



CS3219 Software Engineering Principles and Patterns

Peer Prep Project Report

Name	Matriculation Number
Toh Zhan Qing	A0218230U
Liao Keng I	A0217532L
Teo Yick Fong Alex	A0221444R
Sim Jun Heng	A0218348Y

Table of Contents

1. Introduction	2
2. Functional Requirements	2
2.1 User Stories	2
2.2 Quantification of FR	3
2.3 Services to fulfil FR	6
3. Non-Functional Requirements	6
3.1 Requirement Prioritization	6
3.2 Availability NFR	7
3.3 Performance NFR	7
3.3.1 Loading times for login and main page	8
3.3.2 End-to-End delay for Real-time code editor	8
3.4 Scalability NFR	9
3.4.1 Scalability of deployed application	9
3.5 Reliability NFR	11
3.5.1 Load testing to test question service	11
3.5.2 Load testing to test user service	12
3.5.3 Feature and regression testing	12
4. Architecture	14
4.1 Architecture Diagram	14
4.2 Architecture Decision - AWS Elastic Beanstalk	14
4.3 Architecture Decision - Docker vs Node.js Runtime	14
4.4 Architecture Decision - MongoDB Atlas	15
4.5 Architecture Description	15
5. Design Pattern	18
6. Tech Stack	18
7. Devops	19
7.1 Testing	19
7.2 Local staging area deployment	19
7.3 Continuous Deployment	20
7.3.1 Frontend deployment	20
7.3.2 Backend services deployment	20
7.3.3 Deployment policy	20
8. Software Development Lifecycle	22
9. Microservices	23
9.1 User Service	23
9.2 Question Service	28
9.3 Matching Service and Collaboration Service	30
9.4 Chat Service	31
10. Implemented Features	32
11. Member Contributions	32

1. Introduction

Peer prep is intended for software engineers preparing for technical coding interviews. Programmers may become more accustomed to the environment by simulating the actual situation. PeerPrep provides a web application in which our target audience is able to replicate the environment for them to familiarise themselves and improve on their skills. Included in this application is a matching function which allows users to get matched to each other, bring them to a real-time code editor where they can work together to solve problems or revise together. Additionally, a chat service is also introduced to encourage improved collaboration and communication.

This application is deployed on the cloud and is currently live at:

<http://cs3219g49-frontend.eba-mxp3gzbj.ap-southeast-1.elasticbeanstalk.com/>

2. Functional Requirements

2.1 User Stories

ID	User Story (US)	FR Mapped to
1	As a user, I want to be able to login into my account with my unique username and matching password.	FR1.3
2	As a user, I want to be able to sign up for a new account so that I will be able to use the service.	FR1.1, FR1.2
3	As a user, I want to be able to logout of my account so that other people will not be able to access it on that same computer.	FR1.4
4	As a user, I want to be able to delete the account I created when I am logged in, so that I can remove my information from the service completely.	FR1.5
5	As a user, I want to be able to change my password when I am logged in, to my preferred new password	FR1.6
6	As a user, I want to be able to select a level of difficulty of a question so that I can attempt a question within my skill range.	FR2.1

7	As a user, I want to be matched with another user searching for the same difficulty so that solving the questions will be within the same skill range for both of us.	FR2.2
8	As a user, I want to be able to know when I would get be matched to another user and be redirected to the collaboration page immediately.	FR2.3
9	As a user, I want to be able to stop the matching process if I no longer wish to do a challenge	FR2.4
10	As a user, I want to communicate with my matched partner so we can collaborate on the question more efficiently.	FR3.1
11	As a user, I want the application to support various difficulties and only give questions corresponding to the difficulty, so that the task is manageable for me.	FR4.1, FR4.2, FR4.4
12	As an admin, I want to be able to add in new questions and delete existing questions so that the question database is always updated with new challenges for the users	FR4.3
13	As a user, I want to have a room dedicated to my matched partner and me so that we have a common interface to work on the question.	FR5.1
14	As a user, I want to be able to save my collaboration code so that I can continue coding at another time as long as the session is still active	FR5.3
15	As a user, I want to see syntax highlighting on my Javascript code so that I can focus my attention on the code logic instead when solving the questions.	FR5.2
16	As a user, I want to be redirected back to my existing matching session if I happen to exit the page accidentally so that I can continue with my training.	FR5.4
17	As a user who is collaborating on the questions with a partner, I want to be able to type and see our changes with minimal latency, so that we can work together in real time	N3.3

2.2 Quantification of FR

Functional Requirements (FR)	Priority
------------------------------	----------

User Service		
1.1	The system should allow users to create an account with username and password on a sign-up page.	High
1.2	The system should ensure that every account created has a unique username.	High
1.3	The system should allow users to log into their accounts by entering their username and password.	High
1.4	The system should allow users to log out of their account.	High
1.5	The system should allow users to delete their account when they are currently logged in.	High
1.6	The system should allow users to change their password when they are currently logged in.	Medium
Matching Service		
2.1	The system should allow users to select the difficulty level of the questions they wish to attempt	High
2.2	The system should be able to match two waiting users with similar difficulty levels and put them in the same room.	High
2.3	If there is a valid match, the system should match the users within 30 seconds.	High
2.4	The system should allow the users to stop the matching process if they choose to.	High
Chat Service		
3.1	The system should allow the matched users to communicate via chat if they are in the same room.	Medium
Question Service		
4.1	The system should supply the matched users with questions from the question bank	High
4.2	The system should present users with a random question corresponding to the difficulty specified by the match.	High
4.3	The system should allow admins to add and remove questions as a whole from the database.	High
4.4	The system should have questions with different difficulty levels.	High
4.5	The system should save the suggested answer for each question.	Medium

Collaboration Service		
5.1	The system should keep track of who is currently in a room/collaborating	High
5.2	The system should have syntax highlighting for submitted code.	Low
5.3	The system should allow the user to save their code in the current session.	Medium
5.4	The system should redirect the user back to his room if he has a current active session	High

Non-Functional Requirements (NFR)		Priority
Security		
N1.1	Users' passwords should be hashed and salted before being stored in the database.	High
N1.2	The system should be CSRF-resilient.	Low
Scalability		
N2.1	The system should be able to support 100 users concurrently while maintaining optimal performance.	Medium
N2.2	The system should be able to match up to 50 users at any time.	Medium
Performance		
N3.1	The system should always return a response within 3 seconds.	Low
N3.2	The website should load and render text and images within 5 seconds.	Low
N3.3	The free text field should be updated in near real-time.	High
Robustness		
N4.1	Mean time to restore system after failure should be lesser than 20 minutes	Medium
N4.2	Data loss after a system failure should be less than an hour's worth.	Medium
Reliability		
N5.1	The system should be able to perform without failure in 99% of use cases	High
Availability		

N6.1	The server should be accessible to users 99.9% of the time every month during business hours GMT +8	High
-------------	---	-------------

2.3 Services to fulfil FR

No.	Service Name	Description
2.3.1	User Service	The user service is responsible for the creation, management, and destruction operations of all users, like login, logout, signup, and account deletion.
2.3.2	Matching Service	The matching service is responsible for matching two users who have chosen the same difficulty level together.
2.3.3	Collaboration Service	The collaboration service is responsible for allowing the two matched users to collaborate and synchronise their states. This included the real-time code editor and questions displayed to both players.
2.3.4	Question Service	The question service is responsible for the storage and retrieval of all questions that are available in the database.
2.3.5	Chat Service	The chat service is responsible for sending messages between the two players.

3. Non-Functional Requirements

3.1 Requirement Prioritization

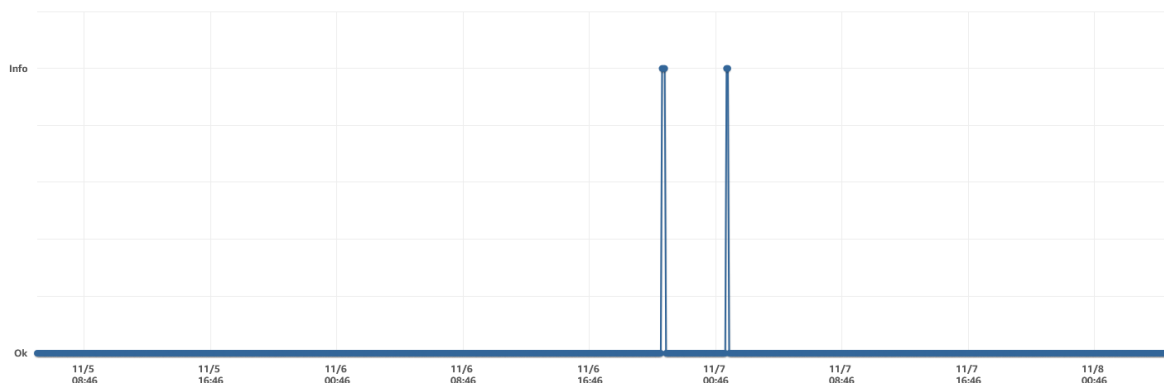
Attributes	Score	Availability	Performance	Reliability	Robustness	Scalability	Security
Availability	4		<	<	<	^	<
Performance	2			<	^	^	<
Reliability	3				<	<	<
Robustness	2					^	<
Scalability	4						<
Security	0						

We prioritised our NFRs according to the table above. We decided that availability, performance, and reliability are of utmost importance to this application.

3.2 Availability NFR

Availability		
N6.1	The server should be accessible to users 99.9% of the time every month during business hours GMT +8	High

Using AWS's cloud infrastructure allows our deployed applications to have high availability. As seen below, we are able to achieve our NFR of 99.9% availability. In fact, over the span of 3 days, our application was available 100% of the time. In the chart below, the health of the application is constantly monitored. As seen, the health of the site is constantly healthy, meaning that users are able to access it. The two spikes correspond to our two commits to main, which triggered a deployment. Due to our immutable deployment policy, the application was live throughout the duration.



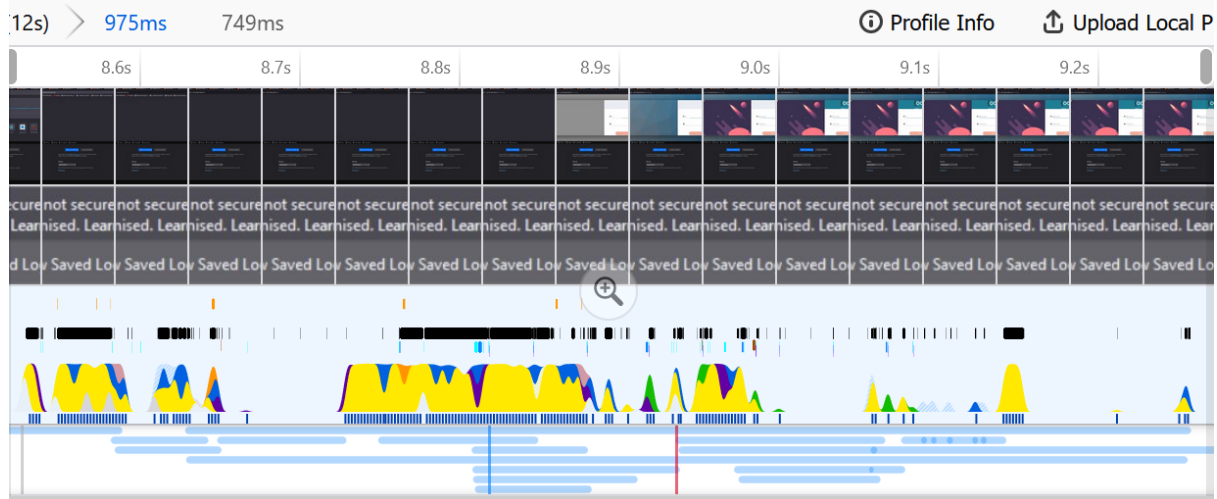
Monitoring the health status of our application

3.3 Performance NFR

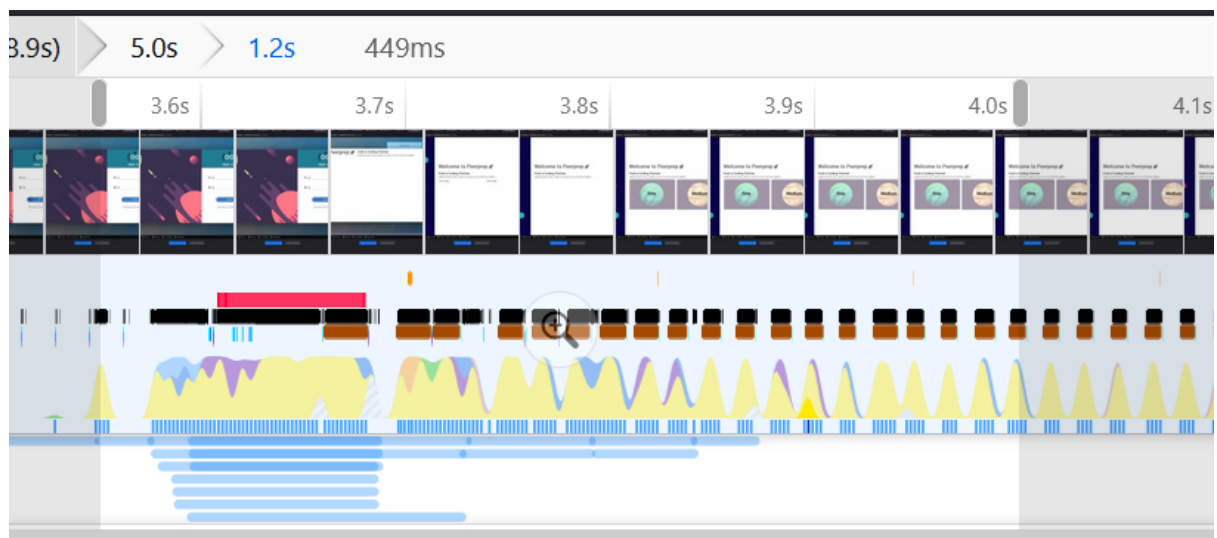
Performance		
N3.2	The website should load and render text and images within 5 seconds.	Low
N3.3	The free text field should be updated in near real-time.	High

3.3.1 Loading times for login and main page

We tested out our deployed application and used the browser's tools to measure the time taken to load our login and main page. As seen in the figures below, the login page and main page took around 750 ms and 450 ms, respectively.



Performance analysis of the login page using Firefox profiler. Result: 750ms



Performance analysis of the main page using Firefox profiler. Result: 450ms

3.3.2 End-to-End delay for Real-time code editor

To test our real time code editor, we used wireshark to capture packets while we typed into the code editor. We used 2 different browsers, navigated to the deployed application, and matched them together. Then we started to capture packets using Wireshark. A small subset of the captured packets is shown below.

181 4.482440	192.168.1.37	54.254.2.61	TCP	54 62884 → 80 [ACK] Seq=1 Ack=128 Win=1021 Len=0
182 4.668496	192.168.1.37	54.255.83.128	TCP	120 62890 → 80 [PSH, ACK] Seq=136 Ack=1 Win=1024 Len=66
183 4.670587	54.255.83.128	192.168.1.37	TCP	60 80 → 62890 [ACK] Seq=1 Ack=202 Win=110 Len=0
184 4.671906	54.254.2.61	192.168.1.37	TCP	116 80 → 62884 [PSH, ACK] Seq=128 Ack=1 Win=111 Len=62
185 4.713953	192.168.1.37	54.254.2.61	TCP	54 62884 → 80 [ACK] Seq=1 Ack=190 Win=1021 Len=0
186 4.873207	192.168.1.37	54.255.83.128	TCP	119 62890 → 80 [PSH, ACK] Seq=202 Ack=1 Win=1024 Len=65
187 4.875332	54.255.83.128	192.168.1.37	TCP	60 80 → 62890 [ACK] Seq=1 Ack=267 Win=110 Len=0
188 4.876485	54.254.2.61	192.168.1.37	TCP	115 80 → 62884 [PSH, ACK] Seq=190 Ack=1 Win=111 Len=61

Capturing packets sent to and from our application using wireshark

These 8 packets correspond to 2 characters typed into the editor. The first 2 packets are sent when a character is typed, and the next two are the incoming packets on the other editor. As seen from the timestamps, the end-to-end delay is 190 ms, which is sufficient to function as a real-time code editor.

3.4 Scalability NFR

Scalability		
N2.1	The system should be able to support 100 users concurrently while maintaining optimal performance.	Medium

3.4.1 Scalability of deployed application

To test the scalability of our deployed application, we send 100 requests to the server every second to simulate a load of 100 concurrent users every second. We went further to test for auto-scaling of the application under immense load, such that one instance is overwhelmed and another needs to be created to deal with the load.

To test for our application automatically scaling up, we temporarily changed the threshold of scaling triggers from network out at 6MB/s to 3000 requests/minute. This will allow us to easily load the application hard enough and see it scale correctly. We then used a tool called “Hey” from <https://github.com/rakyll/hey> to help us load the website. We set the program to send 100 requests per second for 5 minutes and waited. The application, as anticipated, started to create new instances when it went over our scaling triggers.

Sending 100 requests/s for 5 minutes to the deployed application

Scaling triggers activated and auto-scaling has started

New instance created and ready

10

Recent events			Show all
			< 1 >
Time	Type	Details	
2022-11-08 05:42:52 UTC+0800	INFO	Removed instance [i-0e93545269663198a] from your environment.	
2022-11-08 05:31:52 UTC+0800	INFO	Added instance [i-0b75f89cb0de87ddc] to your environment.	

New instance terminated

3.5 Reliability NFR

Reliability		
N5.1	The system should be able to perform without failure in 99% of use cases	High

3.5.1 Load testing to test question service

To find out how reliable our question service is in delivering questions to the frontend on a high traffic condition, we conducted load testing on the common api calls that frontend makes and evaluate the reliability as:

Number of success requests / Total number of requests

We ran the tests with Postman and returned the number of requests with the status code 200. To simulate a heavy load on the question service, we can set the iteration value to 1000. The results were promising, with a 100% success rate on the test.


peer prep - Run results

Run Again

Automate Run

+ New Run

Export Results

 Run on Today, 17:22:09 · [View all runs](#)

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	none	1000	3m 10s	2000	18 ms

RUN SUMMARY

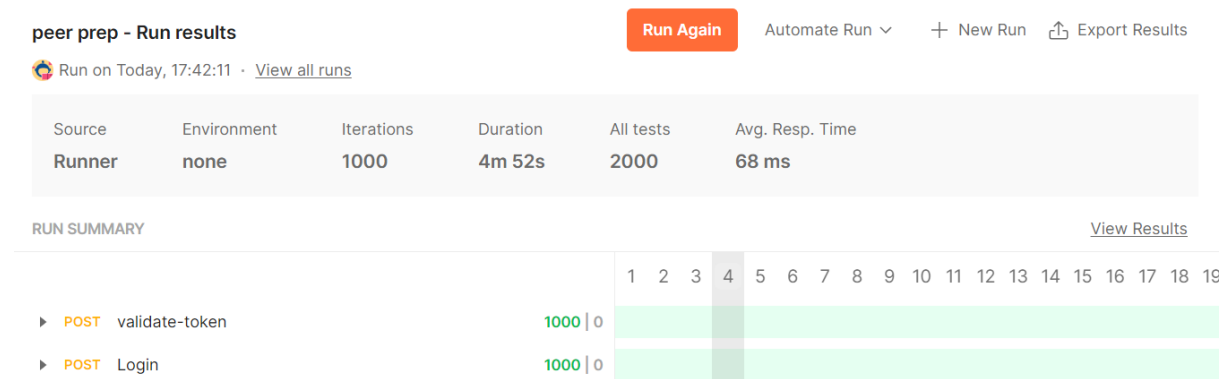
[View Results](#)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
▶ POST getQuestionContent																			
▶ POST getQuestionAnswer																			

100% successful request for question service

3.5.2 Load testing to test user service

In a similar manner, we tested the top 2 API calls—login and verify-token requests—for the question service using the same methodology. Every time a login attempt is made, these two requests are necessary. The results showed that 1,000 requests had a total success rate of 100%.



100% successful request for user service

3.5.3 Feature and regression testing

To minimise the errors that may be introduced when we develop new features, we developed unit test cases in our microservices during our development cycles. The test cases are placed in the respective microservice folders. Every test case created is run as part of our GitHub actions workflow to find any potential flaws in our feature. By doing this, we can make sure that any bugs are found during the development process and that our services are always operational.

```
user-service > test > JS index.js > describe('Test change password') callback > INVALID_USERNAME
206
207 describe("Test create user", () => {
208   before(async () => {
209     // Clear DB and insert test user
210     await userModel.deleteMany({})
211     const testPassword = await createSaltAndHash("existinguser")
212     const user = userModel({ username: "existinguser", password: testPassword })
213     await user.save()
214   })
215
216   after(async () => {
217     // Clear DB
218     await userModel.deleteMany({})
219   })
220
221   it("Testing create new user returns success", done => {
222     const VALID_USERNAME = "newuser"
223     const VALID_PASSWORD = "newuser"
224     let validUser = { username: VALID_USERNAME, password: VALID_PASSWORD }
225     chai.request(app)
226       .post('/api/user/signup')
227       .send(validUser)
228       .end((req, res) => {
229         expect(res).to.have.status(201);
230         expect(res.body.message).to.equal(`Created new user ${VALID_USERNAME} successfully!`)
231         done();
232       })
233   })
}
```

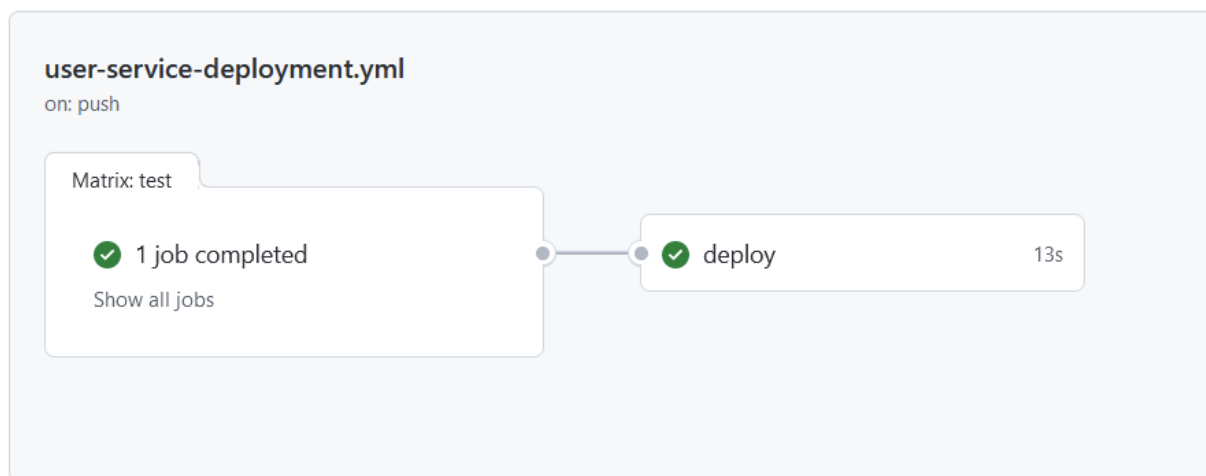
User service unit test cases

```

17 jobs:
18   test:
19
20     runs-on: ubuntu-latest
21
22     strategy:
23       matrix:
24         node-version: [16.x]
25         # See supported Node.js release schedule at https://nodejs.org/en/about/releases/
26
27     env:
28       ENV: TEST
29       JWT_TEST_KEY: ${ secrets.JWT_TEST_KEY }
30       DB_CLOUD_URI_TEST: ${ secrets.DB_CLOUD_URI_TEST }
31
32     steps:
33     - uses: actions/checkout@v3
34     - name: Use Node.js ${ matrix.node-version }
35       uses: actions/setup-node@v3
36
37       with:
38         node-version: ${ matrix.node-version }
39         cache: 'npm'
40
41     - run: npm install
42     - run: npm test

```

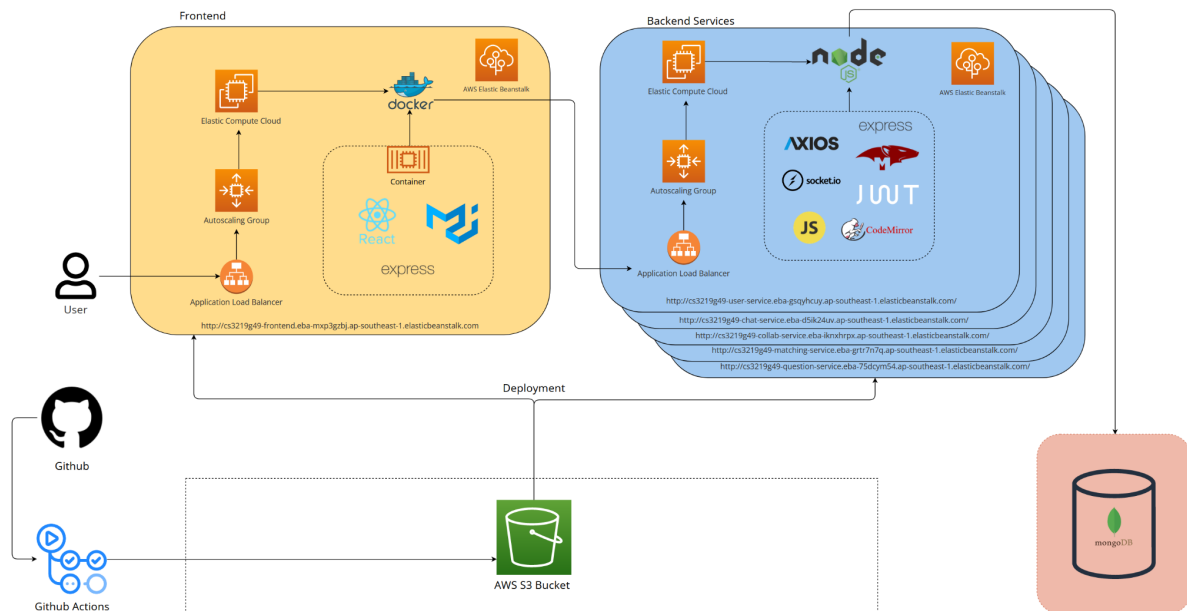
Github Actions verifies all test case passes before deployment



Deployment workflow will only run if the corresponding test workflow has completed without errors

4. Architecture

4.1 Architecture Diagram



4.2 Architecture Decision - AWS Elastic Beanstalk

AWS Elastic Beanstalk(EB) allows us to quickly launch web applications, supporting many platforms like node.js, Docker, Python and many more. AWS EB comes with many features, including load balancing and auto-scaling. AWS EB is a Platform-as-a-Service product, meaning that we do not have to manage lower level components like the Operating System and Runtime. This makes it easy and quick to deploy our application to the cloud without having to manually configure load balancers and EC2 instances.

By using AWS EB, we can also easily create our CI/CD pipeline for continuous deployment of our application. Using GitHub actions, we can create a deployment package which gets uploaded to AWS Simple Storage Service (S3). Thereafter, we can instruct AWS EB to pull in and deploy that deployment package. This way, we can achieve continuous deployment rather easily as compared to manually setting up our environments on EC2 instances.

4.3 Architecture Decision - Docker vs Node.js Runtime

For our deployment platform, we chose node.js for all of our backend services and a Docker platform for our frontend. This is because the frontend requires older dependencies, which was causing conflict in the dependency tree. This made it difficult to deploy on the node.js platform, which experienced problems while deploying the frontend. As such, we chose to dockerize our frontend and use AWS EB to deploy the docker container. We decided to not

dockerize all of our backend services as the docker container took considerably longer times to deploy as compared to directly deploying on the provided node.js runtime.

4.4 Architecture Decision - MongoDB Atlas

To select the database to store our data in, we primarily choose a NoSQL database as we are primarily working with object-like data in our application. If a relational model was chosen, more complex joins and operations will need to be done in order to query data. A document-based JSON object store is perfect for our needs.

As for the deployment of this database, we chose MongoDB Atlas as it has a well-documented and easy to use object modelling tool: Mongoose. AWS DocumentDB is similar as well, but does not provide much benefit at this point. At the scale that we are currently at, MongoDB would be sufficient and the benefit of a better documented and easier to use object modelling tool outweighs its benefit of better scaling. MongoDB loses out in scaling, but that would only come in handy when we need to store large amounts of data up to gigabytes and even terabytes of data, which will not be the case until much later.

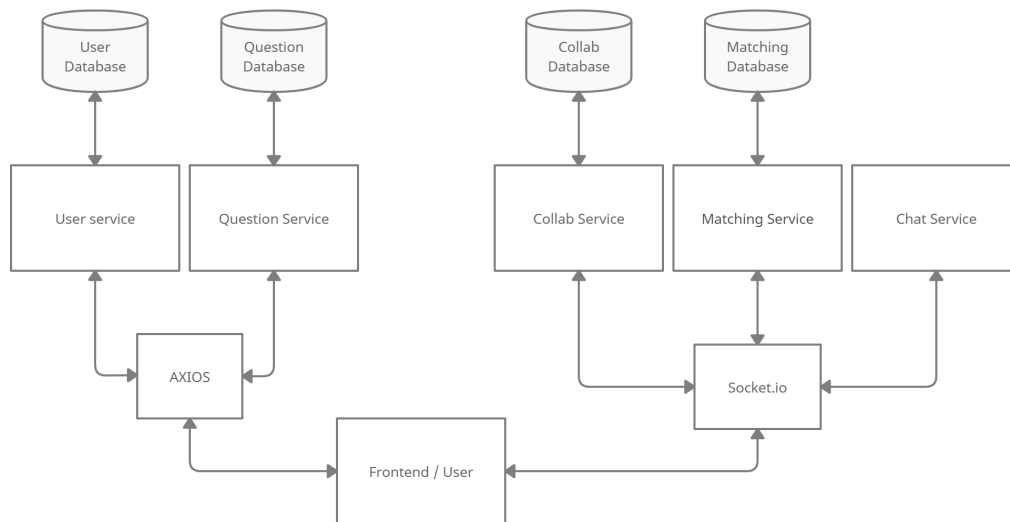
Each microservices has their own backing data store in a total of 5 MongoDB instances. This allows the Separation of Concerns between data models. In the event that one of the databases fails, only that particular microservice will be brought offline and will not affect the other microservices.

4.5 Architecture Description

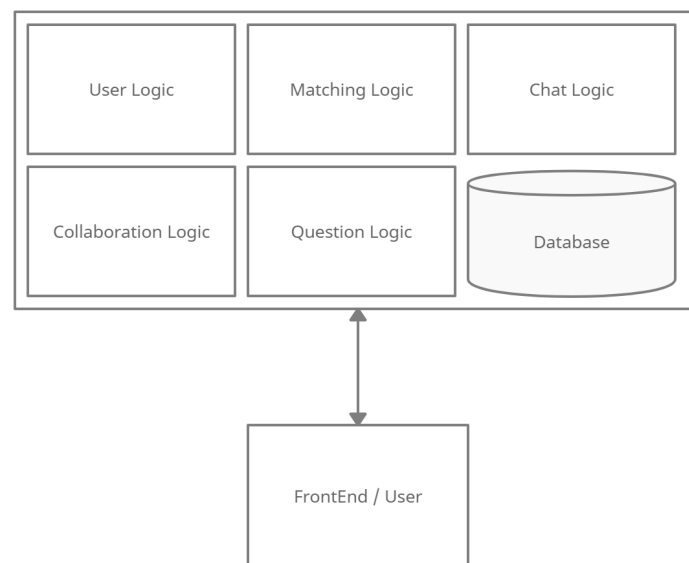
Peer Prep's frontend is built using React library and Material UI components.

Peer Prep's backend architecture is based on microservices. Backend components are divided into categories based on their responsibilities: user service, matching service, collaboration service, question service, and chat service. These components will have their own communication protocols with our React frontend, which will be documented in more detail in **Section 9. Microservices**

Below illustrates the architectural differences between a microservice architecture and a monolithic architecture. It can be observed that in the microservices architecture, all of the services and their respective databases are separate from one another and the failure of one will not affect another. Also, it is easily scalable as we can start another instance of a particular service while in monolithic, we would have to start another instance of the entire program which is costly due to the overhead. Therefore, our team has decided to implement a microservices architecture.



Architectural diagram of microservices structure



Architectural diagram of monolithic structure

The division of responsibilities based on bounded context ensures that while one service may be temporarily unavailable, other components may continue to function normally. For example, in the situation where users were matched into the same difficulty room, and the chat service experienced unexpected service offline, the collaboration service that keeps track of the common code editor can still serve clients code changes as usual, improving the application's availability.

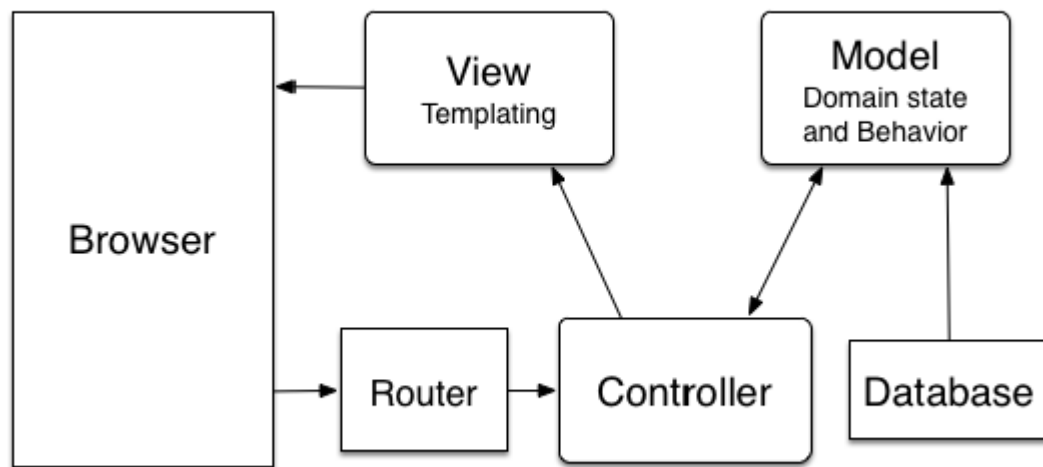
When user management is handled by the user service. Information that entails are username, password, user role and a unique user-id to identify each user.

The process of finding a match based on a matching algorithm is handled by a matching service. Peer Prep currently supports matching users based on the difficulty of technical questions, which users can select. When a match is found, a common room id is generated and passed to the collaboration service (collab) to create a shared workspace.

After obtaining the common room id, the collab service handles the initialization of a common room for users to collaborate in by joining them both to the same socket. Socket protocol offers real-time listening and publishing of events, outperforming http protocol in terms of latency.

Collab service's main responsibilities include obtaining a common question id from the question service and monitoring the common text editor for updates.

5. Design Pattern



MVC Variant Design Pattern

The core design of peer prep utilises a variant of MVC Model. As our application is built using the MERN stack (MongoDB, Express.js, React, Node.js), we achieve separation of concern which allows us to maintain and develop features independently. It is possible for us to switch tech stacks during the development phase without having to modify existing components.

Requests from the browser would be routed to their intended controllers. The controller consists of components from the various services that retrieve the schema from the model and pull the stored information from the database. This information is sent back to the controller, which passes it to the view and updates the browser.

The motivation to use MVC over the other MV* variant is the ease of testing individual components. With a clear separation of components, testing developers can independently verify each feature's functionality, allowing us to run our testing phase in parallel.

6. Tech Stack

Component	Technologies used	Reasons
FrontEnd	React	Reusable components
BackEnd	Express.js	Fast and easy to link to databases like MongoDB
Database	MongoDB	Able to handle high volume and scaling to accommodate high loads
Deployment	AWS S3, AWS Elastic Beanstalk, MongoDB Cloud Atlas	Easily scalable and cost effective scheme
Pub-sub Messaging	Socket io	Able to provide a low latency and bi-directional communication between server and client
Cloud providers	AWS, MongoDB Cloud Atlas	Easy to use and cost effective
CI/CD	GitHub Actions	Seamless integration as our repository is in Github
Orchestration service	Docker-Compose	Able to run multiple containers as services, similar to our microservices architecture
Project management service	Github Issues	Seamless integration as our repository is in Github

7. Devops

7.1 Testing

Continuous testing is done on every push to any branch. This workflow is triggered automatically using GitHub actions. The result will be available to see within a few minutes of this workflow run.

7.2 Local staging area deployment

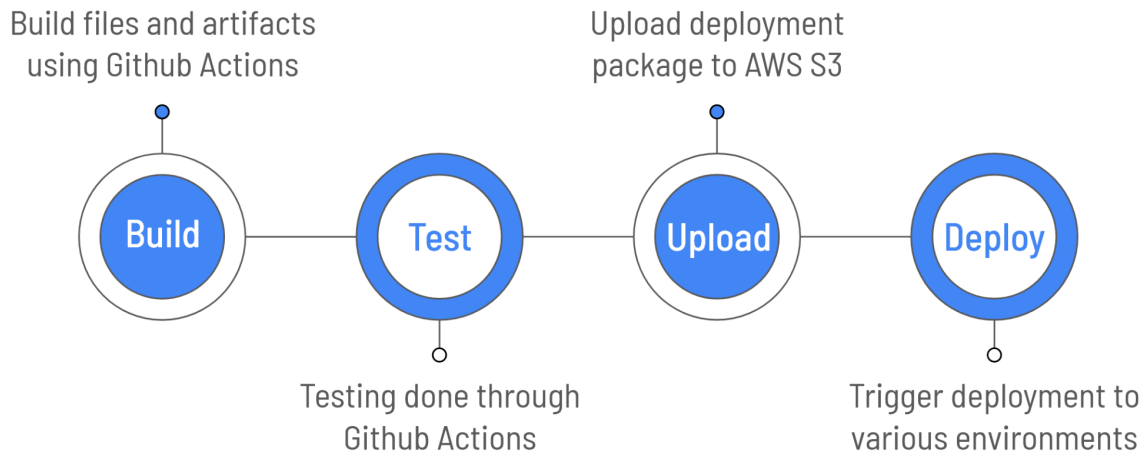
The deployment to the local staging area is done using Docker and Docker compose. When running Docker compose, Docker will build the 6 containers and expose their relevant ports, which allows the front end to communicate with the backend microservices.

```
[+] Building 107.2s (40/40) FINISHED
=> [cs3219g49-question-service internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [cs3219g49-collaboration-service internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [cs3219g49-frontend-service internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [cs3219g49-user-service internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [cs3219g49-chat-service internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [cs3219g49-matching-service internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [cs3219g49-question-service internal] load .dockerignore
=> => transferring context: 34B
=> [cs3219g49-collaboration-service internal] load .dockerignore
=> => transferring context: 34B
=> [cs3219g49-frontend-service internal] load .dockerignore
=> => transferring context: 34B
=> [cs3219g49-user-service internal] load .dockerignore
=> => transferring context: 34B
=> [cs3219g49-chat-service internal] load .dockerignore
=> => transferring context: 34B
=> [cs3219g49-matching-service internal] load .dockerignore
=> => transferring context: 34B
=> [cs3219g49-matching-service internal] load metadata for docker.io/library/node:alpine
=> [cs3219g49-matching-service 1/5] FROM docker.io/library/node:alpine
=> [cs3219g49-question-service internal] load build context
=> => transferring context: 598B
=> CACHED [cs3219g49-matching-service 2/5] WORKDIR /usr/src/app
=> [cs3219g49-question-service 3/5] COPY package*.json ./
=> [cs3219g49-question-service 4/5] RUN npm install
=> [cs3219g49-collaboration-service internal] load build context
=> => transferring context: 435B
=> [cs3219g49-collaboration-service 3/5] COPY package*.json ./
=> [cs3219g49-frontend-service internal] load build context
=> => transferring context: 8.51kB
=> [cs3219g49-user-service internal] load build context
=> => transferring context: 677B
=> [cs3219g49-frontend-service 3/5] COPY package*.json ./
=> [cs3219g49-chat-service internal] load build context
=> => transferring context: 220B
=> [cs3219g49-collaboration-service 4/5] RUN npm install
=> [cs3219g49-matching-service internal] load build context
=> => transferring context: 443B
=> [cs3219g49-user-service 3/5] COPY package*.json ./
```

Building the docker compose project to bring PeerPrep into local staging environment

7.3 Continuous Deployment

Our CI/CD process is done through GitHub actions. On every push to main, all tests will be run on individual workflows. If the tests fail, deployment will be stopped and reported in GitHub actions. When the test passes, the microservice will be automatically deployed to AWS as specified in 6.3.1 and 6.3.2.



A graphical view of our CI/CD process

7.3.1 Frontend deployment

Frontend is deployed to the AWS EB running docker. It uses GitHub actions to build the React app and upload this as a build artifact. The build files are then uploaded to AWS S3 and a deployment on AWS EB is triggered. AWS EB will then pull in the deployment package from our S3 bucket and update the application accordingly.

7.3.2 Backend services deployment

The other services (user service, matching service, question service, and chat service) are deployed to AWS EB using GitHub actions as our CI/CD tool. On every successful test, all services will be packed and uploaded to AWS S3. This will then trigger a deployment to AWS EB, which will deploy our microservices.

<input type="radio"/>	cs3219g49-chat-service	OK	Chat Service	2022-10-31 15:34:14 UTC+0800	2022-11-01 16:41:55 UTC+0800	cs3219g49-chat-service.eba-d5ik24uv.ap-southeast-1.elasticbeanstalk.com	ver-0ceb35a703f85117c836952b26771a6153b660a5
<input type="radio"/>	cs3219g49-collab-service	OK	Collaboration Service	2022-10-28 15:56:12 UTC+0800	2022-11-01 16:42:18 UTC+0800	cs3219g49-collab-service.eba-iknhrpx.ap-southeast-1.elasticbeanstalk.com	ver-0ceb35a703f85117c836952b26771a6153b660a5
<input type="radio"/>	cs3219g49-frontend	OK	Frontend	2022-11-02 20:43:03 UTC+0800	2022-11-02 21:11:38 UTC+0800	cs3219g49-frontend.eba-mxp3gzbj.ap-southeast-1.elasticbeanstalk.com	ver-3213eaa01ddf74dfe7ef22c8d10ad13af316ef06
<input type="radio"/>	cs3219g49-matching-service	OK	Matching Service	2022-10-28 14:34:41 UTC+0800	2022-11-01 16:42:12 UTC+0800	cs3219g49-matching-service.eba-grtr7n7q.ap-southeast-1.elasticbeanstalk.com	ver-0ceb35a703f85117c836952b26771a6153b660a5
<input type="radio"/>	cs3219g49-question-service	OK	Question Service	2022-10-27 15:06:37 UTC+0800	2022-10-27 16:28:35 UTC+0800	cs3219g49-question-service.eba-75dcym54.ap-southeast-1.elasticbeanstalk.com	ver-1796aa42ea254fd9c26ff3bdfb776cf4ece3d809
<input type="radio"/>	cs3219g49-user-service	OK	User Service	2022-10-27 16:30:05 UTC+0800	2022-11-01 16:42:21 UTC+0800	cs3219g49-user-service.eba-gsqhcyu.ap-southeast-1.elasticbeanstalk.com	ver-0ceb35a703f85117c836952b26771a6153b660a5

PeerPrep's backend services deployed on AWS EB

7.3.3 Deployment policy

Deployment is done through an immutable deployment process. When a deployment is triggered, a new instance of the application is created. Health checks are then performed to ensure that the application is up and running. All this while, all traffic is still served by the

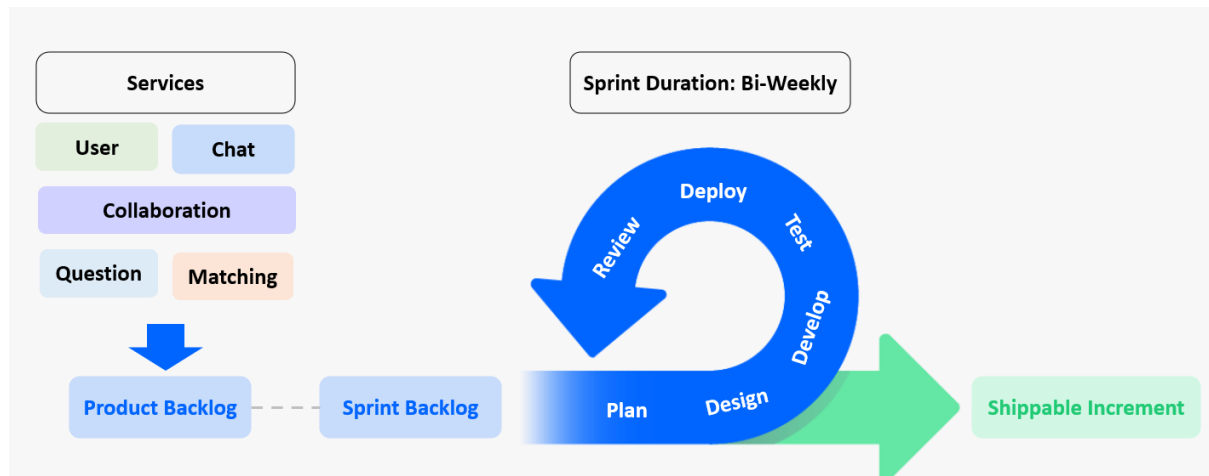
earlier version. Once the health checks are passed, this new instance is connected to the load balancer and the old version is taken down. If the health checks fail, the new instance gets terminated, essentially rolling back the deployment to the previous version.

Enhanced health overview									
Instances: 2 Total, 1 Ok, 1 Severe									
Learn more about enhanced health.									
<div>Filter by</div> <div>Instance actions</div> <div>Auto refresh (6 s)</div>									
Instance ID	Status	Running	Deployment ID	Requests/sec	2xx Responses	3xx Responses	4xx Responses	5xx Responses	
<div> <div></div> <div>Degraded</div> <ul style="list-style-type: none"> 28.6 % of the requests are failing with HTTP 5xx. ELB processes are not healthy on 1 out of 2 instances. Application update in progress on 1 instance. 0 out of 1 instance completed (running for 7 minutes). ELB health is failing or not available for 1 out of 2 instances. All instances are in same availability zone (ap-southeast-1a). </div>									
Overall		N/A	N/A	0.7	42.9%	28.6%	0.0%	28.6%	
<div> <div></div> <div>Severe</div> <ul style="list-style-type: none"> 100.0 % of the requests are failing with HTTP 5xx. Checking instance health (running for 2 minutes). Process default has been unhealthy for 18 seconds (Target.ResponseCodeMismatch). </div>									
i-0a028d92da070c87d		6 minutes	9	0.2	0	0	0	2	
i-0bab0107ec2a65075	Ok	1 day	8	0.5	3	2	0	0	

An example of a failed deployment run

As seen, the new deployment has been deployed and is currently failing its health checks. This would result in it being terminated after a few minutes. Note that the older deployment is still currently serving requests and thus users will still be able to use the application.

8. Software Development Lifecycle



A graphical view of our SLDC

We have adopted the scrum agile framework for the development, delivery, and maintenance of our Peer Prep project. At the start of every sprint, our team decides on tasks based on priority and assigns them to different members. Each sprint will last for two weeks. At the conclusion of each sprint, we will review everyone's work, identify any issues, deploy a shippable increment, and repeat.

Product Backlog						
User Story ID	User Story	Task Description	Link to FR	Priority	Sprint	Task Owner
US1	As a user, I want to be able to login into my account with my unique username and matching password.	Create Login Page Frontend	FR1.3	High	1	Keng I
		Backend Implementation for Login	FR1.3	High	1	Alex
US2	As a user, I want to be able to signup for a new account so that I will be able to use the service.	Create Signup Frontend	FR1.1, FR1.2	High	1	Keng I
		Backend Implementation for Signup	FR1.1, FR1.2	High	1	Alex
US3	As a user, I want to be able to logout of my account so that other people will not be able to access it on that same computer.	Enable Logout button on Main Page	FR1.4	High	2	Alex
		Backend Implementation for Logout	FR1.4	High	2	Alex
US4	As a user, I want to be able to delete the account I created when I am logged in, so that I can remove my information from the service completely.	Enable Delete Account button on Main Page	FR1.5	High	3	Keng I
		Backend Implementation for Delete Account	FR1.5	High	3	Keng I
US5	As a user, I want to be able to change my password when I am logged in, to my preferred new password	Create Change Password Frontend	FR1.6	High	4	Keng I
		Backend Implementation for Change Password	FR1.6	High	4	Alex
US6	As a user, I want to be able to select a level of difficulty of a question so that I can attempt a question within my skill range.	Create Matching Frontend	FR2.1	High	1	Jun Heng
US7	As a user, I want the system to be matched with another user searching for the same difficulty so that solving the questions will be within the same skill range for both of us.	Backend Implementation for User Matching	FR2.2	High	1	Zhan Qing
US8	As a user, I want to be able to know when I would get matched to another user and be redirected to the collaboration page immediately.	Connecting Matching Frontend and Backend	FR2.3	High	2	Jun Heng, Zhan Qing
US9	As a user, I want to be able to stop the matching process if I no longer wish to do a challenger		FR2.4	High	2	Jun Heng, Zhan Qing
US10	As a user, I want to communicate with my matched partner for us to collaborate on the question more efficiently.	Create Chat Service Frontend	FR3.1	Medium	3	Jun Heng
		Create Chat Service Backend	FR3.2	Medium	3	Jun Heng, Zhan Qing
US11	As a user, I want the application to support various difficulties and only get questions corresponding to the difficulty, so that the task is manageable for me.	Create Question Service Frontend and Backend	FR4.1, FR4.2, FR4.4	High	4	Keng I
US12	As an admin, I want to be able to add in new questions with it's corresponding difficulty levels so that the question database is always filled up with new challenges for the users	Create Add & Delete Question Frontend	FR4.3	High	4	Keng I
US13	As a user, I want to have a room dedicated to my matched partner and I, so that we have a common interface to work on the question.	Create Collaboration Service Frontend	FR5.1	High	3	Jun Heng
		Create Collaboration Service Backend	FR5.1	High	3	Zhan Qing
US14	As a user, I want to be able to save my collaboration code so that I can continue coding at another time as long as the session is still active	Enhance Collaboration Service Backend	FR5.3	Medium	4	Zhan Qing
US15	As a user, I want to see syntax highlighting on my Javascript code, so that I can focus my attention on code logic instead when solving the questions	Enable Styling on CodeMirror	FR5.2	Low	4	Zhan Qing
US16	As a user, I want to be redirected back to my existing matching session if I happen to exit the page accidentally so that I can continue with my training.	Enable collaboration checks on all frontend pages	FR5.4	High	4	Jun Heng

A product backlog table tracking all requirements and priority

Using the product backlog table above, we can quickly determine the link between FRs and priority for each task. The sprint backlog can then be determined more efficiently based on this table.

9. Microservices

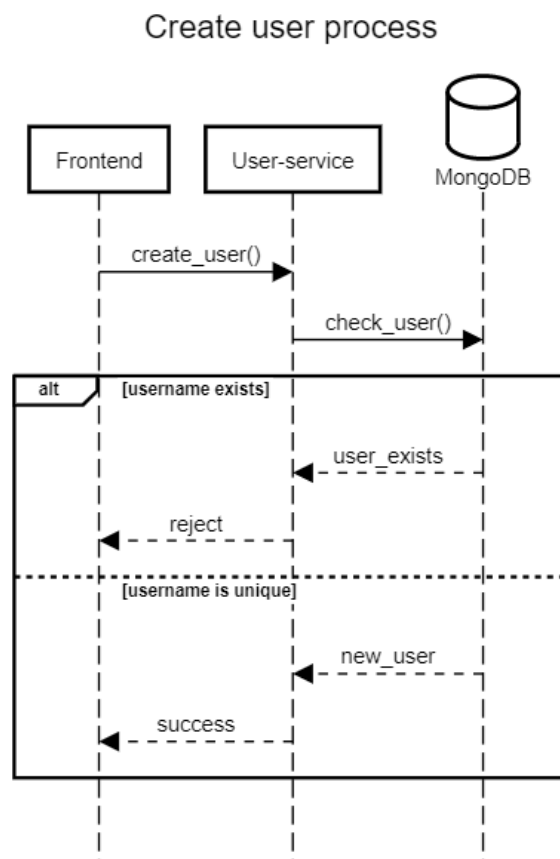
9.1 User Service

JWT Tokens

JWT Tokens are used to authenticate and authorise our users. Once users have logged in using the proper credentials, our user service will generate and sign a JWT token using a secret key. The token is delivered to the user and saved as a browser cookie. The user will be logged in automatically without having to enter his credentials if the token is still valid when he accesses Peer Prep the following time. A valid token meets two criteria:

- Token has not been tampered.
- Token has not been blacklisted.

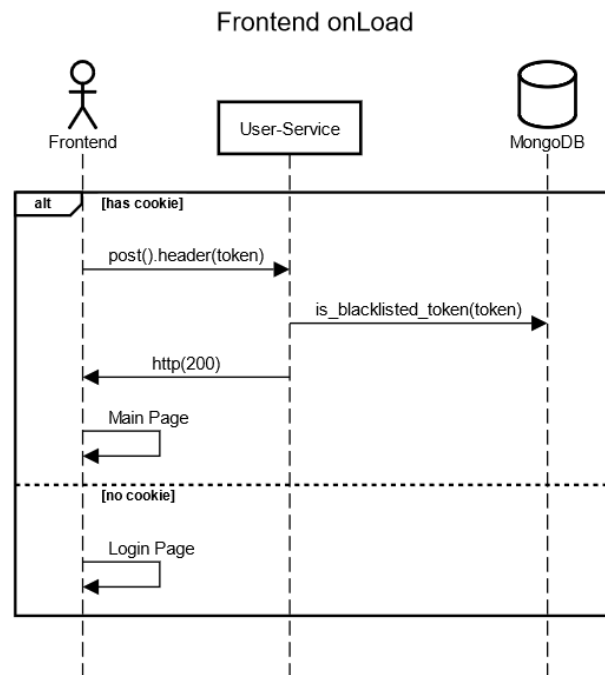
Create User Implementation



An sequence diagram showing interactions between services when creating a user

Create user process: When the user service receives a request to create a user, it determines whether the requested username already exists in the database. Only unique usernames are permitted in Peer Prep's user service. User service will reject existing username requests with the reserved status code 409 and respond with success otherwise.

Session Implementation

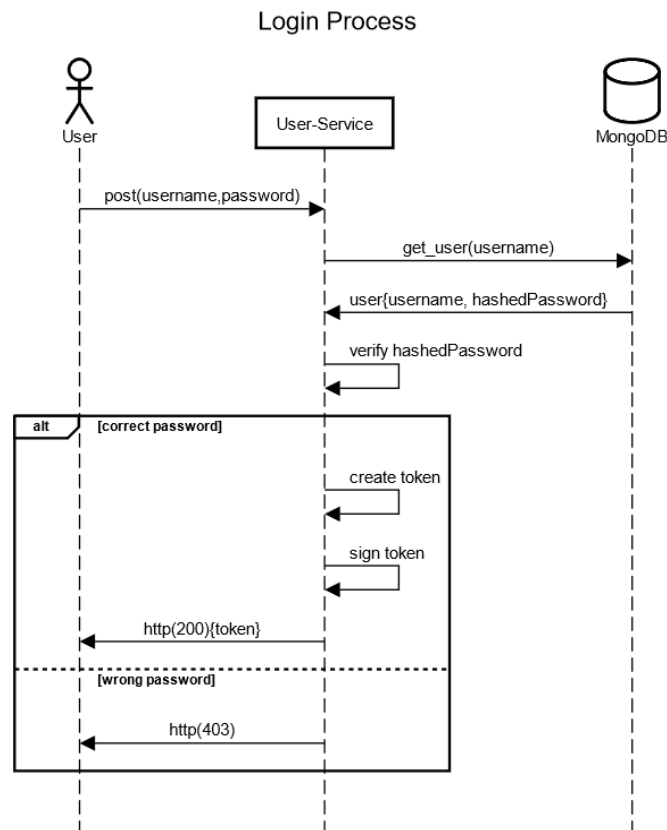


An sequence diagram showing interactions between services when the frontend is loaded

When the user first visits the frontend, the frontend checks if there is any existing cookie named `access token` stored in the browser. If this token exists, it will send a HTTP POST request to the user service with this token attached in the request's authorization header.. The user service will verify this token and check if it is blacklisted. Blacklisted tokens are those that are no longer functional because the user has already made a request to log out. This prevents a token from being reused for subsequent login sessions after it has been deemed invalid as a result of a logout request.

If it passes both checks, it will send a HTTP 200 response back, and the frontend will render the mainpage to the user. If it fails any of the checks, the frontend will remove the cookie from the browser and render the login page to the user instead.

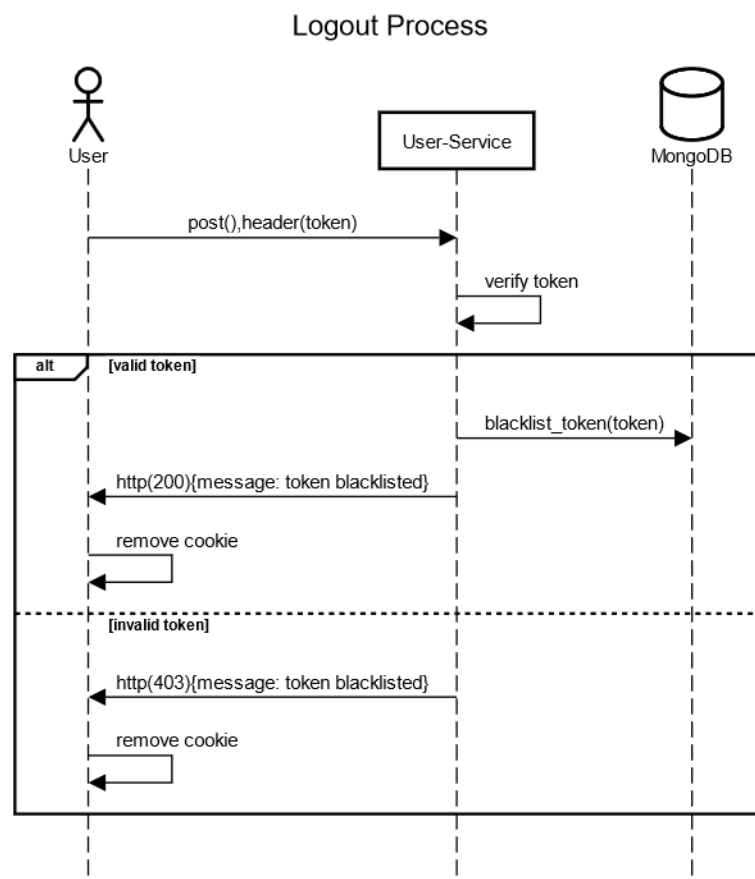
Login implementation



An sequence diagram showing interactions between services when a user logs in

Login sequence: When the user attempts to login, the frontend sends a HTTP POST request containing the user's credentials to the user service. The user service gets the user's hashed password from the database and verifies it. If the password is correct, it will generate a signed JWT and return it to the frontend to be stored as a cookie. If the authentication fails, a HTTP 403 is returned. If any of the fields are missing or empty after trimming leading whitespaces, an error code 400 will be returned.

Logout Implementation

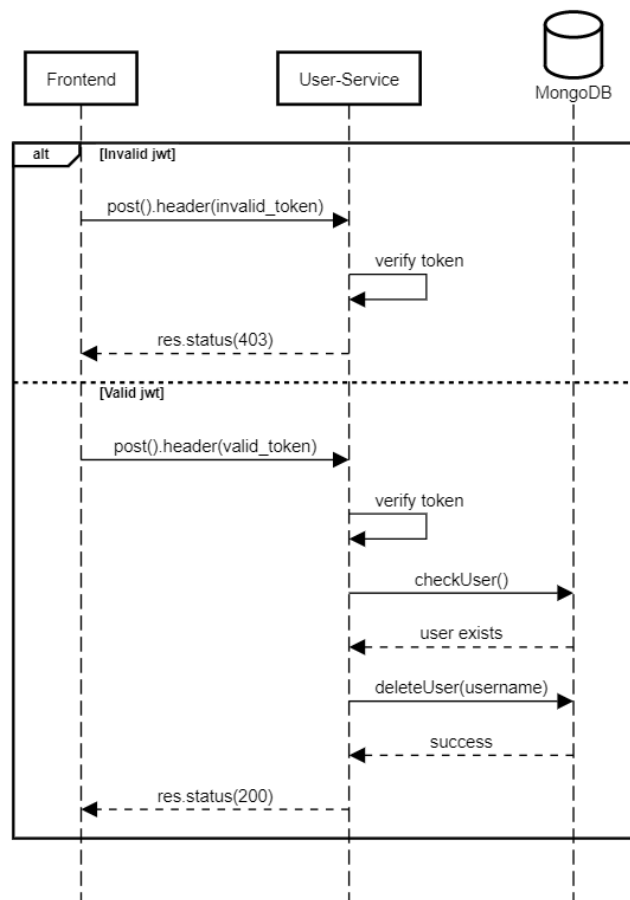


An sequence diagram showing interactions between services when a user logs out

Logout sequence: When the user logs out, the frontend sends a HTTP POST request to the user service with the JWT on the request header. The user service verifies this token. If the token is verified, it will create a new entry in the blacklisted token database and send a HTTP 200 response, indicating a successful logout. The stored cookie will be subsequently deleted from the browser. If the JWT is invalid, a HTTP 403 is sent and nothing will be executed.

Delete User Implementation

Delete user process



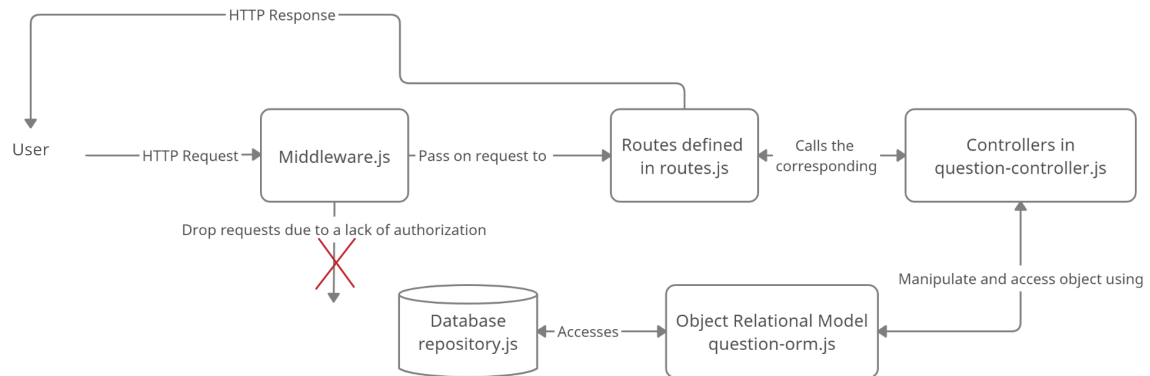
An sequence diagram showing interactions between services when deleting a user

User makes a post request to user service, containing the JWT token in the header. The user service verifies this token and rejects the request if the JWT is invalid. This ensures only users who have a valid login session can delete their own account. User service then proceeds to check if the username exists in the database. If a user does exist, a delete request will be made to MongoDB using the username and respond to the frontend if it is successful. All intermediate errors will result in a status code of "500 internal error responses sent to the frontend.

9.2 Question Service

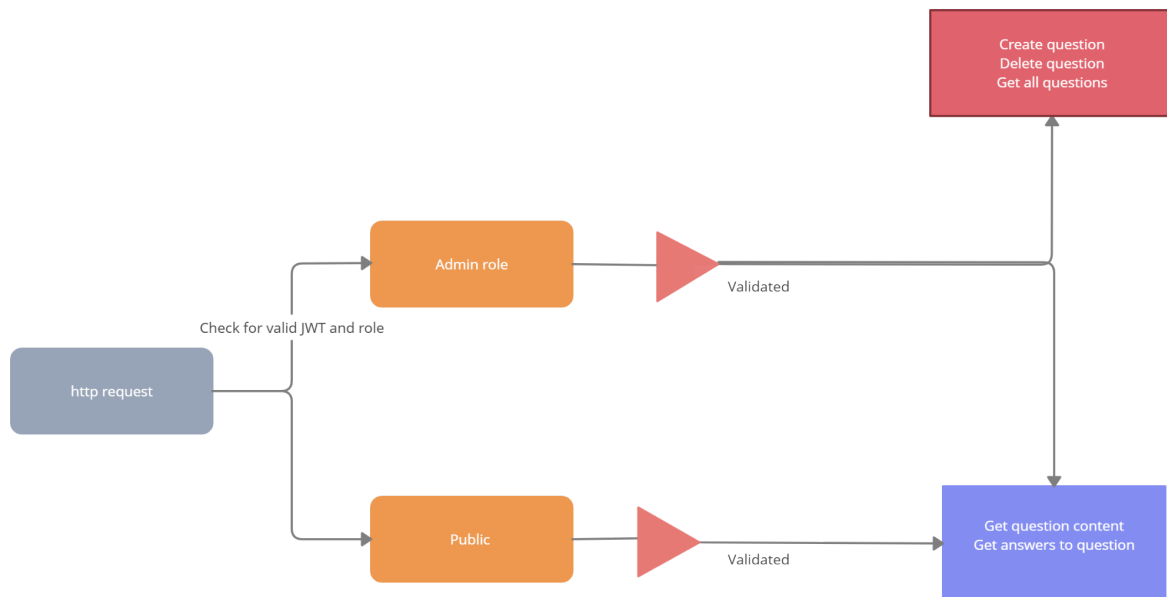
Question service is a microservice that communicates only with collab service and question database. This service is responsible for maintaining a database of questions of varying difficulty and supplying questions to the collab service whenever a new collaboration room opens.

Question service is built using Express JS. The breakdown of the structure of question service is as follows:



A high level implementation diagram of question service

Authorization paths



Authorization paths and their resources for question service

Question service is designed to be publicly accessible to allow requests to get questions from the database. Therefore, most of our APIs have a **'public'** route, returning the response without any requirements.

Question service also contains APIs that help to manage the collection of questions that we have. Because CRUD of question records could have an impact on the availability of our program if misused, we restrict access to the APIs that are responsible for them to **'admin'** only. In order to make these requests, accounts must have the **'admin'** role in their JWT token. Consequently, requests without the admin role will be rejected with http status code 403 Forbidden.

API supported:

- Create question (reserved for admins)
- Delete question (reserved for admins)
- Get all questions (reserved for admins)
- Get random question ID based on difficulty
- Get question content based on question ID
- Get question answer based on question ID
- Get maximum number of question of a specific difficulty

Questions Model

Each question has a question ID, question content, question answer.

Question ID is the id value generated by MongoDB when the document is created. As this value is unique, we decided to use it as our question ID as well.

Question content and answers are both required information during the creation process to ensure database consistency.

Design Considerations

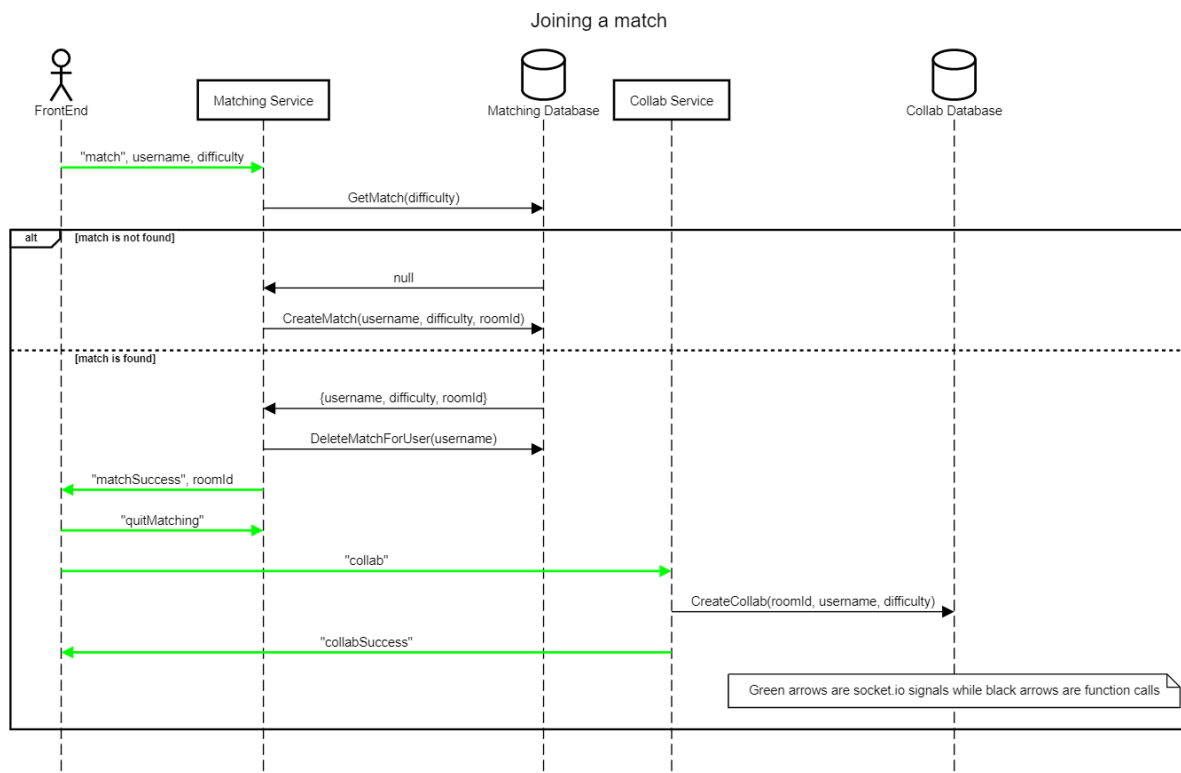
We decided to divide obtaining questions into two steps: obtaining a random question id and then obtaining the question content. Rather than obtaining the entire question, including the content, obtaining only the id helps to reduce resource waste. In the case where the random id generator returns an attempted question id, the frontend has to repeat the http request again. Since the server did not include the entire content in its response payload, this can allow for better performance in time during the skipping process. As a result, this division aids future extensions such as skipping questions and keeping track of the order of questions attempted so that users can navigate through our list of questions without getting a repeat.

9.3 Matching Service and Collaboration Service

Matching Service is in charge of matching two users searching for a match with the same difficulty (i.e., a Match). Once a Match is found, it will remove the corresponding entry from the database and return to the frontend the same roomId for both users who are matched.

Collab Service will get the room ID from the frontend and create an entry in the collab database. (There will be 2 entries in the end as 2 users will be matched with each other and they will both create an entry with the same collab-id in the collab database)

Below is an illustration of how the control flow of collab and matching services work. Note that green arrows are socket.io signals with the signal name in quotes.

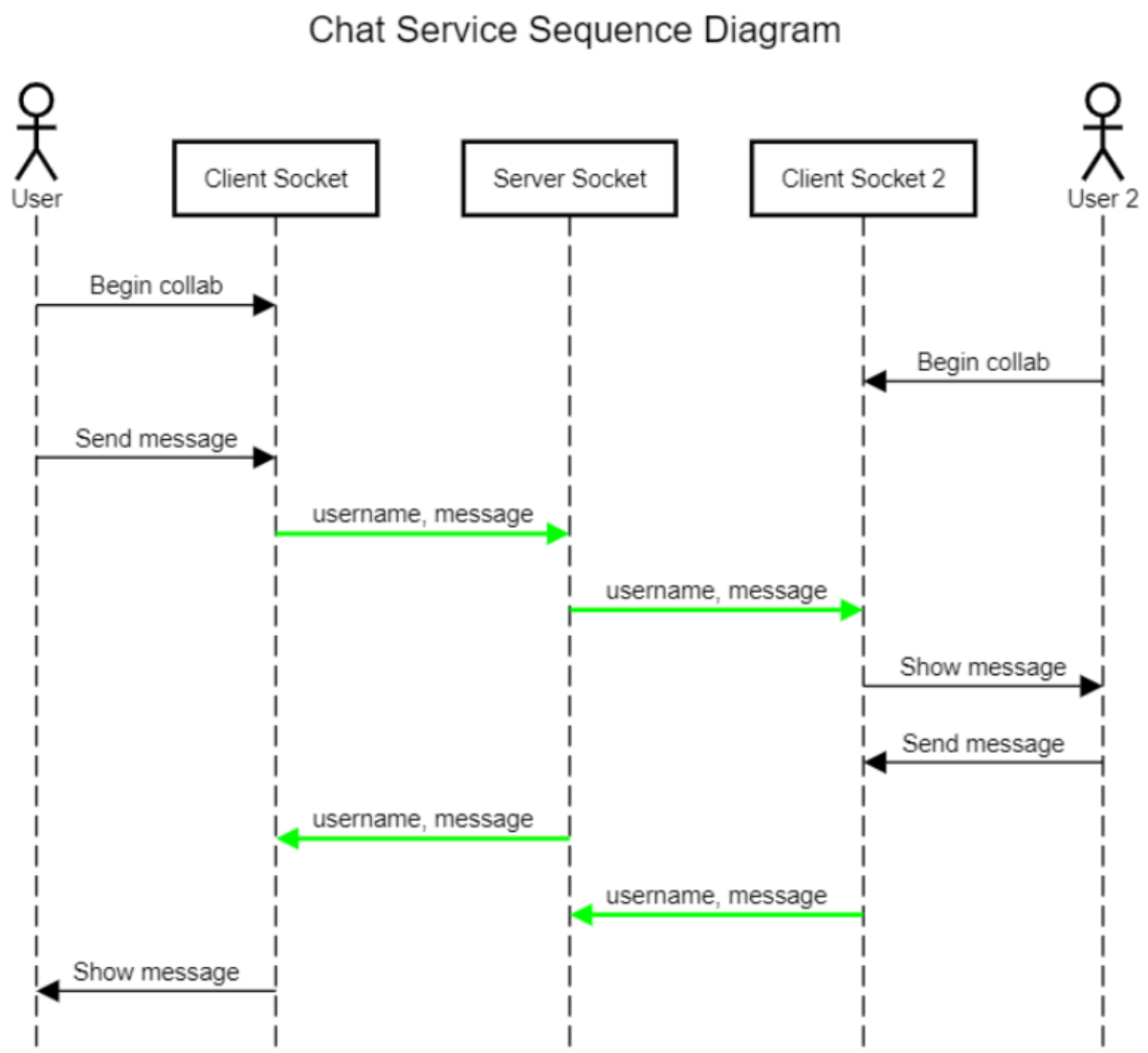


A sequence diagram showing interactions between matching and collaboration service when matching users

When joining a match, the frontend will call Matching Service to check if an existing user who is finding a match with the same difficulty is present in the matching database. If no matches are found, a match containing the username, difficulty, and roomId will be created.

Suppose otherwise, and an existing user searching for the same difficulty exists in the database; matching service will get the match and delete it from the database. This match will be returned to the frontend together with the 'matchSuccess' signal to the socket. The frontend will quit the matching service (by leaving the socket room) with a signal "quiteMatching" and signal to the collab service with a "collab" signal. Collab Service would create an entry in the collab database and return to the frontend a success signal.

9.4 Chat Service



A sequence diagram showing the implementation of chat service

The chat service is in charge of allowing communications between 2 users. Similar to the collab service, this service uses socket.io to enable low-latency, bidirectional, and event-based communication between them.

Upon the creation of the collab room, both clients' chat sockets will join the newly established collab room using its unique ID. For every message sent, the sockets emit signals based on collab room IDs with the username and message tags attached to them. On the other user's end, the message and the user's name will be displayed when this signal is received. The communication is bidirectional, as per the diagram above.

10. Implemented Features

Must Have			
1	User Service	2	Collaboration Service
3	Matching Service	4	Basic UI
5	Question Service	6	Local Machine Deployment using Native Tech Stack
Nice to Have			
1	Chat Service	2	AWS App Deployment
3	Fancy UI	4	Effective Use of CI/CD
5	Deployment to Local Staging Environment using Docker Compose		

11. Member Contributions

	Contributions	
S/N	Task Description	Task Owner
Technical Contribution		
1	Backend Implementation for Login	Alex Teo
2	Backend Implementation for Signup	
3	Enable Logout button on Main Page	
4	Backend Implementation for Logout	
5	Backend Implementation for Change Password	
6	AWS Deployment Setup	
7	Unit Testing	
8	CI/CD	
9	Dockerization	
10	Create Login Page Frontend	Keng I
11	Create Signup Frontend	
12	Enable Delete Account button on Main Page	
13	Backend Implementation for Delete Account	
14	Create Change Password Frontend	
15	Create Question Service Frontend and Backend	
16	Create Add & Delete Question Frontend	Jun Heng
17	Create Matching Frontend	
18	Create Chat Service Frontend	
19	Create Collaboration Service Frontend	Jun Heng, Zhan Qing
20	Enable collaboration checks on all frontend pages	
21	Connecting Matching Frontend and Backend	Zhan Qing
22	Create Chat Service Backend	
23	Backend Implementation for User Matching	
24	Create Collaboration Service Backend	
25	Enhance Collaboration Service Backend	
26	Enable Styling on CodeMirror	
Non-Technical Contribution		
27	Slides, Project Documentation, ReadMe	Alex Teo
28	Slides, Project Documentation, ReadMe	Keng I
29	Slides, Project Documentation, ReadMe	Jun Heng
30	Slides, Project Documentation, ReadMe	Zhan Qing

12. Suggested Enhancements

A suggested enhancement for the next iteration is to allow users to enter collaboration rooms without finding a match. By enabling this feature, our site will be able to work even with a limited user base. The following is a description of how it would work. If no other user with the same difficulty has been found after 30 seconds, we will allow users to match with themselves. This means that a user will be able to view the collaboration question page as if he were in a room with someone else.

Another enhancement that our team has considered for the next iteration is the addition of test cases for socket.io. Similar to how we had test cases for api calls (i.e, user service), implementing test cases for sockets would help detect regression bugs before deployment - thus aiding us in ensuring our application's overall reliability.

Another enhancement that our team was keen to work on is the integration of email addresses and profiling for each account. Currently, accounts are simply a means to match two users together. If we can capture more information on user preferences and statistics, it can help us develop better matching algorithms that cater to each group of users. In addition, having an email address would allow users who have lost access to their account, retrieve access again, possibly through a unique link that resets passwords.

Appendix A : Terms

No.	Term	Meaning
1	FR	Functional Requirements
2	NFR	Non-Functional Requirements
3	Match	2 users searching for the same difficulty
4	Collab	Collaboration service
5	AWS	Amazon Web Services
6	AWS EB	Amazon Web Services Elastic Beanstalk
7	AWS EC2	Amazon Web Services Elastic Compute Cloud
8	AWS S3	Amazon Web Services Simple Storage Service
9	CI/CD	Continuous Integration/Continuous Deployment
10	API	Application Programming Interface
11	JWT	JSON Web Token
12	SLDC	Software Development Life Cycle