PeerPrep

1) Background and Purpose

Increasingly, students face challenging technical interviews when applying for jobs which many have difficulty dealing with. Issues range from a lack of communication skills to articulate their thought process out loud to an outright inability to understand and solve the given problem. Moreover, grinding practice questions can be tedious and monotonous.

To solve this issue, we created an interview preparation platform and peer matching system called PeerPrep, where students can find peers to practise whiteboard-style interview questions together. A general description of the application is as follows:

A student who is keen to prepare for his technical interviews visits the site. He creates an account and then logs in. After logging in, the student selects the question difficulty level he wants to attempt today, i.e., Easy, Medium, or Hard. The student then waits until he is matched with another online student who has selected the same difficulty level as him. If he is not successfully matched after 30 seconds, he times out. If he is successfully matched, the student is provided with the question and a free text field in which he is expected to type his solution. This free text field should be updated in near-real time, allowing both the student and his match to collaborate on the provided question. He is also able to chat with his match. After the students finish working on the question and are ready to end the session, any of them clicks on a “Finish Session” button which returns each student to the question difficulty selection page. From this page, the student is able to update his password by navigating to the profile page, otherwise he logs out.

2) Individual Contributions

Members	Contributions
Desmond To Cie Hong	Technical ● Frontend ○ Signin page: PR#3 ○ Unit test: PR#6 ○ Alert message context: PR#22 ○ User context: PR#25 ○ Fix update profile message bug: PR#74 ○ Fix navbar UI bug or add functionality: PR#75, PR#84 ○ UI responsiveness: PR#78 ○ Development setup in Docker: PR#79 ○ Fix backend development in Docker bug: PR#83 ○ Enhance UIUX of the application: PR149 ● Communication service ○ Basic text messaging: PR#30 ○ Leave session: PR#48 ○ Chat persistence when refreshing page: PR#51 ● Collaboration service ○ Leave session: PR#48 ● User service ○ Password salting and hashing: PR#7 ○ Signin and signup: PR#22 ● CI: PR#4 ● CD (Continuous Deployment): ○ CD: PR#88 ○ Secrets and environment variable: PR#92, PR#93 ○ Deployment to Google Cloud Platform: PR#135 ○ Optimise deployment of application with smaller image: PR#153 ● README: PR#157, PR#158
	Non-Technical ● PR reviews: PR#5, PR#8, PR#19, PR#20, PR#23, PR#24, PR#28, PR#36, PR#44, PR#50, PR#54, PR#77, PR#80, PR#81, PR#87, PR#133, PR#136, PR#146 ● Project report, such as FR, NFR, architecture diagram, and activity diagram
Liew Xin Yi	Technical ● Frontend ○ Match page: PR#28 ○ Matching logic and bug fixes: PR#33, PR#41, PR#53, PR#87, PR#133, PR#146 ● User service ○ Sign in logic: PR#8 ● Matching service ○ Setup and matching logic: PR#20 ● Auth service

	○ Split “user service” into user service and auth service: PR#136 ● Gateway ○ Reverse proxy: PR#54 ○ Authentication subrequests: PR#136 ● Docker: ○ Docker Compose: PR#44, PR#77 ○ Live reload: PR#80, PR#81 ● CI/CD: ○ CI: PR#2 ○ CD: PR#136
	Non-Technical ● PR reviews: PR#7, PR#30, PR#45, PR#46, PR#79, PR#83, PR#84, PR#135 ● Project report, such as FR, NFR, and activity diagram (matching logic)
Ng Hsiao Jiet	Technical ● Frontend ○ UI/UX prototype (Adobe XD) ○ room page, home page: PR#5, PR#23, PR#24 ○ reusable UI components such as difficulty buttons, question box, chat box and coding editor ● Question service: PR#35, PR#50 ○ logic to create and read the technical interview questions ○ communication between Matching service to ensure matched users can view the same question in the same room ○ display a random question of the selected difficulty by the user
	Non-Technical ● PR reviews: PR#3, PR#5, PR#17, PR#22, PR#23, PR#24, PR#25, PR#35, PR#36, PR#41, PR#48, PR#50, PR#51, PR#59, PR#78, PR#90, PR#149 ● Project report, such as FR, NFR, and architecture diagram, pattern diagrams, database diagram
Pua Li Xue	Technical ● User service: PR#19, PR#46 ○ prevent creation of duplicate usernames logic: PR#17 ○ logout logic ○ delete account logic ○ JWT: generation, authentication, blacklisting ● Collaboration service: PR#36, PR#45 ○ connect matched users to same collaborative code editor ○ logic to sync collaborative code editor using Socket.IO ○ storing of code in Redis such that it can be loaded if user refreshes page: PR#59

	<ul><li>Frontend<ul><li>collaborative code editor: PR#36</li></ul></li></ul>
	Non-Technical<ul><li>PR reviews: PR#6, PR#22, PR#28, PR#30, PR#33, PR#35, PR#41, PR#53, PR#74, PR#75</li><li>Project report, such as FR, NFR, and sequence diagram</li></ul>

3) Functional Requirements

User Service

S/N	Functional Requirement	Priority	Completion
FR1.1	The system should allow users to create an account using their username, email and password.	High	✅
FR1.2	The system should ensure that every account created has a unique username and email.	High	✅
FR1.3	The system should allow users to delete their account.	High	✅
FR1.4	The system should allow users to change their password.	Medium	✅

Matching Service

S/N	Functional Requirement	Priority	Completion
FR2.1	The system should match two different users who selected the same difficulty level.	High	✅
FR2.2	If there is a valid match, the system should match the users within 30s.	High	✅
FR2.3	If a valid match cannot be found within 30 seconds, the system should inform the user that no match is available.	High	✅
FR2.4	The system should delete the corresponding match request when the user disconnects.	High	✅

Question Service

S/N	Functional Requirement	Priority	Completion
FR3.1	The system should provide 3 question difficulty levels that users can choose from, i.e., Easy, Medium, Hard.	High	✅
FR3.2	The system should provide information about the question, i.e., title, difficulty, categories, content.	High	✅
FR3.3	The system should allow users to view their past attempts with the associated statistics, i.e., date of attempt, duration, peer who attempted together, etc.	Low	
FR3.4	The system should provide question categories that users can choose from, i.e., Data Structure, Dynamic Programming, Sorting Algorithms, etc.	Low	

Collaboration Service

S/N	Functional Requirement	Priority	Completion
FR4.1	The system should join the two matched users into the same room.	High	✅
FR4.2	The system should allow the two matched users to edit code in the same code editor.	High	✅
FR4.3	The system should send users back to the question difficulty selection page when his/her match has disconnected.	High	✅
FR4.4	The system should let the user know when his/her match has disconnected.	High	✅
FR4.5	The system should ensure that the user leaves the room after 10 seconds since his/her match disconnects.	Medium	✅
FR4.6	The system should sync the code when the user refreshes the page.	Medium	✅

Communication Service

S/N	Functional Requirement	Priority	Completion
FR5.1	The system should join the two matched users into the same room.	High	✅
FR5.2	The system should allow the two matched users to chat with each other with text.	High	✅
FR5.3	The system should ensure that the user leaves the room immediately when s/he finishes the session.	High	✅
FR5.4	The system should notify the users if it is themselves or their peer who left the room.	High	✅
FR5.5	The system should sync the chat messages when the user refreshes the page.	Medium	✅
FR5.6	The system should delete all the chat messages when the two matched users leave the room.	Medium	✅
FR5.7	The system should ensure that the user leaves the room after 10 seconds since his/her match disconnects.	Medium	✅
FR5.8	The system should store all the chat messages with respect to the room which the two matched users joined.	Medium	✅
FR5.9	The system should allow the two matched users to communicate with each other using audio calls.	Low	

Auth Service

S/N	Functional Requirement	Priority	Completion
FR6.1	The system should allow users to log into their accounts by entering their username/email and password.	High	✅
FR6.2	The system should allow users to log out of their account.	High	✅
FR6.3	The system should authenticate protected backend requests, i.e., HTTP requests and Socket.io connections, from users.	High	✅
FR6.4	The system should authenticate protected frontend requests, i.e., HTTP requests and Socket.io connections, from users.	High	✅

Frontend

S/N	Functional Requirement	Priority	Completion
User Service			
FR7.1	The system should display a sign up form with username, email and password inputs for users to sign up for an account.	High	✅
FR7.2	The system should display a delete button for users to delete their account.	High	✅
FR7.3	The system should display the information, i.e., username, email and password, that the users used to sign up for their account on the signup page.	Medium	✅
FR7.4	The system should display an edit button for users to update their password.	Medium	✅
Matching Service			
FR7.5	The system should display three difficulty level buttons, i.e., Easy, Medium and Hard, for users to select the difficulty level of the question that they wish to attempt.	High	✅
FR7.6	The system should display a timer to keep users informed of the time left during the matching process.	Medium	✅
FR7.7	If a valid match cannot be found, the system should display two action buttons, i.e., Retry and Cancel, for users to decide on their next action.	Medium	✅
Communication Service			
FR7.8	The system should display a list of messages between the two matched users.	High	✅

FR7.9	The system should display an input field to type chat messages.	High	✅
FR7.10	The system should display a send button to send chat messages.	High	✅
FR7.11	The system should display a scrollbar to view previous chat messages.	High	✅
FR7.12	The system should display the author name of the chat messages.	High	✅
FR7.13	The system should scroll the chats to the bottom of the list once sending its own message and stay at the same place when peer sent some messages to prevent missing out unread messages.	Low	✅
FR7.14	The system should allow the users to drag the chat icon to prevent it from blocking the view after opening the chat	Low	✅
Collaboration Service			
FR7.14	The system should display a code editor for users to type in.	High	✅
FR7.15	The system should display a scrollbar to view the full code when the code gets long.	High	✅
FR7.16	The system should display a finish button for users to leave the room after they are done.	High	✅
FR7.17	The system should redirect the user to the difficulty level selection page when he finishes the session.	High	✅
FR7.18	The system should redirect the user to the difficulty level selection page and notify him when his match disconnects after 10 seconds.	Medium	✅
Question Service			
FR7.19	The system should display the correct question based on the difficulty level chosen by the user.	High	✅
FR7.20	The system should display the information of the questions, i.e., title, difficulty, categories, content.	High	✅
Auth Service			
FR7.21	The system should display a sign in form with username/email and password inputs for users to sign in to their account.	High	✅
FR7.22	The system should display a sign out button for users to sign out of their account.	High	✅
FR7.23	The system should have an indicator to reflect the sign in/out status of users.	Medium	✅

System Wide			
FR7.24	The system should prompt users with appropriate error messages when encountering errors.	High	✅
FR7.25	The system should prompt users with possible fixes after encountering errors.	Medium	✅
FR7.26	The system should prompt users of their actions taken, e.g., “Login successful.”	Medium	✅
FR7.27	The system should display a not found page when the user enters an invalid path for the website.	Low	✅
FR7.28	The system should display a loading screen when the serving of the HTML pages from the server is delayed.	Low	✅

Deployment

S/N	**Functional Requirement**	**Priority**	**Completion**
FR8.1	The system should be deployable on a local machine.	High	✅
FR8.2	The system should be deployable using Docker Compose.	Medium	✅
FR8.3	The system should be deployed on Google Cloud Platform (GCP).	Medium	✅
FR8.4	The system should have an NGINX API gateway that serves as a reverse proxy and handles authentication subrequests.	Medium	✅
FR8.5	The system should have an effective CI/CD workflow.	Medium	✅

4) Non-Functional Requirements

Performance

We focus on providing a short response time to provide a good user experience. This is especially important when users are collaborating and communicating while writing their code.

S/N	Non-functional requirement	Priority	Completion
NFR1.1	Authorization of user login should take no more than 2 seconds.	Medium	✅
NFR1.2	Database should allow retrieval of questions within 3 seconds.	High	✅
NFR1.3	The system should allow code editing collaboration to be done in near real-time, i.e., within 1 second.	High	✅
NFR1.4	The system should transmit messages onto the chat box sent by users within 3 seconds.	High	✅
NFR1.5	The system should load the views within 2 seconds.	High	✅

Usability

We aim to provide an intuitive interface such that users, including new users, are able to use the app easily. They should be able to navigate through different pages without confusion. Clear error messages or prompts should be provided to help users correct their actions when necessary.

S/N	Non-functional requirement	Priority	Completion
NFR2.1	The system should be intuitive enough or well guided by prompts for users to navigate.	High	✅
NFR2.2	The system should use a suitable font size so that users are able to read clearly.	High	✅
NFR2.3	The system should show a clear error message when some actions fail.	High	✅
NFR2.4	The system should prompt a confirmation from users when performing dangerous actions (e.g. deleting their account).	High	✅
NFR2.5	The system should notify the users after successful actions (e.g. when chat is sent successfully).	Medium	✅

Security

Users should be confident that their sensitive information is stored securely. Their accounts should not be compromised easily and accessed by impersonators. Thus the following non-functional requirements aim to satisfy the above situations.

S/N	Non-functional requirement	Priority	Completion
NFR3.1	Users' passwords should be hashed and salted before storing in the DB.	Medium	✅
NFR3.2	Users' passwords should be at least 8 characters long.	Medium	✅
NFR3.3	Users' passwords should contain at least 1 special character and 1 numeric character.	Medium	

Scalability

We would like to ensure that users do not face issues accessing the services due to an increased load. Building a scalable application also enables us to reach more users in the future.

S/N	Non-functional requirement	Priority	Completion
NFR4.1	The system should support up to 5 concurrent users.	Medium	✅
NFR4.2	Database should allow more than 50 questions to be recorded.	Medium	✅

Portability

Users shall be able to access our web application using their preferred modern web browser. Thus, we aim to build our application that functions and looks similarly in different browsers.

S/N	Non-functional requirement	Priority	Completion
NFR5.1	The system should be able to run on Chrome.	High	✅
NFR5.2	The system should be able to run on Firefox.	High	✅
NFR5.3	The system should be able to run on Safari.	High	✅
NFR5.4	The system should be able to run on Microsoft Edge.	High	✅

5) Developer Documentation

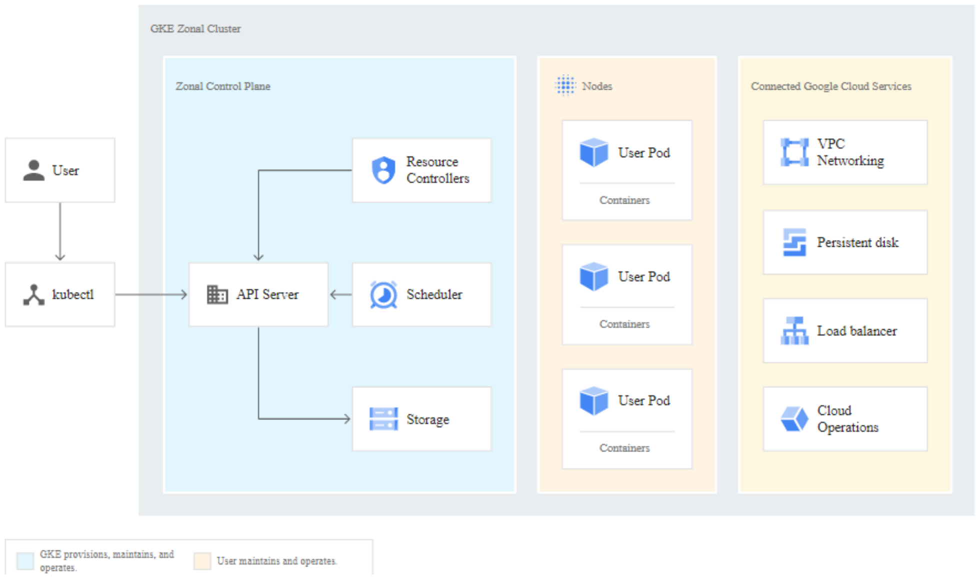
Development Process

Github workflow

Basically, we followed the forking workflow taught in CS2103/T to develop PeerPrep. Below is a brief description of the process of the workflow.

1) **Set up the team repository**
   The main team repository is set up such that every member can push their code into this repository after developing and rigorous testing.
2) **Fork the repository**
   Each member of the team will fork the repository from the team repository and locally develop the service assigned individually before making a pull request into the team repository.
3) **Review, update and merge the PRs**
   Any of the members can help to review the PR, provide any comments for improvement. If there isn't any, approve and merge the PR into the team repository. All members have to make sure to sync their local repository after every PR.
4) **Dealing with conflicts**
   If there are any conflicts during merging of PR, the individual that created the PR will have to resolve the conflict locally first before making the merge into the team repository.

Production Process



PeerPrep is using the Standard Cluster Architecture of Google Cloud Platform (GCP). PeerPrep makes use of Google domains, one for frontend and the other for gateway. We are making use of GCP's services namely:

1) Container Registry (To store Docker images)

2) Google Kubernetes Engine [GKE] (After creating Docker containers from the images, this service helps to place them into GKE pods and provide a static IP to allow access to these pods. A GKE ingress controller is configured to act as a load balancer to establish connection between an external IP to the internal static IP of the pods.)
3) Network services (Utilizing Kubectl and Google Managed Certificate object, a SSL certificate can be provisioned to the 2 ingress controllers (frontend & gateway) to establish secure HTTPS connection.
4) IAM and Admin services (A service account that can be used in Github Actions continuous deployment)

Setting up Github Actions CD onto GCP

1) Checkout to Github repository
2) Setup GCloud CLI and login with service account created with IAM (https://acloudguru.com/hands-on-labs/creating-and-manage-iam-roles-on-google-cloud)
3) Configure Docker to use GCloud CLI
4) Get our project credentials
5) Build Docker image
6) Push the image to Container Registry
7) Set up Kustomize
8) Update the GKE secrets, which the value is obtained from Github Actions Secrets
9) With Kustomize, edit the image specification (of those specified in Deployment.yaml) to have a tag similar to the Github commit SHA
10) Output the latest update of the K8s file to the standard input and use this standard input to kubectl apply
11) Check the deployment status

Current Usage

- 8 Docker containers (Since we are using cloud DB in production, there is no need to start up the Mongo and Redis containers in production, compared to during development)
- 1 GKE Autopilot Cluster (GKE will perform auto provision, configuration and management of the nodes and infrastructure in the cluster)
- 8 Workload/GKE Pods (Deployment.yaml applied here)
- 8 Services (Services.yaml applied here, 6 ClusterIp and 2 NodePort service type)
- 2 Ingress (Ingress.yaml applied here. Only frontend and gateway)
- 7 Secrets

Product Backlog

User service - Desmond

S/N	Requirement	Wk 7	Wk 8	Wk 9	Wk 10	Wk 11	Wk 12
FR1.5	The system should allow users to delete their account.		■				
FR1.6	The system should allow users to change their password.		■				

Matching Service - Xin Yi

S/N	Requirement	Wk 7	Wk 8	Wk 9	Wk 10	Wk 11	Wk 12
FR2.1	The system should match two different users who selected the same difficulty level.				■		

Question Service - Jiet

S/N	Requirement	Wk 7	Wk 8	Wk 9	Wk 10	Wk 11	Wk 12
FR3.1	The system should provide 3 question difficulty levels that users can choose from, i.e., Easy, Medium, Hard.	■	■				
FR3.2	The system should display information about the question, i.e., title, description, input, output, examples.		■	■	■		
FR3.4	The system should display question categories that users can choose from, i.e., Data Structure, Dynamic Programming, Sorting Algorithms, etc.		■	■	■		

Collaboration Service - Li Xue

S/N	Requirement	Wk 7	Wk 8	Wk 9	Wk 10	Wk 11	Wk 12
FR4.1	The system should join the two matched users into the same room.	■	■				
FR4.2	The system should allow the two matched users to edit code in the same code editor.	■	■				
FR4.3	The system should send users back to the question difficulty			■	■		

	selection page when his/her match has disconnected.			▨	▨		
FR4.4	The system should let the user know when his/her match has disconnected.			▨	▨		
FR4.5	The system should ensure that the user leaves the room after 10 seconds since his/her match disconnects.			▨	▨		
FR4.6	The system should sync the code when the user refreshes the page.					▨	

Communication Service - Desmond

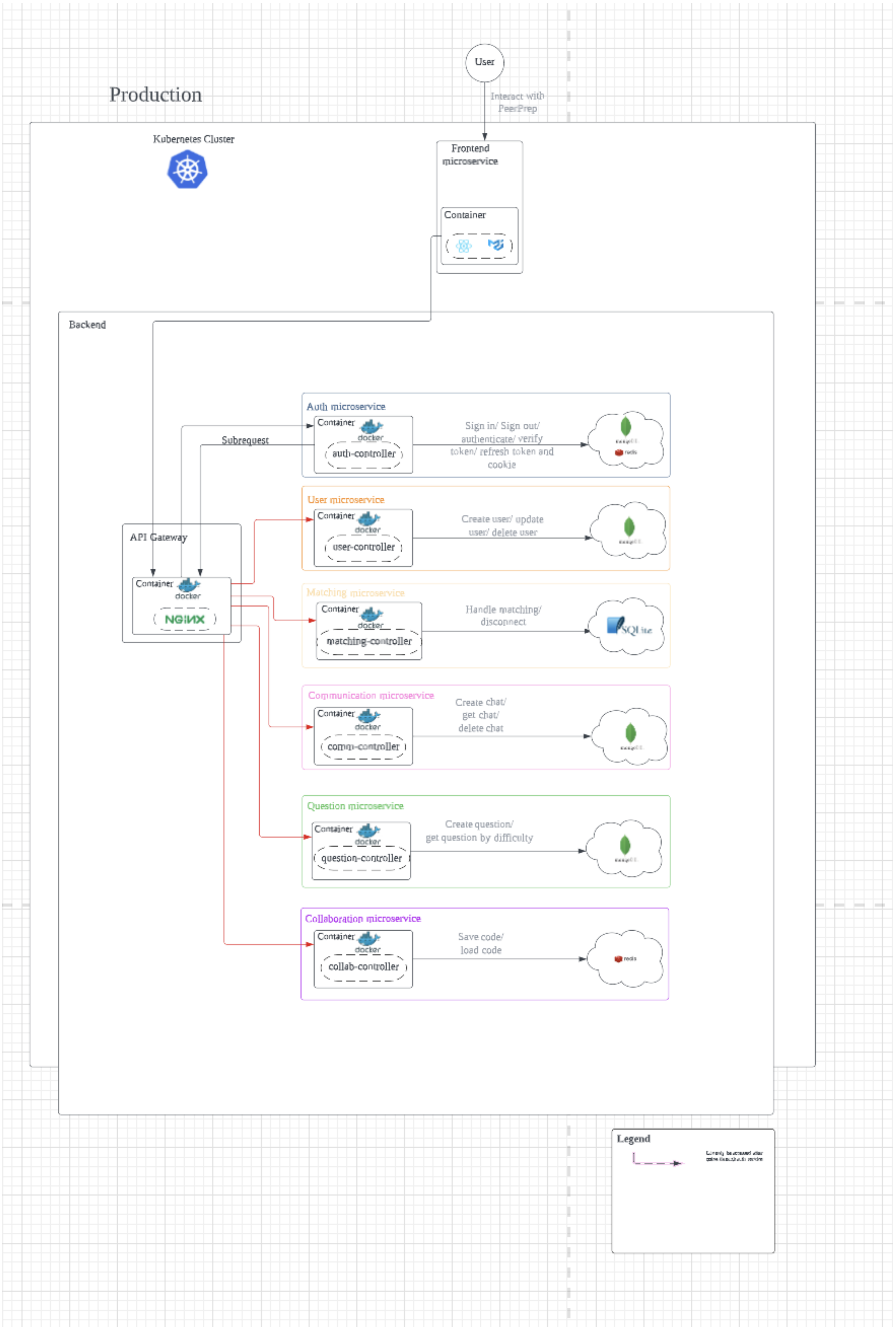
S/N	Requirement	Wk7	Wk8	Wk9	Wk10	Wk11	Wk12
FR5.1	The system should allow the two matched users to chat with each other.	▨					
FR5.5	The system should sync the chat messages upon refreshing the page.				▨		
FR5.6	The system should delete all the chat messages of a destroyed room from the database.				▨		
FR5.7	The system should force kick everyone out of a room within 5 seconds when one of the users leaves.				▨		
FR5.8	The system should store all the chat messages with respect to the room both users joined.				▨		
FR5.9	The system should allow the two matched users to communicate with each other using audio calls.					▨	

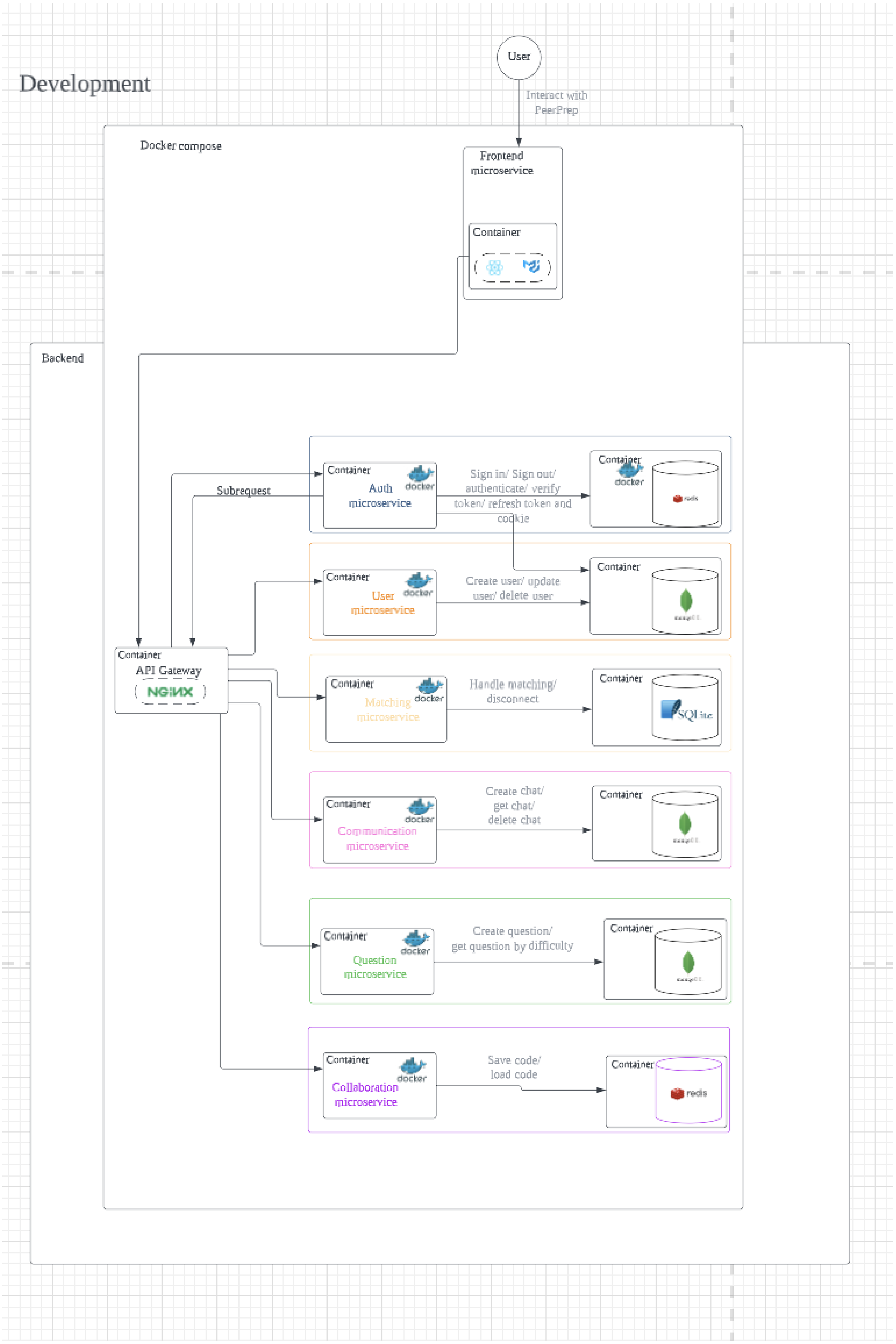
Deployment - Desmond, Xin Yi

S/N	Requirement	Wk7	Wk8	Wk9	Wk10	Wk11	Wk12
FR7.2	The system should be deployable using Docker.				▨	▨	
FR7.3	The system should be deployed to GCP.					▨	▨
FR7.4	The system should have an NGINX API gateway that serves as a reverse proxy and authorization/authentication provider.					▨	▨

Design Diagrams

Architecture diagram


Production
User
Interact with PeerPrep
Kubernetes Cluster
Frontend microservice
Container
Backend
API Gateway
Container
docker
NGINX
Subrequest
Auth microservice
Container
docker
auth-controller
Sign in/ Sign out/ authenticate/ verify token/ refresh token and cookie
mongoDB
redis
User microservice
Container
docker
user-controller
Create user/ update user/ delete user
mongoDB
Matching microservice
Container
docker
matching-controller
Handle matching/ disconnect
SQLite
Communication microservice
Container
docker
comm-controller
Create chat/ get chat/ delete chat
mongoDB
Question microservice
Container
docker
question-controller
Create question/ get question by difficulty
mongoDB
Collaboration microservice
Container
docker
collab-controller
Save code/ load code
redis
Legend


Development
User
Interact with PeerPrep
Docker compose
Frontend microservice
Container
Backend
Container
Auth microservice
docker
Subrequest
Sign in/ Sign out/ authenticate/ verify token/ refresh token and cookie
Container
docker
redis
Container
User microservice
docker
Create user/ update user/ delete user
Container
mongoDB
Container
API Gateway
NGINX
Container
Matching microservice
docker
Handle matching/ disconnect
Container
SQLite
Container
Communication microservice
docker
Create chat/ get chat/ delete chat
Container
mongoDB
Container
Question microservice
docker
Create question/ get question by difficulty
Container
mongoDB
Container
Collaboration microservice
docker
Save code/ load code
Container
redis

Auth service

1) User sign in
2) User log out
3) Acknowledge validity of JWT token
4) Authenticate using JWT token

User service

1) Creation of user’s account
2) User profile update
3) Delete user’s account

Matching service

1) Handle matching logic
2) Handle disconnect logic

Communication service

1) Handle joining of room
2) Create chat
3) Handle chat
4) Read chats
5) Handle end of session

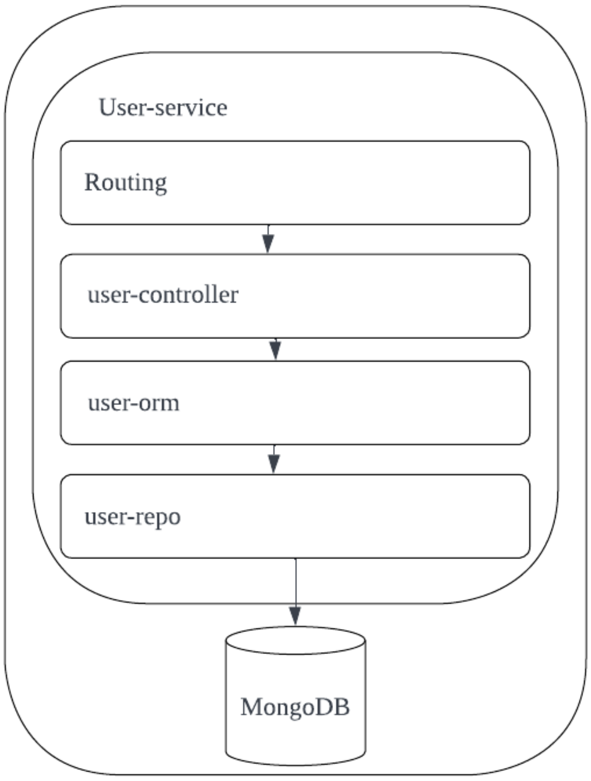
Question service

1) Get all questions
2) Creation of question
3) Get question
4) Get question by difficulty
5) Delete question

Collaboration service

1) Handle joining of room
2) Handle update of code
3) Handle end of session

Layered (similar for other microservices)



Design Decisions

Technology Stack

Component	**Technology**
Frontend	React, Material UI
Backend	Express.js
Database	MongoDB, SQLite, Redis
Deployment	Google Cloud Platform (GCP)
Pub-Sub Messaging	Socket.IO
Cloud Providers	GCP
CI/CD	GitHub Actions
Orchestration Service	Docker-Compose (development), Kubernetes (production)

Project Management Tools	GitHub Issues

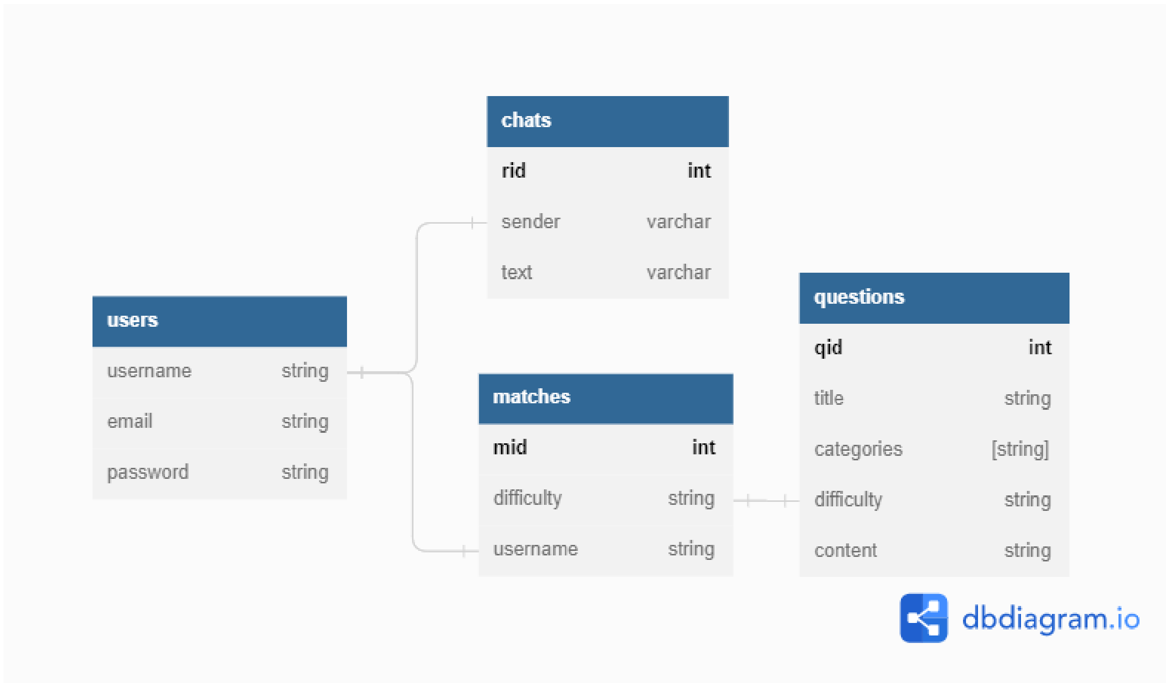
Frontend
We made use of React + Material UI to do up the frontend of PeerPrep. React allows the team to create reusable components that can be used in the different pages of PeerPrep. Material UI allows us to quickly realise certain UI design of PeerPrep without needing to code the UI from scratch, thus saving the team time to focus on the other microservices.

Backend
MongoDB:

User-service - A structured user schema is created to store the information of the users that sign up. Since we presume a wide scale usage of PeerPrep in the future, using MongoDB helps in allowing us to scale out as more and more users sign up to make use of our app.

Question-service - A structured question schema is created to store the information of the interview questions that can be retrieved to be displayed in the Room page. Similarly, using MongoDB to store questions allows us to scale the database eventually to store more questions (NFR4.2).

Communication-service - A structured chat schema is created to store the information of the chat that can be retrieved to be displayed in the chat box in the Room page. Since there is a need to make sure the chat messages are persistent to prevent the issue of accidental reloading of the Room page, which would cause all the chat messages to disappear, MongoDB is used to store the chat messages (FR5.5). We did consider using Redis for fast and temporary storage of chats, but it seems to us that there might be issues regarding storing the chats as a key-value pair since chats need to keep track of many other information such as the roomId.



Redis:

Auth-service - We decided to use Redis to store blacklisted JWTs. The users' JWT is added to the database whenever they sign out or delete their account (FR1.3). While we also clear the browser cookie which contains the JWT, this further prevents users from manually supplying their previous JWT in the cookie to access the services as the JWTs are checked against the stored blacklisted tokens. Since we need to authenticate users whenever they access different services and navigate across pages, the database would be accessed frequently. As such, a fast accessing time is desired (NFR1.1) and led us to choose Redis.

Collaboration-service - Since we want to make sure that the code in the code editor remains persistent in the event of a refresh of the room page (FR4.6), we thought of using Redis to store the code for slightly longer than the period the two peers are in the room. We can easily store this information by setting the roomId as the key and the written code in the code editor as the value of the key-value pair stored in Redis. Since Redis is an in-memory database, using Redis also allows the code to be retrieved and loaded to the users quickly if they refreshed their page, allowing them to continue coding with minimal waiting time.

SQLite:

Matching-service - We have considered MongoDB, SQLite and Redis to store the match requests.

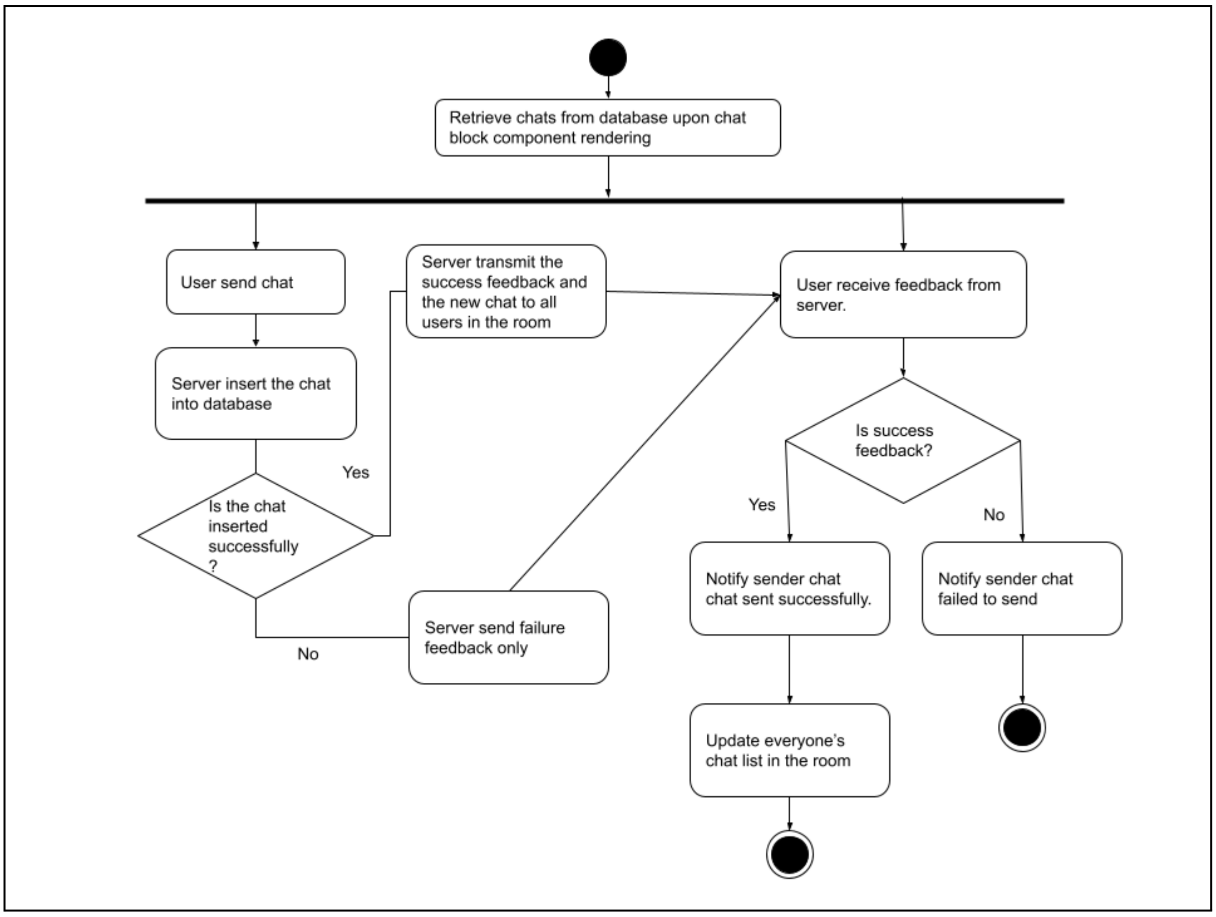
Comparing MongoDB and SQLite, both are persistent where MongoDB stores data in actual databases while SQLite stores data in files. Since we value fast matching (FR2.2), SQLite would be a better choice as file retrievals are faster than database retrievals. Although MongoDB is able to store more data, SQLite is sufficient as the match requests are temporary, i,e, requests are created, stored for 30s, and deleted thereafter.

Next, comparing SQLite and Redis, data retrieval is fast since SQLite stores data in files while Redis stores data in memory. However, since Redis is a simple string key-value store, Redis is inefficient to work with with our matching logic, i.e., same difficulty level and different users. SQLite, on the other hand, provides more straightforward conditional queries.

Design considerations

Communication service

Chat logic


Retrieve chats from database upon chat block component rendering
User send chat
Server insert the chat into database
Is the chat inserted successfully ?
Yes
No
Server transmit the success feedback and the new chat to all users in the room
Server send failure feedback only
User receive feedback from server.
Is success feedback?
Yes
No
Notify sender chat chat sent successfully.
Notify sender chat failed to send
Update everyone's chat list in the room

RoomId
At the start, it was decided to obtain the roomId from the URL so that both users in the same room will be able to chat in the same session. However, this might introduce some vulnerabilities of other users being able to access the room just by supplying the room id in the URL in their own browser. We understand this issue and will take it into consideration in future iterations by introducing JWT token for authentication.

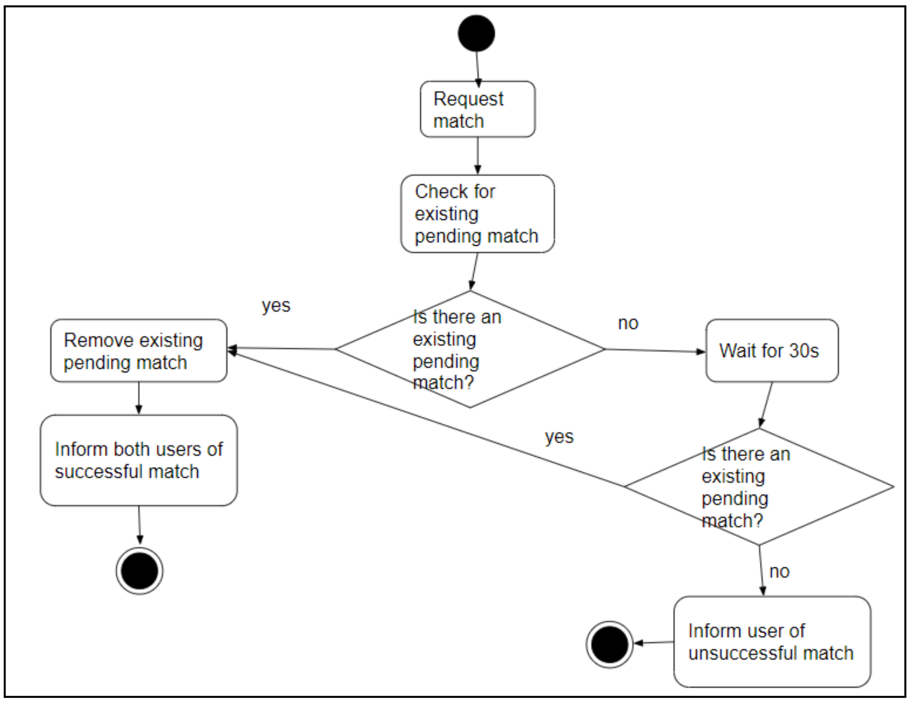
Scrolling behaviour
Previously, the chat will automatically scroll to the latest messages only when the user sends a message. The thought process for this is to avoid the unwanted and annoying behaviour of scrolling to the bottom every time the peer sends a message since the user might be busy coding. Given another scenario where the user is still reading previous messages above sent by his/her peer, if the scrolling works such that it will scroll to the latest message every time a new message is sent (be it by the user or his/her peer), the user might miss seeing previous messages on top. Taking this into consideration, we disabled automatic scrolling to allow users to continue reading previous messages. A possible future improvement is to implement a chat system similar to Telegram where a badge on the side displays the number of unread messages. This UI will allow the user to keep track of any new messages when reading previous messages. Then, introducing a notification sound will also help to notify the user of a new message when he/she is busy coding.

Sending chat messages

Previously, we assumed that users would just hit 'Enter' to send their messages. However, we feel that this might not be as intuitive in terms of how to send the messages. It might not seem natural to users since there are no signs or indications for them to send their input. As such, we decided to render a "Send" button after users type in any input in the chat's text box. This makes it more intuitive and allows users to send messages by clicking the button or hitting 'Enter'.

Matching service

Backend Matching Logic


Request match
Check for existing pending match
yes
Is there an existing pending match?
no
Remove existing pending match
Wait for 30s
Inform both users of successful match
yes
Is there an existing pending match?
no
Inform user of unsuccessful match

Synchronisation between frontend and backend matching

It is very likely that there will be synchronisation issues between frontend and backend matching due to the different internal system timers and communication delays. To enhance usability (NFR2.1), we implemented a buffer logic in the frontend.

When the backend emits a matching failure event before the frontend timer counts down to zero, the frontend buffers the event and only notify users of the matching failure once the timer ends. On the other hand, when the frontend timer reaches zero before the backend emits the matching failure event, the frontend will display a loading prompt, i.e., "Loading…", until the backend emits either a matching success or failure event.

Session behaviour

A session is handled by a room created with Socket.IO from matching-service. A room will have at most two users. Users will be kicked out from the room and the session will be ended for the following scenarios:

1. At least one user clicks and confirms on finishing the session
    a. This is required based on the documentation
2. At least one user closes the tab/browser
    a. Since one user disconnected and is unable to return to the same room, we consider the interview session has ended, thus removing the other user as well

3. A user keeps on refreshing the page
    a. The session should be closed since this is abnormal behaviour and too many refreshes add additional cost to the server

Question service

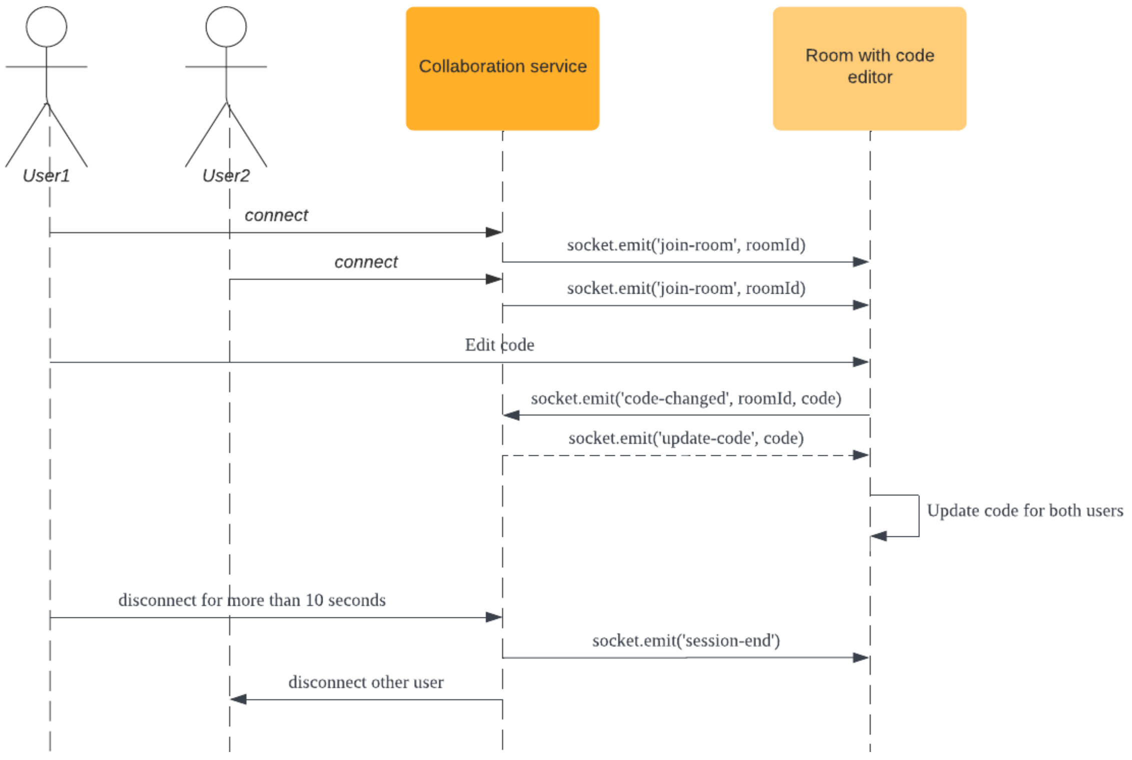
Display question behaviour

The 2 peers that selected the same difficulty for their question will be navigated to the same room (same roomId) and be shown the same coding interview question of that selected difficulty. We also have to keep in mind that the questions shown every time the 2 peers match, even if they are the same peers that matched previously, have to be randomly selected from the pool of questions of the selected difficulty.
Hence, we came up with the idea to encode the roomId of the 2 peers which converts the string to an integer, which is used as the ‘random’ number of skips in the pool of questions based on the selected difficulty.
Eg. If the roomId for peer 1 and peer 2 is encoded to produce 4 and the pool of questions for the selected difficulty is as such: [question 0, question 1, question 2], the app will retrieve question number (4 % 3 = 1) from the pool of questions. Thus, displaying question 1 for both of the peers.

Collaboration service

Collaborative code editing logic

When two users are matched into a room, their socket (Socket.IO) will join the room to subscribe and publish events happening in the room (FR4.1). Whenever a user edits his/her code in the editor, a ‘code-changed’ event is published to the collaboration service server, which in turns publishes a ‘update-code’ event to the other user in the room. Upon receiving the ‘update-code’ event, the second user’s code editor will be updated with the latest code (FR4.2, NFR1.3).
If a user disconnects or leaves the session, a ‘session-end’ event will be published after 10 seconds. This is to prevent early termination of a session in case the user only refreshed his/her page and not actually leaving the session. Upon receiving the ‘session-end’ event, sockets in the room will disconnect and users will be sent back to the home page (FR4.5).


Collaboration service
Room with code editor
User1
User2
connect
socket.emit('join-room', roomId)
connect
socket.emit('join-room', roomId)
Edit code
socket.emit('code-changed', roomId, code)
socket.emit('update-code', code)
Update code for both users
disconnect for more than 10 seconds
socket.emit('session-end')
disconnect other user

<u>Code editor behaviour</u>

We decided to use CodeMirror as it is a lightweight editor which offers functionalities such as syntax highlighting, displaying line numbers, and setting the appearance of the editor with provided themes. We displayed the line numbers so that it resembles popular code editors that users are familiar with, and also allows peers to refer to their code easily when communicating. For the appearance of the code editor, we chose a theme which displays a white background, with texts that are mainly green or blue in colour depending on the syntax. This is consistent and matches the overall theme of our application. Currently, the code editor highlights the code as JavaScript. A possible improvement in the future is to allow users to select their preferred programming language.

Design Principles & Patterns

Principles

Single Responsibility Principle

In PeerPrep, it is ensured that every class only focuses on one task and therefore only has one responsibility in the application. We made sure to break up the parts of the UI and group them as individual components. Thus, there exists many individual components such as AlertMessage, DifficultyButtons…

A microservices architecture will ensure that each service is loosely coupled such that there is low to no dependency linked between each service. Thus, this principle allows us to work independently in the team to develop each service without the need to worry about having complicated dependencies between each service.

Open-Closed Principle

By employing a microservices architecture, we ensure that the application is open to extension but prevents unnecessary modification of the other services that are not related to new services we develop in the future.

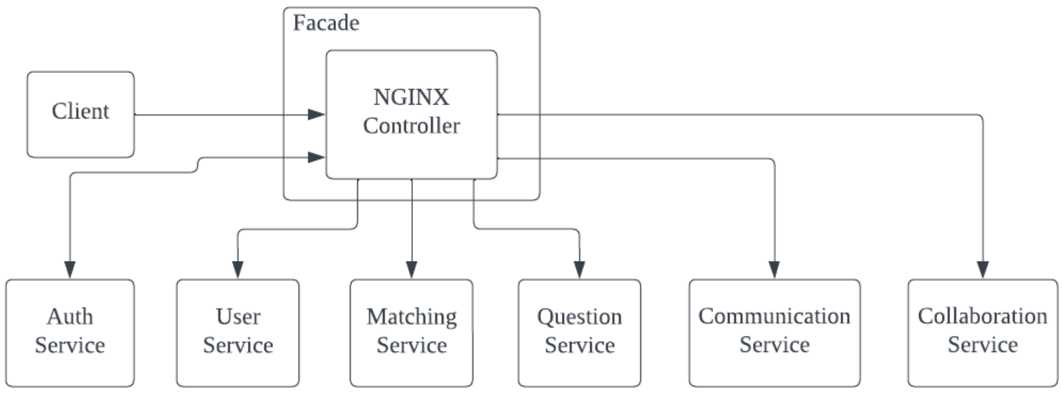
Dependency Injection Principle

There is a lot of emphasis on this principle in PeerPrep as each service will separate the model and repository of the service from the controller that calls the operations that are performed on the model objects. Doing so abstracts out the complexities of performing the model operations from the controller. The controller can just call the methods using the correct interface. Suppose there is a need to change the database provider in the future, we can just modify the repository without changing the controller, as long as the repository follows the same interface for method calls.

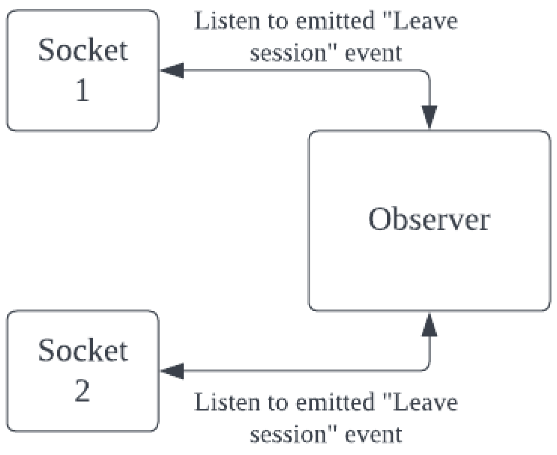
Patterns

Facade pattern

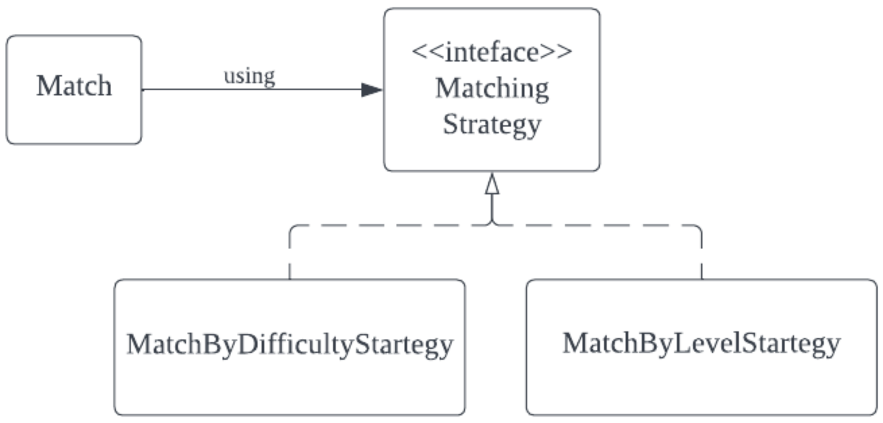
In PeerPrep, we made use of a facade pattern to allow users to access the necessary functionalities through the NGINX controller, while hiding the complexities away from the users.



Observer pattern
In PeerPrep, the observer pattern is used in the sense that a connected socket will wait for when the other corresponding socket emits a ‘leave session’ event, which will cause the other socket to leave the connection (room).


Socket 1
Listen to emitted "Leave session" event
Observer
Socket 2
Listen to emitted "Leave session" event

Strategy pattern
In future iterations of PeerPrep, we are thinking of including additional ways to match the different users. An example can be by users’ level [Users can gain levels by completing different coding questions after being matched with another user]. Thus, we can employ the strategy pattern to create different implementations of the Matching service.



6) Improvements & Enhancements

Deploy to Google Cloud Platform(GCP) using Github Actions

Implementation (Create YAML file for each services):

- Add “id-token” with the intended permissions.
- Authenticate via credentials json
- Setup and run Google Cloud CLI
- Configure docker to use Google Cloud CLI tool as credential helper for authentication
- Obtain GKE credentials to deploy to cluster
- Build Docker image with Github commit SHA and Github ref of branch or tag
- Push Docker image to Google Container Registry
- Setup up Kustomize that is used to customize Kubernetes objects through a kustomization file
- Delete and recreate Kubernetes Secrets to store confidential data such as the Redis cloud host, port and most importantly, password.
- Deploy the Docker image to GKE cluster

Future enhancements

- We understand that though we have put a lot of thought into preventing unauthorised access to user profile data through the frontend, there are some vulnerabilities that still exist through the backend. For example, it is ensured that users will not be able to manipulate other user’s data except their own when updating their profile. However, we realised that skillful hackers are able to make an update on any other user’s data after they sign in by sending a PUT request to the backend through Postman. Therefore, we do note the need to include protection protocols at the backend in future iterations to prevent these malicious operations.
- Due to the limitations of CodeMirror, when one user updates his/her code, the changes will be reflected correctly in the other user’s code editor. However, the other user’s cursor will be set to the beginning of the editor. However, this behaviour should not be too disruptive in the context of our application since the “interviewer” would not be coding as much as the “interviewee” in most scenarios. The “interviewer” would usually provide some hints and type in the code editor when the “interviewee” is stuck, thus the behaviour of resetting the cursor position of the other user would be minimal under normal circumstances. Nevertheless, this could be improved in future iterations by finding workarounds or using other libraries that support tracking cursor positions.

7) Reflections & Learning Points

The biggest takeaway from developing PeerPrep would be the implementation of microservices architecture. We realised how easy it would allow the division of different functionalities of an application such that different teams of developers can focus on developing different functions without the need to converge together and discuss multiple times.
Comparing to what each of us experienced in CS2103/T, where there was a meeting every week, for this application, we did not have to meet up as much, as long as everyone understands what they have to do for each service. Evidently, this architecture allowed us to develop and deploy much faster than for CS2103/T (much more complex features).
Additionally, microservices separate faults in the sense that if an error occurs in user-service, it will not affect the running of match-service. Furthermore, having this attribute allows us, developers, to easily pinpoint the issue in the service and quickly generate a fix to rectify the problem.
Ultimately, this chosen architecture is viewed as favourable because of it allowing us to swiftly develop independently, test and deploy the viable product.