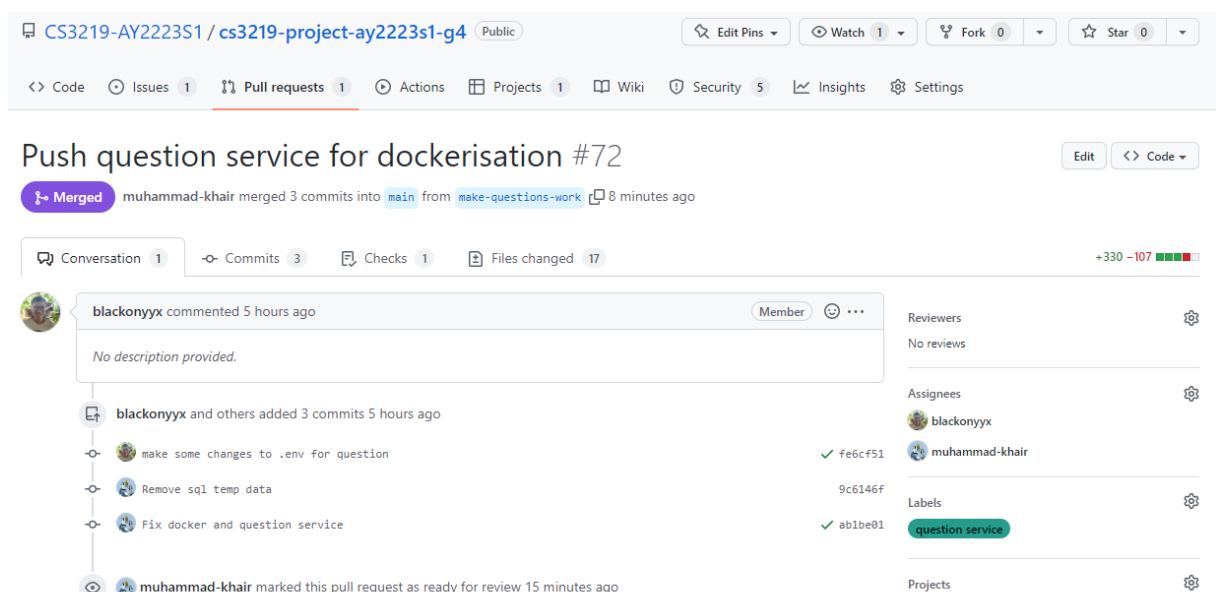
The project is split into 3 major iterations, as recommended by the module's guidelines. In order to plan for the work done within each iteration, a backlog is created and initial FRs and NFRs are recorded. From there the work is assigned to each team member, and internal APIs and DTOs were discussed to allow parallel work.



Issue ID	Title	Priority	Assignee	Status
#60	Establish security for RESTful endpoints	MEDIUM	muhammad-khair	Closed 6 days ago
#59	Deploy application on a local staging environment with Docker Compose	nice to have	muhammad-khair	Closed 15 hours ago
#58	Implement test automation	LOW, nice to have	muhammad-khair	Closed 14 hours ago
#56	4.3 The system should be able to provide the solution when a completion prompt has been indicated by either peer in the room	frontend, MEDIUM, question service	muhammad-khair	Closed 7 days ago
#55	4.5 The system should provide options (Topic/ Company) aside from difficulty to select questions and match users	LOW, matching service, nice to have, question service	muhammad-khair	Closed 14 hours ago
#54	Provide endpoint to get a user details based on id	HIGH, user service	muhammad-khair	Closed 7 days ago
#52	3.8 The system should auto-close the room after disconnection from both users or termination from either user.	collaboration service, MEDIUM	muhammad-khair	Closed 14 days ago
#51	3.7 The system should be able to detect disconnection and prompt the inactive user's partner.	collaboration service, MEDIUM	muhammad-khair	Closed 14 days ago

Figure 4.1.1: GitHub Issues page

The project is maintained on GitHub through GitHub Projects and Issues. From there the issues are tagged with their respective domains (user, matching, etc.) as well as the assignee for the task. Additionally, work is done on separate branches and is joined through Pull Requests.



CS3219-AY2223S1 / cs3219-project-ay2223s1-g4 Public

Push question service for dockerisation #72

Merged muhammad-khair merged 3 commits into main from make-questions-work 8 minutes ago

Conversation 1 Commits 3 Checks 1 Files changed 17 +330 -107

blackonyyx commented 5 hours ago

No description provided.

blackonyyx and others added 3 commits 5 hours ago

- make some changes to .env for question
- Remove sql temp data
- Fix docker and question service

muhammad-khair marked this pull request as ready for review 15 minutes ago

Reviewers: No reviews

Assignees: blackonyyx, muhammad-khair

Labels: question service

Figure 4.1.2: GitHub Issues page

The project is also managed with the milestone feature so the team is able to check the overall progress within a given timeframe.

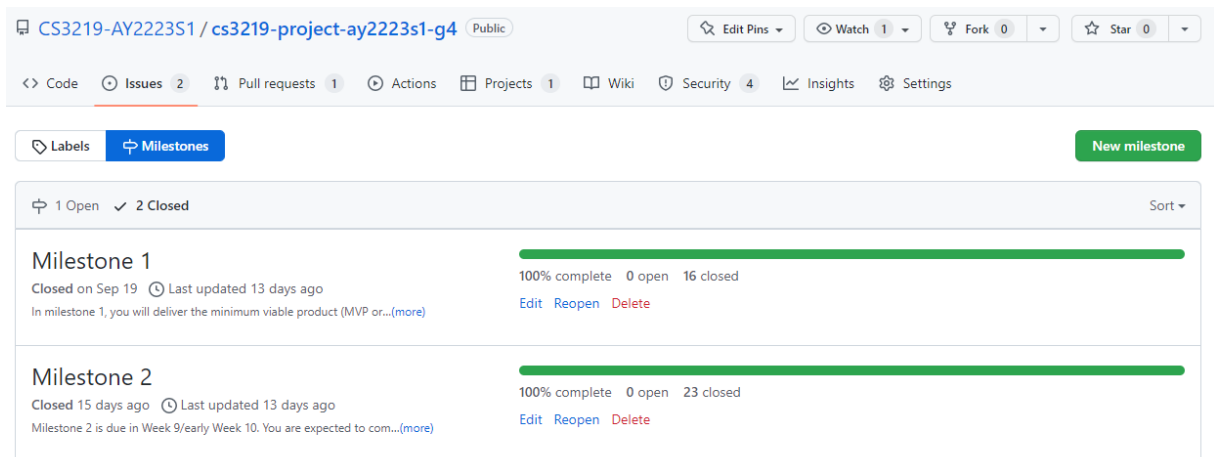


Figure 4.1.3: GitHub Milestones page

Additionally, continuous integration is employed with GitHub Actions and Docker containers to run tests on the services. These are run by branches before being integrated into the main branch.

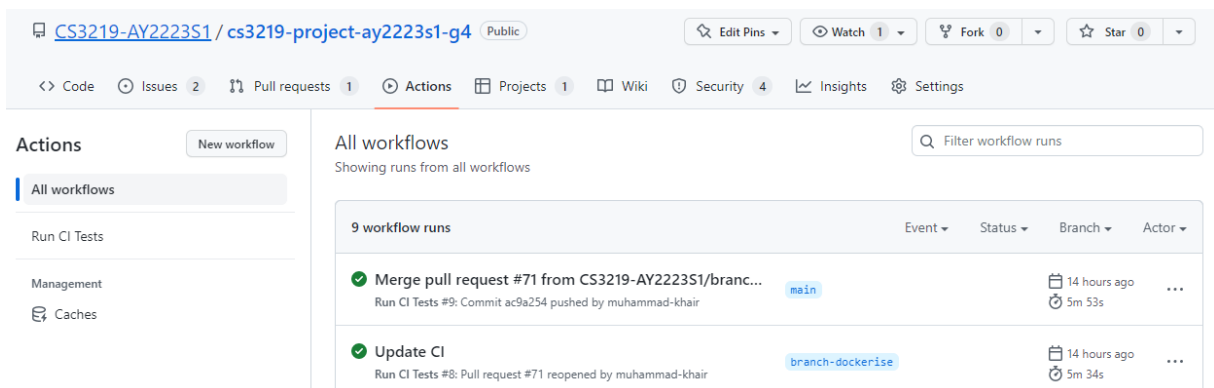


Figure 4.1.4: GitHub Workflow page

When it comes to team dynamics and team meetups, the team holds weekly meetings to discuss overall progress and goals to be cleared by the following week. Additionally, individual pairs would have separate catch-ups to discuss service integration (either through PubSub events or API endpoints as well as their DTOs). The team works on a democratic structure with shared roles - allowing each of the members to have equal responsibility for the tasks at hand.

## 4.2. Project Timeline

The contributions made on an individual basis are collated in the tables below for each iteration. These tags can be referenced to the FRs and NFRs in Section 2.2 and 2.3 respectively.

### 4.2.1. Iteration 1

For iteration 1, the main deliverables are shown below, which mainly focus on completing the Minimum Viable Product (MVP) of Software Interview Xchange by setting up the codebase, and project management tools and implementing the “Must-Have” features.

- User Service
- Matching Service
- Question Service
- Collaboration Service
- Basic UI for frontend interactions

	Week 3	Week 4	Week 5	Week 6	Recess
Ee Liang	Read up on requirements	Setup codebase Explore auth0 NFR-SEC-1 NFR-SEC-2 NFR-SCA-2	FR-U-1 FR-U-2 FR-U-3 FR-U-4	Docs (User) NFR-SEC-4	
Michaelia	Read up on requirements	NFR-SCA-2 Setup codebase Read up on sockets	FR-M-1	FR-M-4 Docs (FRs)	Docs (Review)
Khair	Read up on requirements	NFR-SCA-2 Setup codebase Read up on sockets and PubSub	FR-M-1 FR-M-2 Docs	FR-M-3 FR-C-1 FR-C-3	Docs (Overall diagrams)
Stephen	Read up on requirements	Setup codebase	FR-U-3	FR-U-4 Docs (Background)	

*Table 4.2.1.1: Iteration 1 timeline*

### 4.2.2. Iteration 2

For the second iteration, the main deliverables are shown below. These include having “Must-Have” objectives cleared and implemented with substantial testing for them.

- Focus on implementing PubSub mechanisms to allow microservice communication
- Clean frontend and integrate auth0 to services
- Build on room/collaboration services (Must Have)
- Build upon question service (Must Have)
- Focus on local deployment of total services (Must Have)

	<b>Recess</b>	<b>Week 7</b>	<b>Week 8</b>	<b>Week 9</b>	<b>Week 10</b>
Ee Liang	Docs	FR-U-5	FR-U-6	FR-U-7	Docs (User)
Michaelia	NFR-PER-2	NFR-PER-4	NFR-SCA-1 NFR-INT-1	NFR-INS-1 Write test cases (Matching) Docs	Docs (Matching & Diagrams) Write test cases (Matching)
Khair	FR-C-2 NFR-PER-1	FR-C-5	FR-C-6 Docs (Architecture)	NFR-INS-1 Docs (PubSub)	Docs (Room & Collab)
Stephen	FR-Q-1 Docs (Report)	FR-Q-2 NFR-SEC-3	FR-Q-3	FR-Q-1 Docs (Question)	Docs (Question)

*Table 4.2.2.1: Iteration 2 timeline*

### 4.2.3. Iteration 3

For the third (and final as to when this document is submitted) iteration, the whole product is to be fully shipped out, together with at least 3 “Nice to have” features. The main objectives for this iteration are as such:

- Beautify frontend UI (Nice To Have)
- Implement history feature (Nice To Have)
- Focus on deployment with docker-compose and CI testing (Nice To Have)
- Clear outstanding issues from previous iterations

	<b>Week 10</b>	<b>Week 11</b>	<b>Week 12</b>	<b>Week 13</b>
Ee Liang	NFR-USE-1	NFR-USE-1 NFR-SEC-3	Frontend NFR-USE-1 Docs (Revisions) FR-C-8	Frontend NFR-USE-1 NFR-SAF-1 Docs (Revisions) Demo
Michaelia	Write test cases (Matching)	NFR-INT-1	NFR-SEC-3 Testing Docs (NFRs, Revisions)	Docs (Revisions) Demo
Khair	FR-C-7 FR-C-4 Write test cases (PubSub, Room & Collab)	FR-C-8 NFR-INS-2	FR-C-8 NFR-INS-2 NFR-SEC-3 Docs (Revisions)	Docs (Revisions) Demo
Stephen	FR-Q-1 FR-Q-4	NFR-INS-2 NFR-PER-3	NFR-INS-2	Frontend Docs (Question) Demo

*Table 4.2.3.1: Iteration 3 timeline*



## 5. Possible Improvements

There are several means as to how we can further improve Software Interview Exchange (SIX) to bring about increased user satisfaction and a closer alignment to the goals of the application. These means will be described with a short description as to how to begin implementing such improvements.

### 5.1. Scalability & Deployment

Deployment would be a critical requirement in order to get SIX off the ground as in order to have a truly worthwhile collaborative experience. While the current proof of concept demonstrates this at a local network level, it is insufficient to capture the full scope of what we aim for.

Hand in hand comes the need for scalability as the application grows. As the user base and requirements grow, there is a need to consider how such services can scale to meet demand. One potential option would be to use Kubernetes to help with the deployment and management of infrastructure - through adaptive scaling of pods. This would include providing the right configurations for each service and its databases. In addition to that, an Nginx instance could help as a reverse proxy and as a means to deal with endpoint redirection for respective services.

### 5.2. User Experience

One key area that the team would like to improve in future works would be to allow users to customise their experience more on SIX. These customisations include favourite languages, profile description, background information and profile pictures - which can be easily extended with a local database paired with Auth0. However, providing such capabilities for users to input must go through verifications (through ensuring trust and safety of our user base) to preserve the wellbeing of all users on the platform. Another key aspect of the application revolves around the collaboration and communication between peers on the platform. Thus, future extensions could include video and voice conferencing with WebRTC protocols.

### 5.3. Matching Selection

Another vector to work on would be improving the training selection process. This could include things like allowing users to select which type of questions or company-based question sets they want to be asked, or providing more options for how they want to be trained, such as timer and hints. This can be done by extending both the question service's database to provide such data upon scraping and the matching service to perform more intricate matching of users based on their selection. The main challenge the team foresees would be deciding on an algorithm that can ensure a reasonable trade-off between pairing guarantee under 30 seconds as well as meeting as much of the users' selection requirements.

## 5.4. Question Variance

To give users the best possible experience to improve their coding skills, it is essential to provide users with more variance on the nature of questions provided for training. For example, questions should be tagged with companies that asked such questions before, and with the nature of questions (such as dynamic programming and graphs for example). Theoretical and design questions can also be considered to provide a holistic experience for the users to better prepare for technical interviews. At the same time, the team plans on extending our quality control measures to ensure that only the most relevant and useful questions are being asked. These can be done by extending the question service fields and matching service to allow such intricate selection, as seen in the previous section - while also extending the user service's database to hold more information on a user's career plans.

## 5.5. Additional Training Vectors

The team recognises that not everyone wants to train with others all the time, so in the future, a solo training mode would be offered. This will allow users to work through questions at their own pace, without having to worry about being interrupted by others. This can be paired with timers and hints, similar to what has been mentioned in the previous sections. The current codebase can be easily extended to provide the solo interface but requires some tweaks with sockets to ensure the confidentiality and reliability of coding in the application. Also, code language selection and type hinting would improve the user's experience by allowing the user to focus on semantics rather than the syntax of their chosen programming language. This can be done by integrating external services (such as Judge0<sup>9</sup>) that provide API syntax-ing and code execution with the current code box.

---

<sup>9</sup> Judge0. Accessed on 8 November 2022. <https://judge0.com/>

## 6. Reflections

Throughout this project, we have encountered many difficulties, especially with time management and the distribution of workload. Initially, a lot of us had not explicitly learnt the frameworks and tools that we were using, and so we had to learn them on the fly whilst also trying to deliver a working product. This led to a lot of frustration and wasted time. We eventually managed to get past these issues by communicating more openly with each other face-to-face about the blockers we encountered and by taking the time to learn the necessary skills. We also made sure to distribute the workload more evenly so that everyone had a stake in the project.

Through this hands-on group project, we have learnt to appreciate modularisation and abstraction more. We realised that by modularising services based on needs and boundaries had helped us keep the codebase clean and maintainable. This has made it easy to identify dependencies between different services. This approach also makes the system flexible to scale individual services as needed, which is one of the essential requirements we had identified. Other than that, we also find it easy to parallelise development as different sub-teams can work on different microservices at the same time without impacting one another's work, due to the endpoints being agreed upon beforehand. This has led to noticeably faster development times, and fewer risks when implementing or updating our planned requirements during our sprints.

Other than modularisation, we also learnt to appreciate dockerisation more. This is because dockerisation allows for the easy packaging and deployment of microservices, which can then be run in parallel on different servers or even in different environments. This makes it an ideal tool for deploying our microservice-based application Software Interview Xchange since we want our application to be easily installable on different OS systems. The process of dockerisation is also relatively simple as we just need to build our microservice image, push it to a Docker registry and then deploy that image to any server that has Docker installed.

Lastly, we as a team have learnt more on the terminology used to describe and justify systems. Conventionally, or in previous modules, implementing an application would just be coding and adding tests without an end goal in mind. However, utilising domain driven design to define the user, requirements, and bounded context allows a more clearer objective to be set. This then allows us to better decide on the overall structure and relevant API discovery to enhance parallel development. Overall, this project allows the team to learn and appreciate software development through this web project initially dubbed as 'PeerPrep'.