CS3219 Software Engineering Principles and Patterns

AY2022/23 Semester 1

Group 57: Final Report

Team member	Matric No.
Chew Hoa Shen	A0200044E
Jim Vincent Andes Engay	A0206031B
Terence Ho Wei Yang	A0196511E
Wong Wen Wei Esmanda	A0206353M

Project Information

Github Repository URL: https://github.com/CS3219-AY2223S1/cs3219-project-ay2223s1-g57

Contributions

Name	Technical Contributions	Non-Technical Contributions
Chew Hoa Shen	Full-stack ● Communication Service Backend development ● Matching Service	Project report
Jim Vincent Andes Engay	Full-stack ● User Service Deployment ● Dockerising microservices ● Creation of Ingress & Services ● Deployment on Kubernetes + HPA	Project report
Terence Ho Wei Yang	Backend ● Question Service Frontend ● Lobby & Codepad Navigation ● Collaboration Service	Project report
Wong Wen Wei Esmanda	UI/UX design Frontend ● Collaboration Service ● Lobby + Matching Service API ● Codepad + Question Service API	Project report

Content Page

1. Introduction

1.1. Problem Background

As students prepare to kickstart their software engineering career, the biggest hurdle to overcome is often the technical interviews. These technical interviews require students to be able to think on the spot and code live in front of the interviewer, at the same time clearly explaining their thought process at every point. Such interviews prove to be highly stressful and practice is imperative to help students ace these interviews.

Currently, platforms such as Leetcode are available for students to hone their problem-solving skills. However, these platforms do not allow students to be able to practise these questions in an environment that mimics the interview format of having another person watch you as you code. In addition, grinding Leetcode questions alone can be tiring and tedious.

1.2. Purpose of the Application

PeerPrep aims to provide aspiring software engineers with a platform where they can hone their live coding with another peer, through a collaborative code editor. By allowing students to practise with their peers, PeerPrep better mirrors the interview environment and helps students to be more comfortable with the pressure of coding in front of the interviewer. Being able to work on the questions with their peers can also serve to provide more enjoyment in the preparation process, as they can help to motivate one another.

1.3. Project Scope

PeerPrep mainly features a collaborative code editor to support collaborative learning as well as a chat feature, to facilitate ease of discussion between the matched peers. It includes a subset of 50 questions from Leetcode for students to practise with, categorised into three difficulty levels. The application allows users to be randomly matched with another user according to the difficulty level chosen.

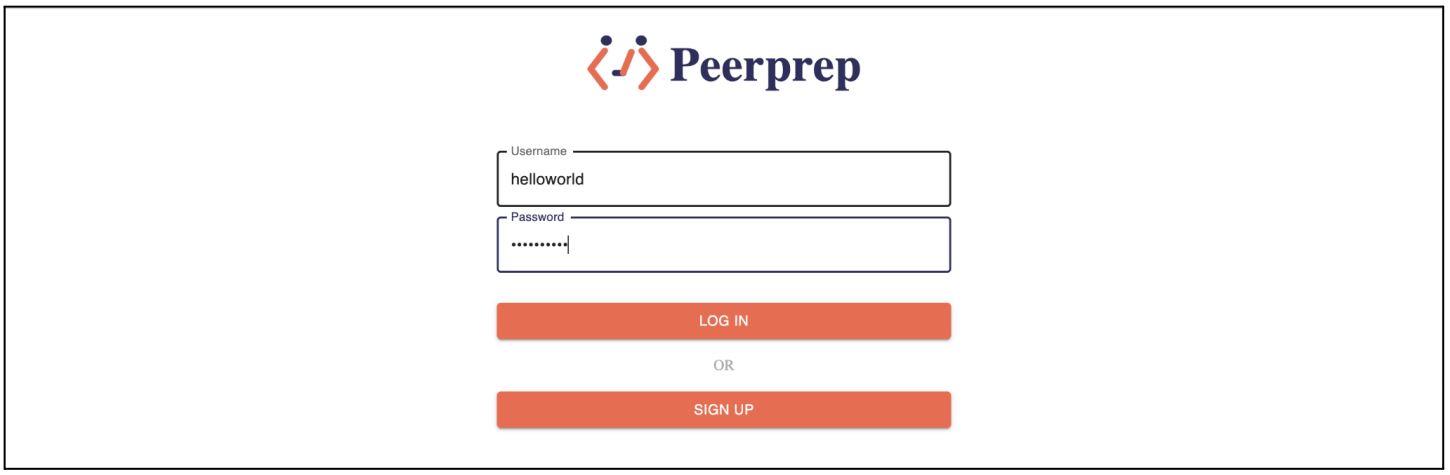
2. Functional Requirements

2.1 User Service

S/N	Functional Requirement	Priority	Design Element
1.1	The system should allow users to create an account with username and password.	High	Sign up page of frontend & user-service backend
1.2	The system should ensure that every account created has a unique username.	High	Checks in user-service backend
1.3	The system should allow users to log into their accounts by entering their username and password.	High	Login page of frontend & user-service backend
1.4	The system should allow users to log out of their account.	High	Settings page of frontend & user-service backend
1.5	The system should allow users to delete their account.	High	Settings page of frontend & user-service backend
1.6	The system should allow users to change their password.	High	Settings page of frontend & user-service backend
1.7	The system should allow users to view their account settings.	Medium	Settings page of frontend & user-service backend

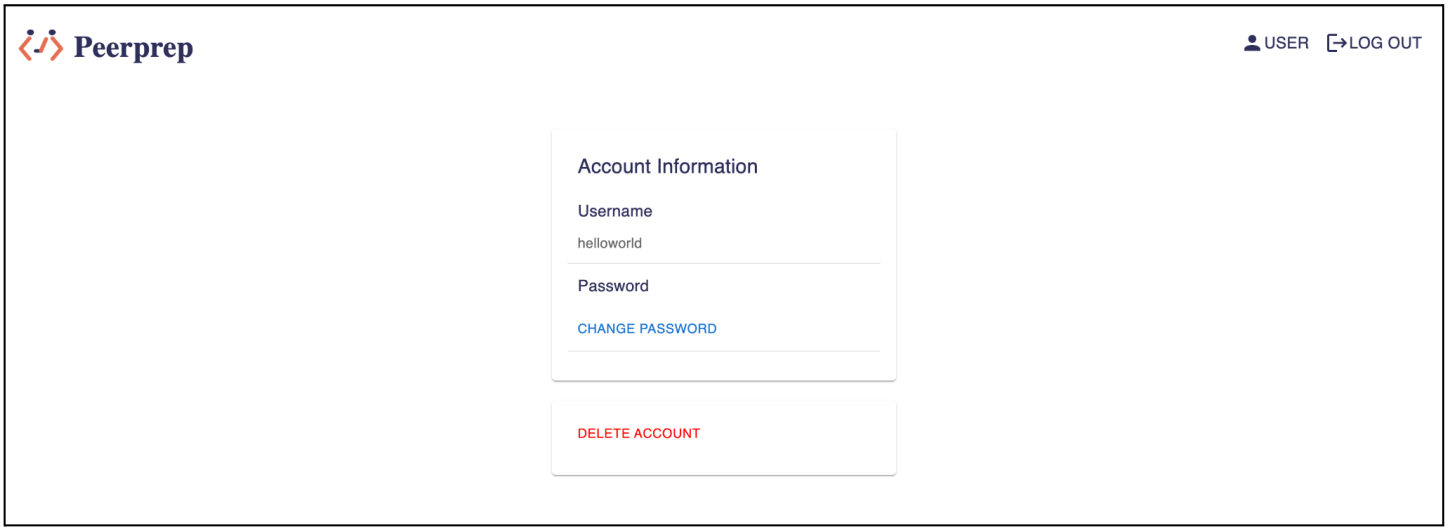
Sign-up page

Log-in page



Settings page

The user can change their password and delete their account on this page. The logout button is available at the header of the application.



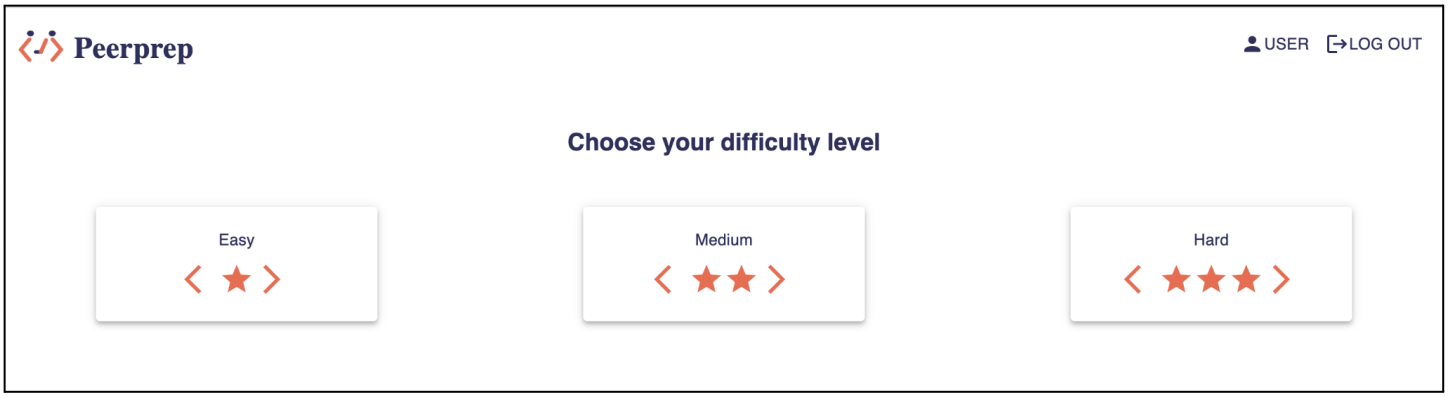
2.2 Matching Service

S/N	Functional Requirement	Priority	Design Element
2.1	The system should allow users to select the difficulty level of the questions they wish to attempt.	High	Frontend home page
2.2	The system should be able to match two waiting users with similar difficulty levels and put them in the same room.	High	Frontend lobby page & matching-service backend
2.3	The system should let the user wait for up to 30 seconds for a potential match	High	Matching-service backend
2.4	The system should inform the users that no match is available if a match cannot be found within 30 seconds.	High	Error message in frontend lobby page after timer runs out

2.5	The system should provide a means for the user to leave a room once matched.	Medium	Codepad frontend & matching-service backend
2.6	The system should not allow users to match with themselves.	Medium	Matching-service backend
2.7	The system should not allow users to be in more than one matched room concurrently.	Medium	Matching-service backend
2.8	The system should automatically disconnect users if their match is disconnected or leave the room.	Medium	Codepad frontend & matching-service backend

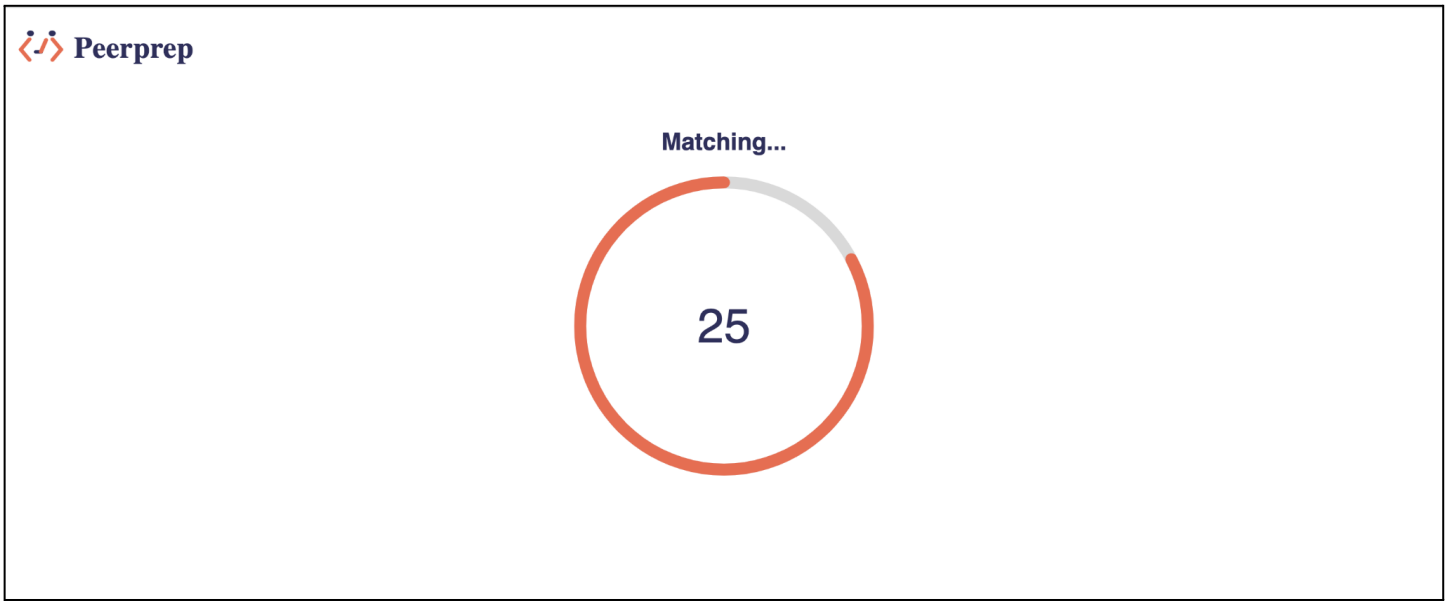
Home page

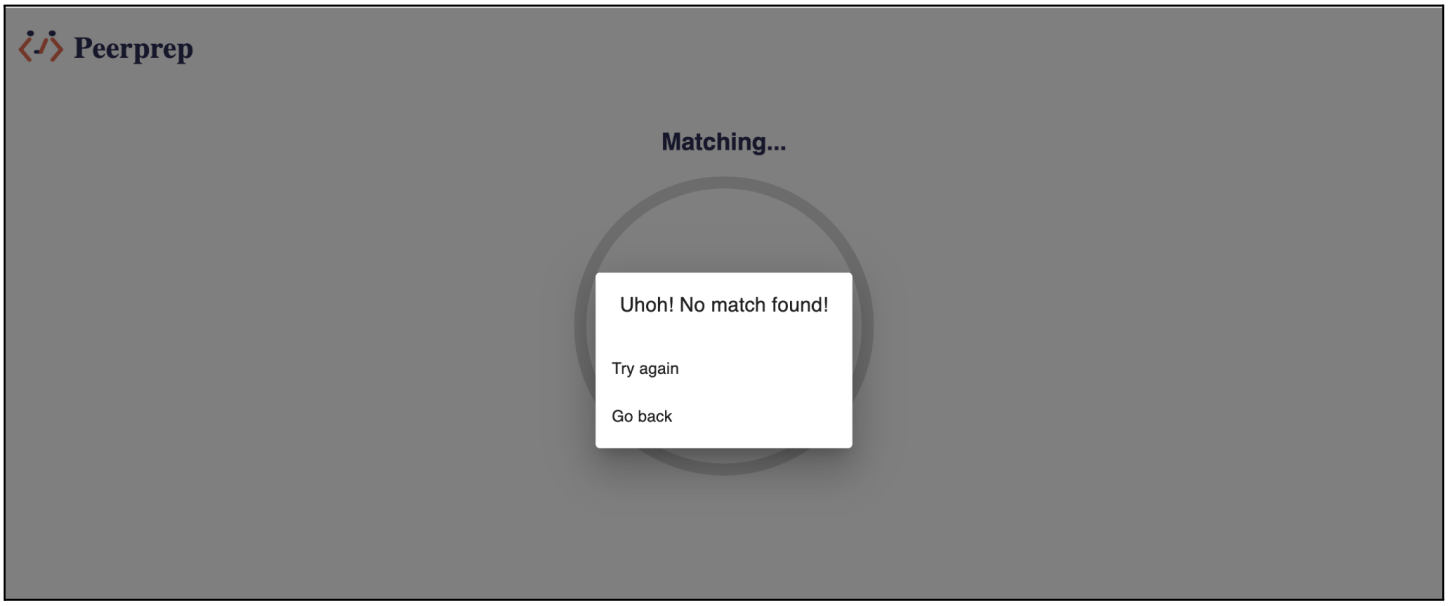
The user is able to select the difficulty level that they wish to practise. Selecting one of the three will redirect users to the lobby page for matching. The frontend connects to the matching-service to match two users with the same difficulty level.



Lobby page

The users will try to match other users for 30 seconds. If no match is found, a prompt is displayed. The user can try to match again or go back to the home page. If a match is found, the user is redirected to the codepad. Users cannot match with themselves.



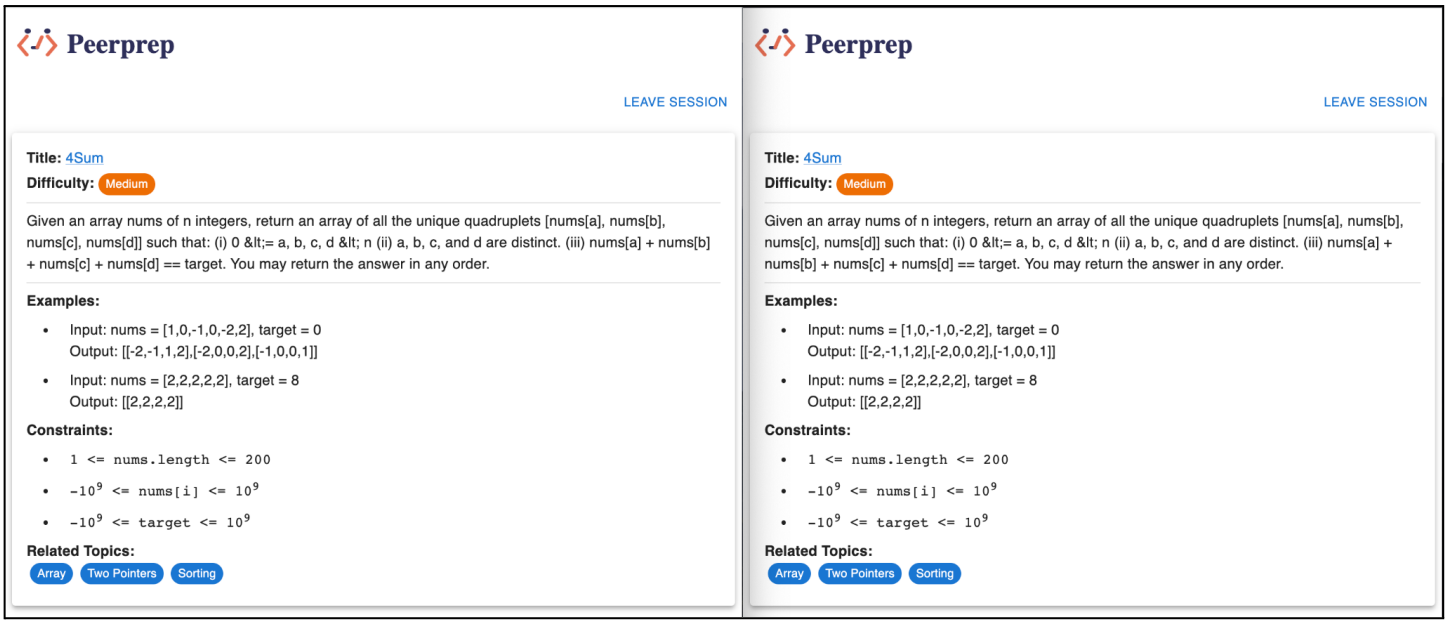


2.3 Question Service

S/N	Functional Requirement	Priority	Design Element
3.1	The system should store a bank of technical interview questions.	High	Backend database storage
3.2	The system should allow users to query for a question of a specified difficulty.	High	Backend server GET request
3.3	The system should use database indexing on the bank of questions stored in the database.	High	Backend database schema
3.4	The system should display the question of difficulty as specified by the user in the Codepad page.	Medium	Frontend codepad page

Codepad

When two users are matched, they are placed in a Codepad page. The top of the page displays the question they will solve. The question is a leetcode-style question retrieved from the Question service.

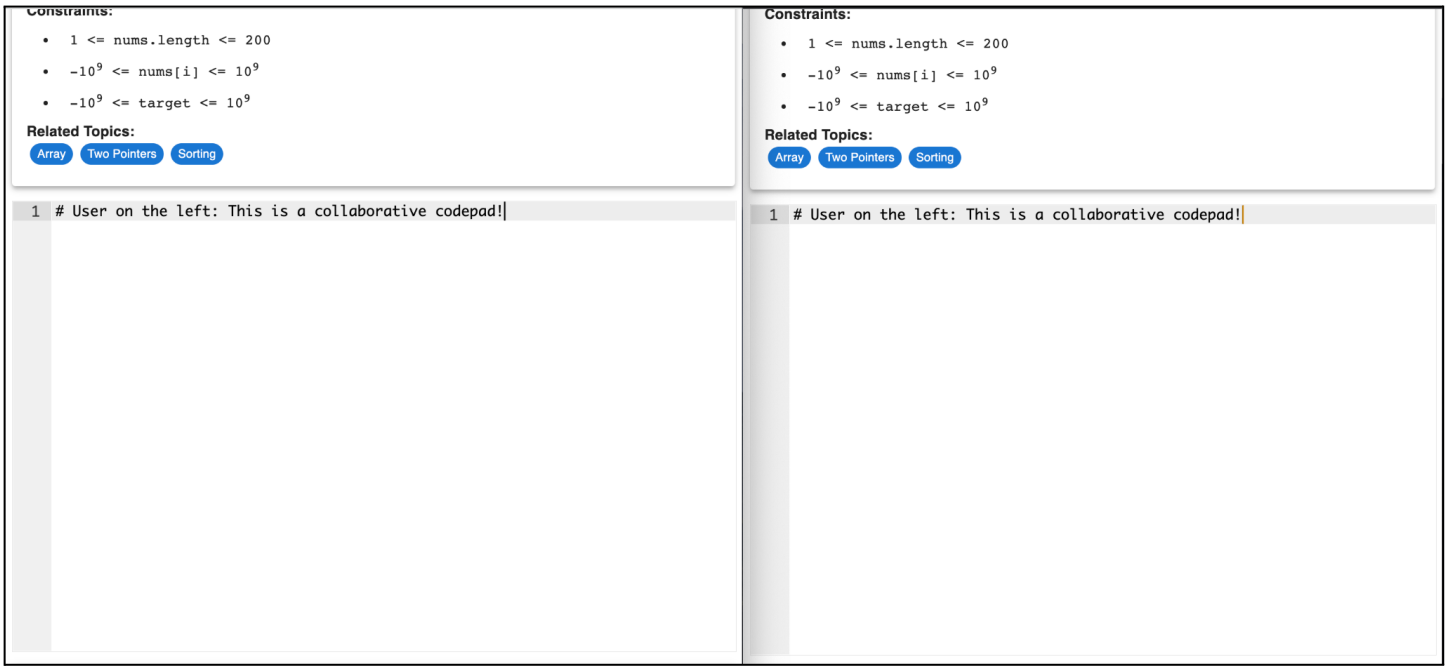


2.4 Collaboration Service

S/N	Functional Requirement	Priority	Design Element
4.1	The system should provide the users with a codepad to type their code in after matching.	Medium	Ace Editor from Firepad
4.2	The system should allow users to type on the codepad.	Medium	Ace Editor from Firepad
4.3	The system should reflect the changes made by one user to the other user.	Medium	Firebase real-time database
4.4	The system should allow users to leave the Codepad page.	Medium	Leave Session button in frontend codepad page
4.5	The system should provide some syntax highlighting for greater readability.	Medium	Ace Editor from Firepad
4.6	The system should reflect to the other user when one user highlights a piece of code in the editor.	Medium	Ace Editor from Firepad

Codepad

Below the question, we provide a collaborative codepad for the users to write their code. We are using Ace Editor from Firepad. The changes made by the user on the left are reflected on the codepad on the right. The ‘Leave Session’ button is reflected above the Question to allow users to leave the page.



The syntax highlighting makes it more convenient for users to keep track of their partner's current cursor in the coder pad

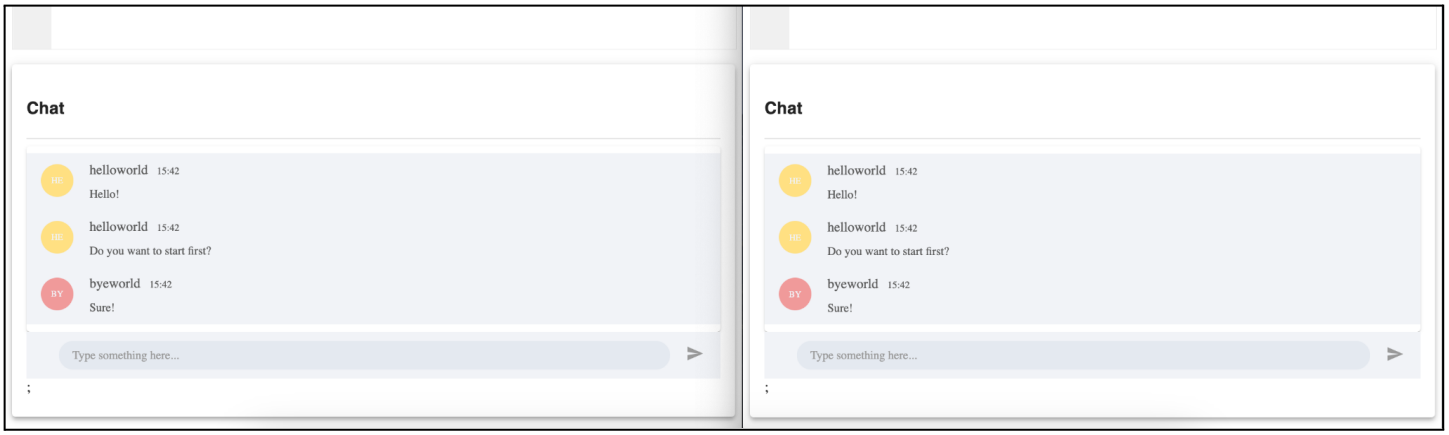
```
# Syntax Highlighting
def helloWorld():
    return "Hello World"

```

2.5 Communication Service

S/N	Functional Requirement	Priority	Design Element
5.1	The system should open a chat room for users when they enter the codepad page.	Medium	Frontend codepad page + PubNub service
5.2	The system should allow users to type and send messages in the chat room.	Medium	Frontend codepad page + PubNub component
5.3	The system should reflect the messages sent by one user to the other user.	Medium	Frontend codepad page + PubNub service
5.4	The system should close the chat room when users leave the codepad page.	Medium	Frontend codepad page + PubNub service

Chat session

Below the codepad, we added a PubNub chat box to allow users to communicate while they are preparing for the interview question. The messages sent by one user are reflected on the other user's chat box as well.


Chat
HE
helloworld 15:42
Hello!
HE
helloworld 15:42
Do you want to start first?
BY
byeworld 15:42
Sure!
Type something here...
;
Chat
HE
helloworld 15:42
Hello!
HE
helloworld 15:42
Do you want to start first?
BY
byeworld 15:42
Sure!
Type something here...
;

3. Non-Functional Requirements

The table below illustrates how we prioritised the NFRs for the application. Through our prioritisation, we deemed **scalability, performance and usability** to be the most important.

NFRs	Score	Availablility	Integrity	Performance	Scalability	Security	Usability
Availablility	1		<	^	^	^	^
Integrity	1			^	^	<	^
Performance	5				<	<	<
Scalability	4					<	<
Security	1						^
Usability	3						

4.1 Performance

Given the main functionality of our application is to promote collaboration while solving technical interview questions, having a near real-time performance would be key to a seamless collaborative experience. As such, we rank performance highly among the other non-functional requirements. This includes reducing the latency in matching as long as there is an available match, near real-time updates when both matches are coding on the code editor as well as the sending and receiving of chat messages from one match to another.

S/N	Non-Functional Requirement	Priority
1.1	Collaboration service should have less than 3 second delay between sending and receiving updates	High
1.2	Communication service should have less than 3 second delay between sending and receiving messages	High
1.3	Matching service should match available users within 3 seconds when a match is found	High
1.4	Frontend user interface should be able to navigate between pages within 3 seconds	Medium

4.2 Scalability

Since the field of computer science is greatly expanding and we see a rise in the intake of students into this field of studies every year, we can expect that this application would gain traction. In addition, considering that the application is a web application, it should be able to host a substantial amount of web traffic.

S/N	Non-Functional Requirement	Priority
2.1	System should be able to support 100 registered users	High
2.2	System should be easy to scale horizontally to support higher traffic loads	High
2.3	System should be able to support 100 concurrent collaboration sessions	High

4.3 Usability

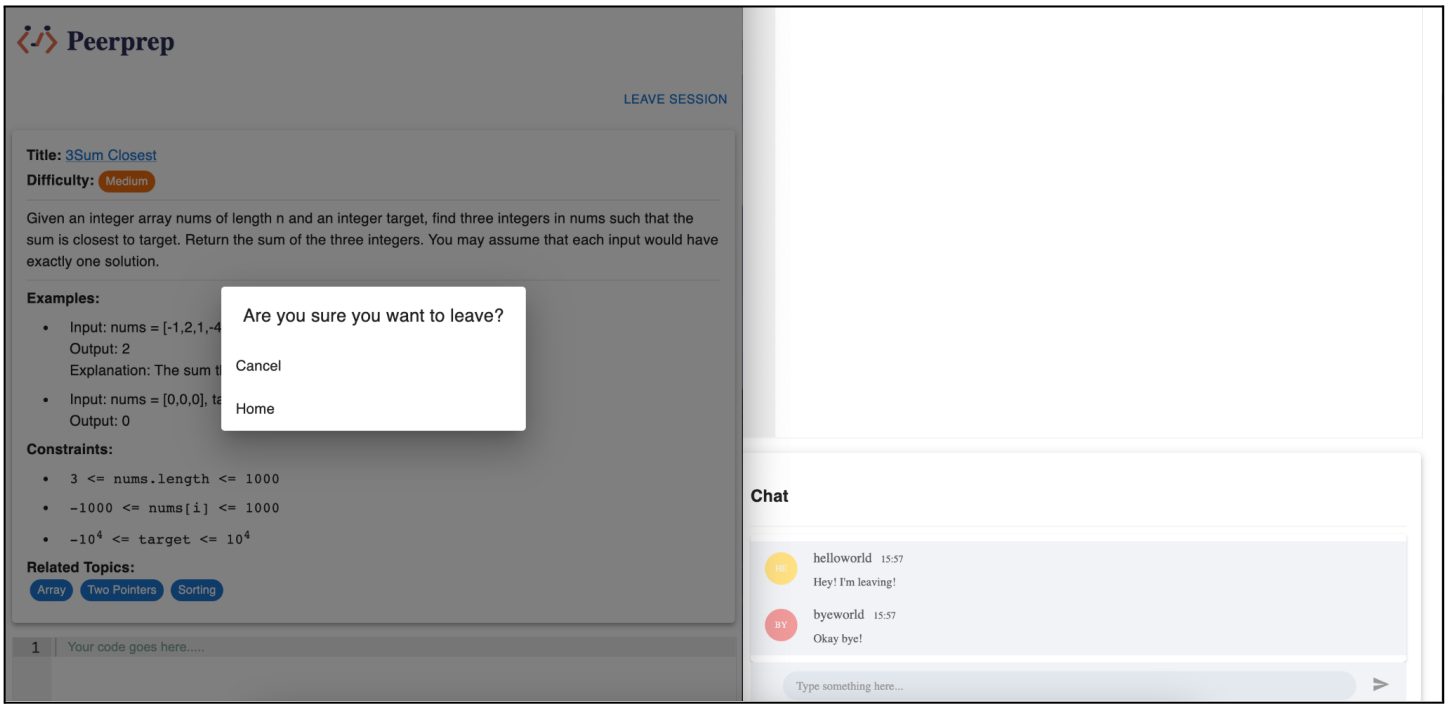
The goal of the portal is to provide users preparing for technical interviews with a hassle-free, easy-to-use solution to improve their job/internship preparation experience. We aim to allow users to quickly on-board the app, guide them towards using the application effectively and assist them when or where something went wrong. We provide error messages and confirmation dialogs to allow users to acknowledge and confirm changes in the application and redirect them to the next step where necessary.

S/N	Non-Functional Requirement	Priority
3.1	UI is clear and concise for users to navigate	High
3.2	CodePage page should be well designed, where the question, codepad and chat can be easily accessed	High
3.3	UI should inform user of any errors that has been encountered	High
3.4	UI should keep user informed on their matching progress	Medium
3.5	UI should provide user-validation when changing password	Medium
3.6	UI should prompt user for extra validation when deleting their account	Medium

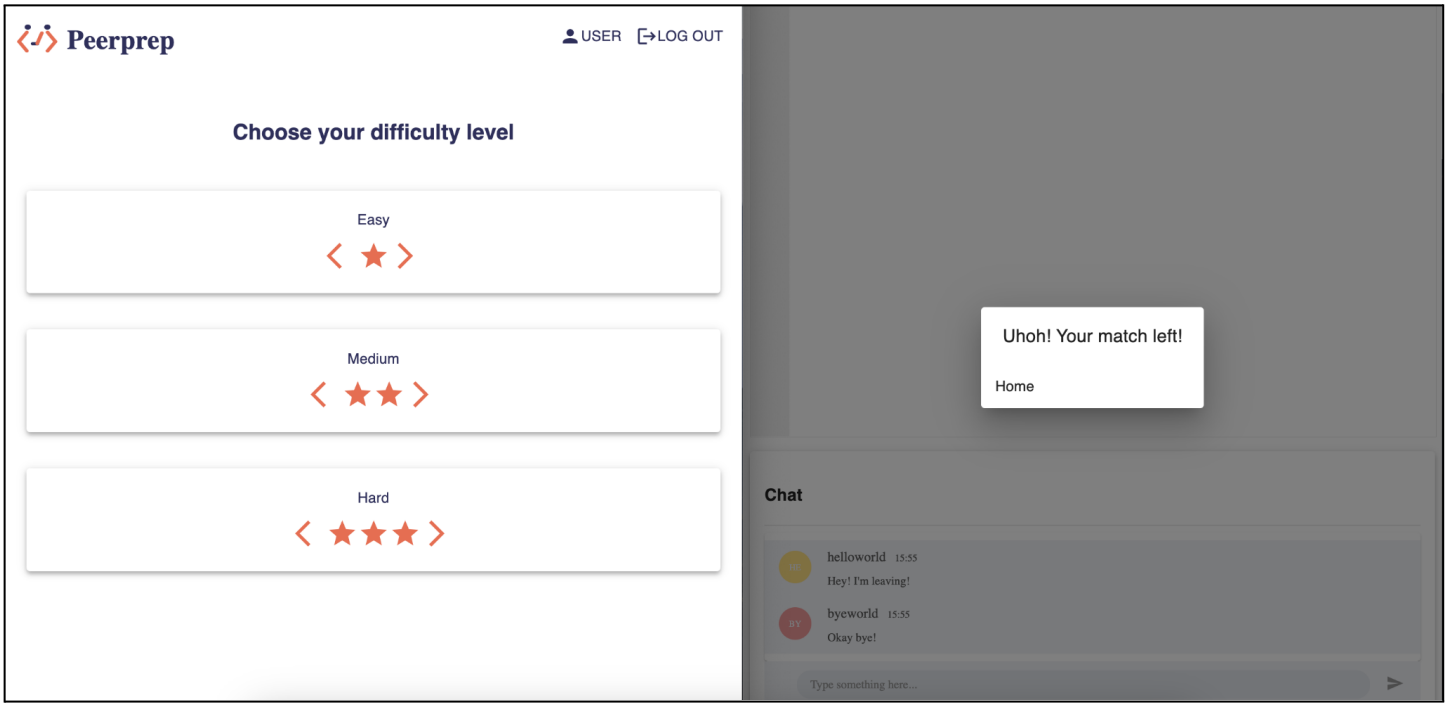
Confirmation & Error Dialogs

One of the errors that the users might encounter is when their partner leaves the codepad. We have designed the application to allow exactly two users to be in the codepad. This is in consideration of the collaborative nature of this application.

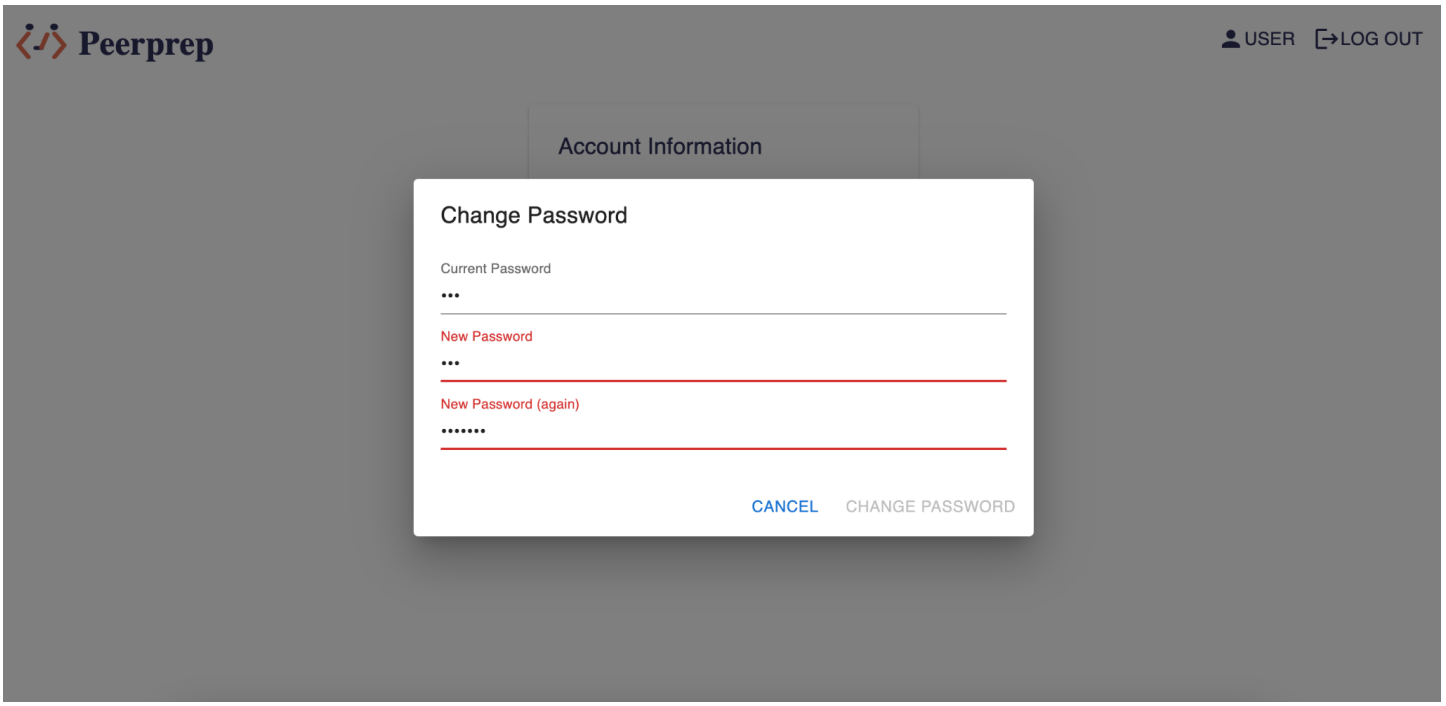
When a user leaves the codepad, it is given a confirmation dialog to acknowledge them leaving the codepad.

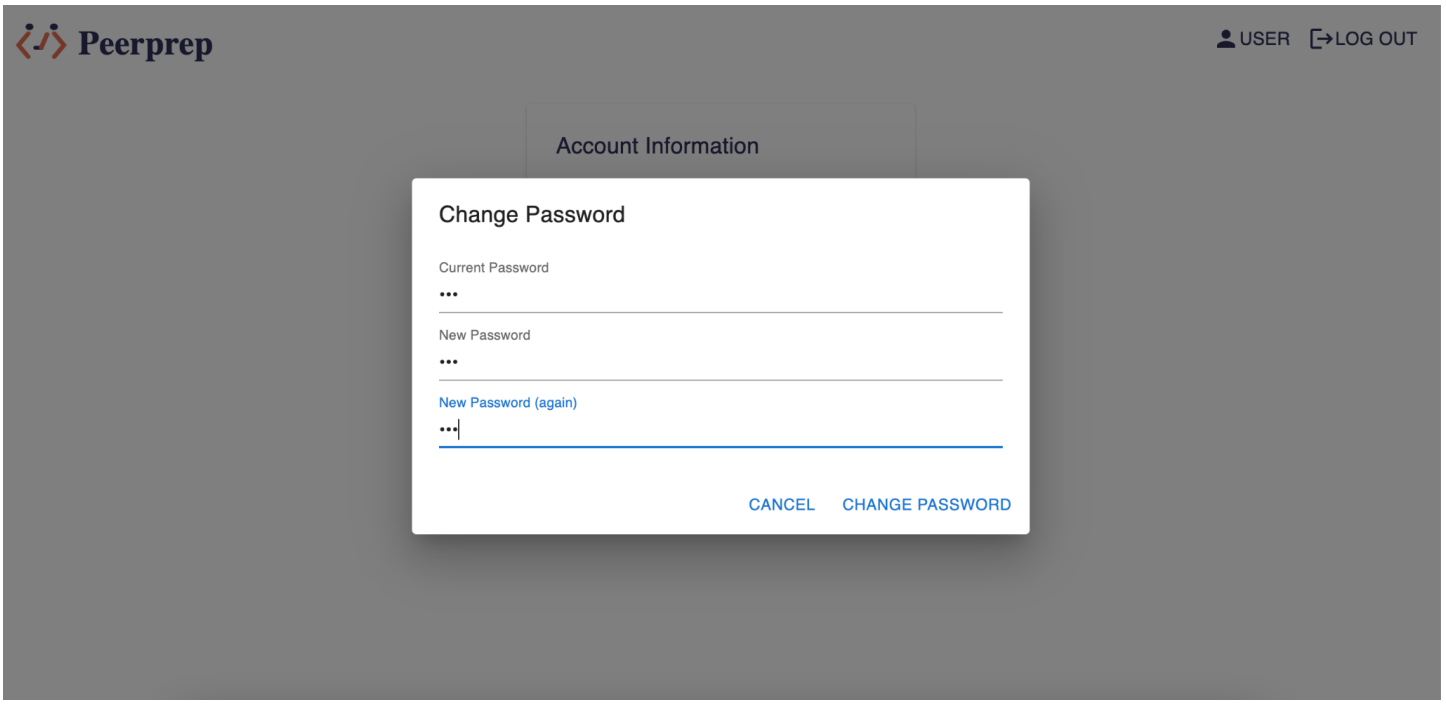


The other user is shown a dialog and a confirmation for the user to acknowledge that their partner has left the codepad. They are then gracefully redirected back to the home page.

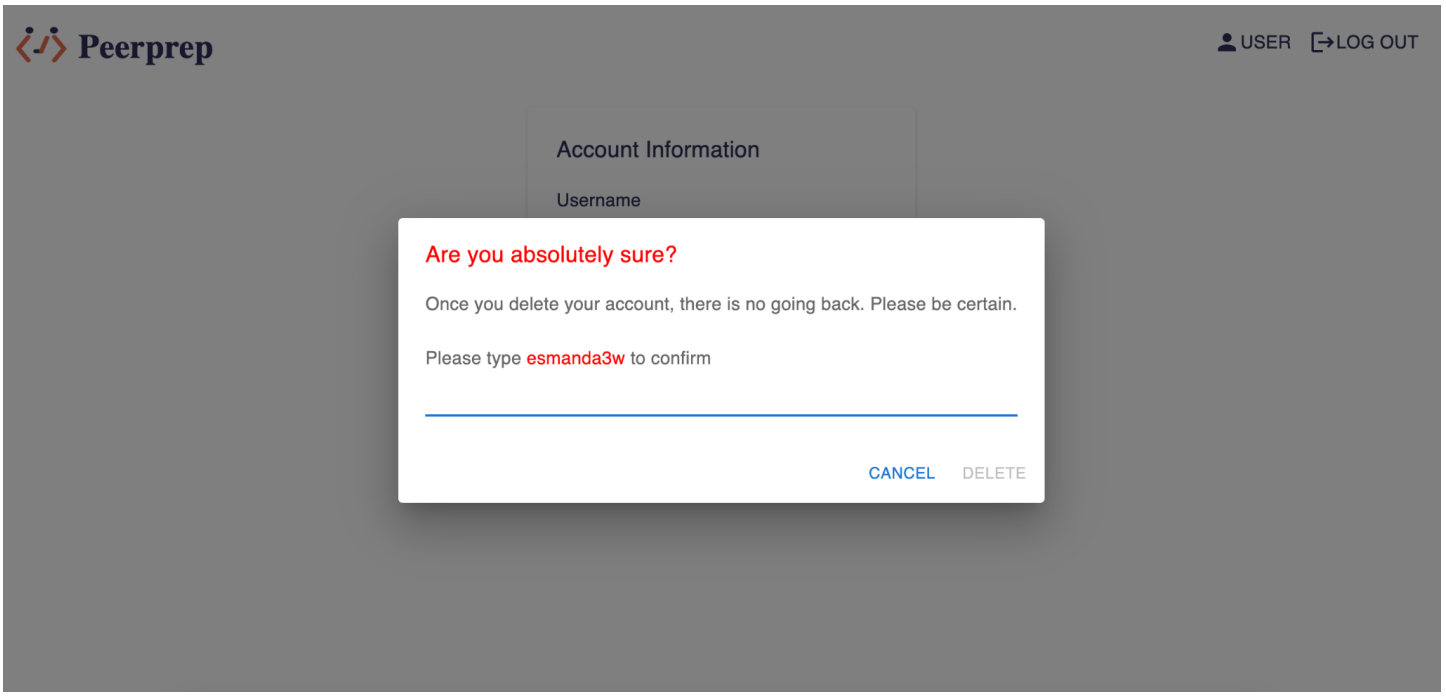


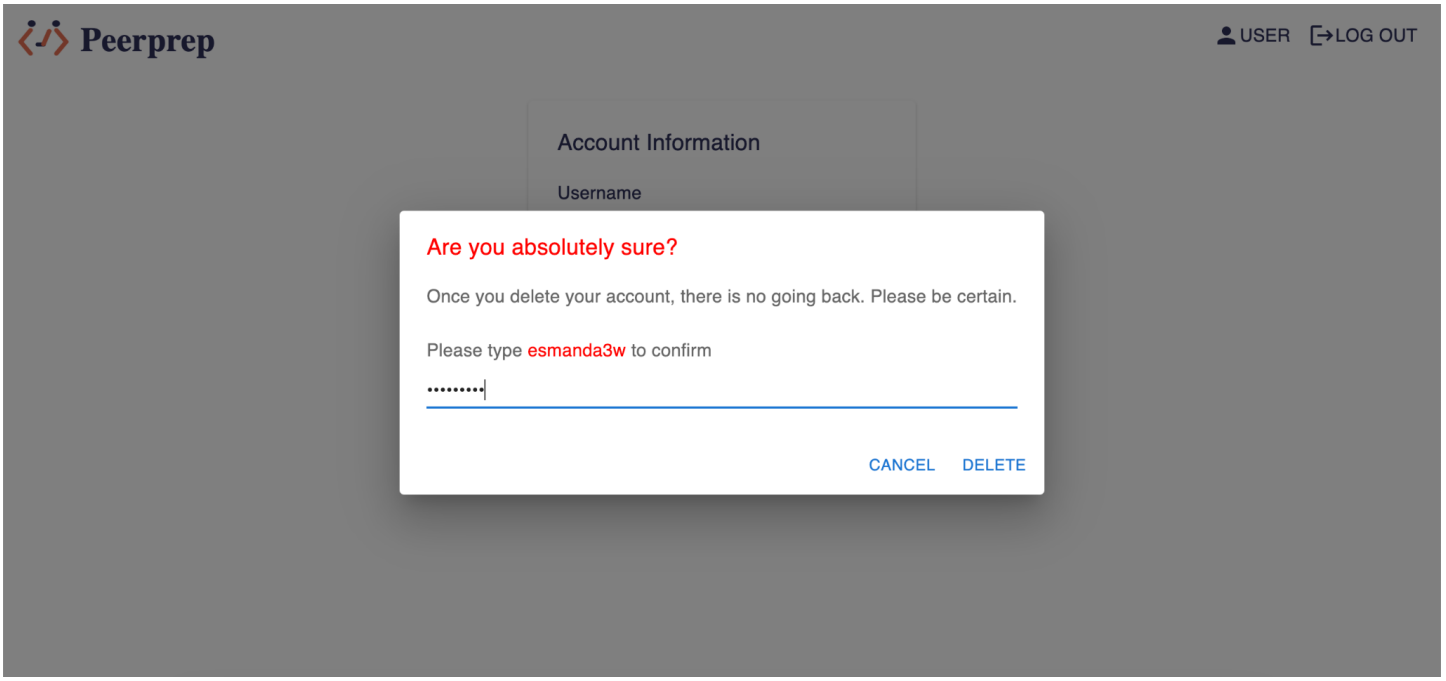
The UI provides a second input field for the new password. If both the ‘New Password’ fields are not matched, the ‘Change Password’ button will be disabled





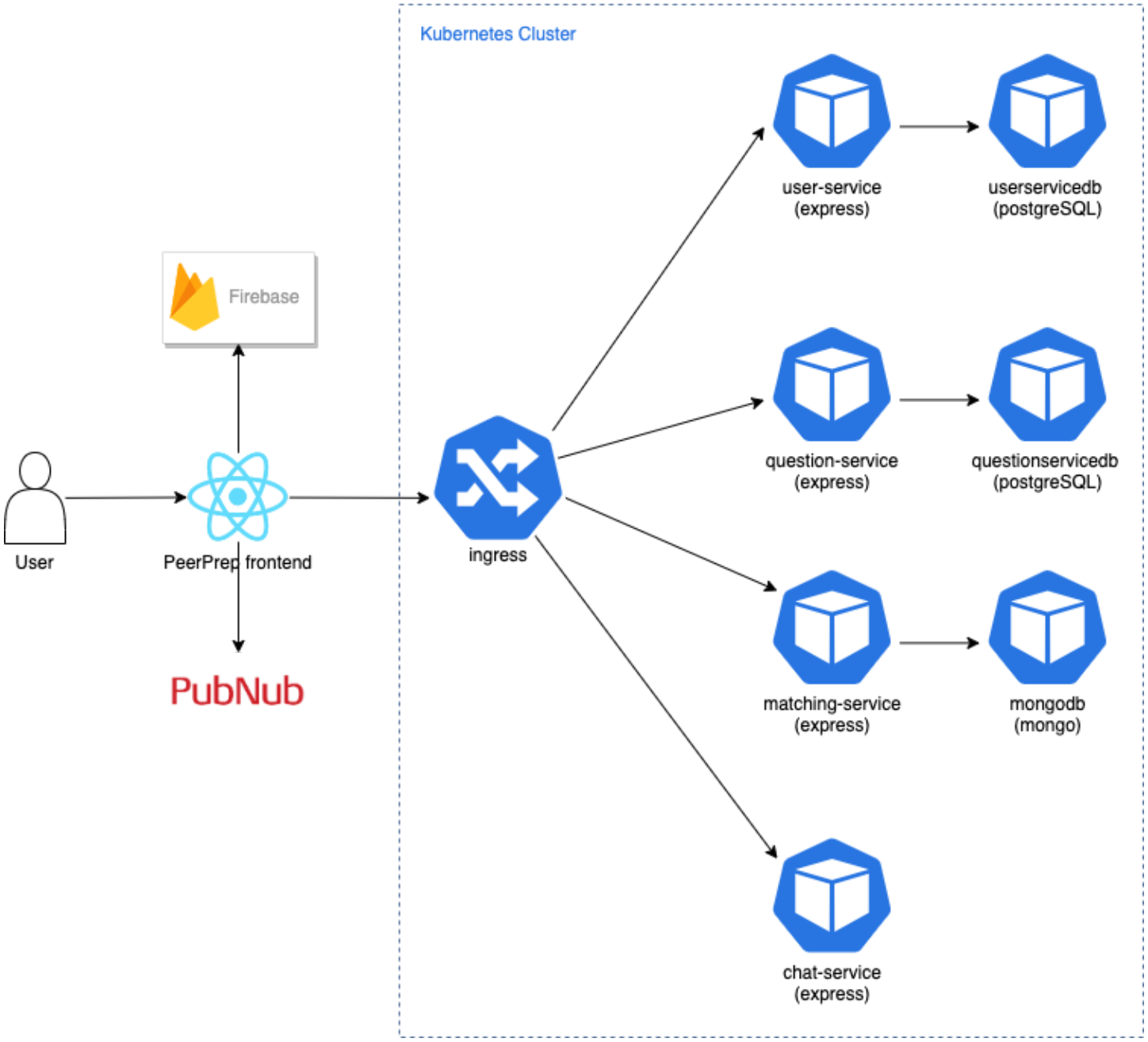
The UI ensures that account deletion validation by inputting their username. The delete button is disabled when the input field is invalidated.





4. Architecture Decisions

4.1 Application Architecture Diagram



4.2 Architecture Decisions

4.2.1 Monolith vs Microservices Architecture

Monolith	**Microservices**
- Easier and faster development due to simplicity of architecture and greater familiarity with the codebase - Ease of deployment since you only need to deploy one backend service	- Ability to develop services independently, aiding the delegation of tasks among team - Flexibility in choosing the technology stack depending on what is most suitable for each service - Smaller codebase for each service which allows for better management and greater ease of locating bugs - Less coupling between different modules

Final Decision: Microservices

As our team considers the architecture to use, we acknowledge that integrating our individual components under a monolithic approach is complex and difficult to do so. This is especially so if we work with different timelines within each sprint. One member modifying a component can easily lead to conflicts in a tightly coupled component that another member is working on. Alternatively, adopting microservices as our architecture allows us to work on each service independently, making it easier to work on individual components and delegate the tasks across the team.

Each team member can also ensure that their services are well tested in isolation before integrating together as a whole. In addition, the isolated services also aid to ensure a lower degree of coupling between different modules as we only expose APIs for inter-service communication.

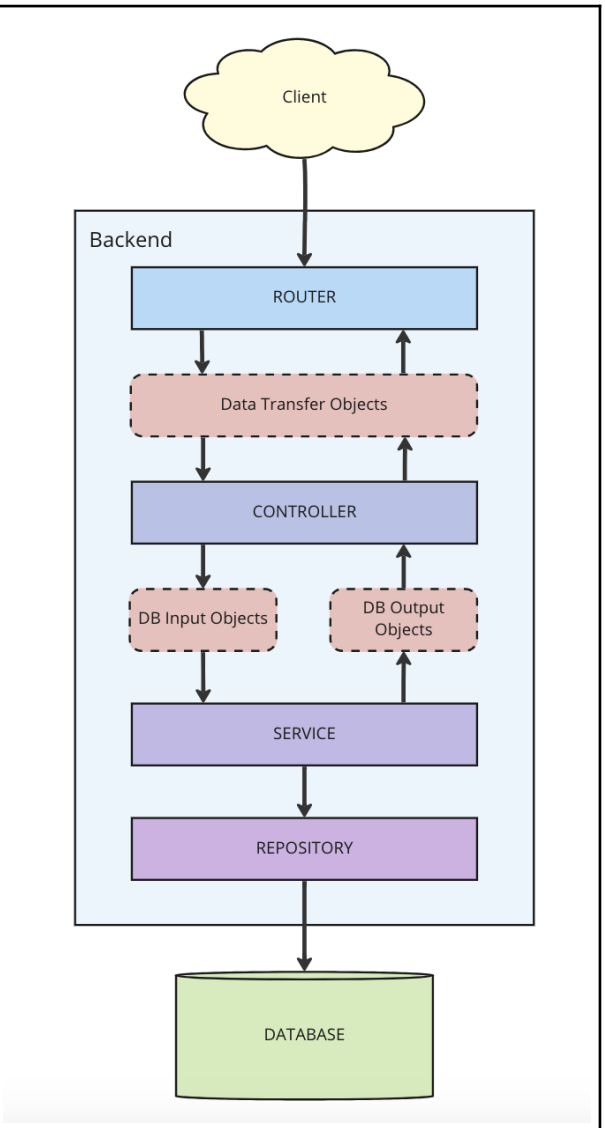
Furthermore, we had to consider the long-term development of the application. If we want to include even more features into our application (such as a video-chat service or compilation of code), we need to be able to easily add them in without causing too many changes to the existing code base. These considerations naturally lead us to select microservices as our application's architecture. A monolith would provide too many obstacles for us to extend the existing application due to the higher degree of coupling.

Lastly, we also considered other requirements such as the scalability of the system. For a web-app that might potentially have high traffic from an influx of users, we need to ensure that the application is able to scale to handle the load. By using microservices, we can easily scale up services that are in-demand which makes microservices the optimal approach to take for this project.

4.2.2 Layered Architecture for Backend Services

For services that require persistence (database storage), we employed the layered architecture for such services. The layers are explained below.

Router	Handles the routes or API endpoints of the backend service. This is where we define the individual HTTP requests and how to handle them. In general, the router extracts the information from the request (such as request.body or query parameters) and passes them forward to the controller as a form of Data Transfer Object.
Controller	The controller is responsible for performing input validation from the router and passing the Data Transfer Object down to the service layer for further computation. If necessary, the controller also maps the DTO to a representation required by the service layer. Once the computation is done, the controller also acts as an adapter to map the output from the service layer to a Data Transfer Object. The DTO is then passed to the router and sent back to the client.
Service	The service layer takes in an input for its underlying service and passes it onto the next layer. In this application, the service passes it onto the repository layer. However, the service layer can also pass the information to other third-party services or perform further computation on the data.
Repository	The repository layer is responsible for communicating queries to the database. It acts as the data access layer and maps the DTOs to a representation required by the database and vice versa.



Our team decided to go with the layered architecture to adhere to Single-Responsibility Principle as each layer has its own defined responsibility to process or validate the input data. As aforementioned, in the future, the application might be communicating with different services and repositories hence having a layered architecture would allow our microservices to swap out existing services and repositories with much greater ease. This allows us to easily extend our microservices without modifying too much of the other layers' implementation.

Furthermore, we have designed different representations of data to cater to the needs of different layers. For example, the router and controller handles what we call the Data Transfer Objects, which are representations of the data sent by the client to the server. The data is validated and converted into what we call Inputs, which are representations required by the services and repositories/data-access layer. The data-access layer and service layer generate what we call Outputs, which are then adapted and returned to the user.

By using different representations across different layers and using the controller as an adapter between the layers, we allow the API level (routers) and the lower levels (services and repositories) to have the flexibility to be modified independently. For example, we can easily swap out databases that we are using from Postgres to Firebase, and we only have to modify how the repository handles the Input from the service layer before passing it over to Firebase, instead of passing it to Postgres. Hence, there is greater separation of concerns even within the microservice.

5. Design Patterns

5.1 Adapter Pattern

Though not explicitly named an Adapter, the Controller in the user-service and question-service serves as an adapter to map Data Transfer Objects to Inputs required by service and repository layers. In addition, they map Outputs to Data Transfer Objects that are to be sent back to the client.

5.2 Facade Pattern

As we employed a microservice architecture, it would not be friendly for the client if it has to connect to multiple services in the backend. Hence, we aim to provide a unified interface for the frontend clients to connect to the backend services via a facade. This interface is provided through our NGINX ingress that forwards the requests to our backend within a Kubernetes cluster. As evident in our architecture diagram above, the client sends its requests to the NGINX ingress server. Based on the endpoint provided by the client, NGINX re-routes the request to the particular service to handle the requests.

In addition, databases and service endpoints within the Kubernetes cluster are not exposed externally and ensures that the clients are only able to access the functionalities that they need.

5.2 Publish-Subscribe Pattern

We made use of the publish-subscribe pattern for the matching service, collaboration service and communication service. Each of these services require messages to be published to specific users upon an event. Due to the nature of these services, the publish-subscribe pattern is the most suitable pattern as it is event-driven, allowing messages to be published according to the nature of the events for the different services. It also enables asynchronous parallel processing of the messages by the subscribers, promoting scalability.

6. Technology Stack

Frontend	ReactTS
Backend	ExpressJS, ExpressTS, Socket.io
Database	PostgreSQL, MongoDB
Containerisation	Docker
Orchestration	Kubernetes
Project Management	GitHub
External Tools	FirePad, PubNub

6.1 Design Decisions

6.1.1 Frontend

React
Our team chose to use React to develop our frontend after considering multiple factors.

Firstly, React is the most popular library for frontend development, with almost 200,000 stars on Github. With such strong community support, we can trust that the library will continue to be actively supported and maintained by contributors, helping to improve upon the library and ensure it stays up to date. In addition, the prolific usage of React worldwide also meant that we could be able to easily gain access to solutions whenever we meet with any difficulty.

Moreover, React allows us to separate our frontend code into different components. This aligns with the Single Responsibility Principle, where each encapsulated component handles their own state, reducing the coupling between different components. The way React helps us to structure our code lowers the complexity of the code, allowing for greater extensibility should we want to add more components.

Finally, since our team has the most experience and familiarity with React, it can flatten the learning curve and increase our efficiency in development. React also supports a wide variety of libraries for UI components that we intend to leverage on, helping us further shorten our development time.

JavaScript vs TypeScript

JavaScript	**TypeScript**
- Easier and faster development in the short run as the team is more familiar with the language	- Strongly typed language, lending itself well to greater ease of debugging - Allows developers to discover errors at compile-time making the code less error prone - Has all the features of JavaScript, and includes additional features
Final Decision: TypeScript While JavaScript would have been the easier choice of language given the greater familiarity of our team with the language, we decided to go with TypeScript as our main development language due to the benefits that it offers. Since we were working on developing the application as a team, we deemed it important to have a code base that is less error prone and easier to maintain. With TypeScript, it forces developers to write more accurate and explicit code, making collaboration easier as every team member would know the exact input and output of a function. Thus, we felt that the upfront investment of picking up the language was a worthy tradeoff.	

6.1.2 Backend

Express
Our team chose to use Express as our backend framework for our microservices.

Firstly, Express is a popular web development framework with a strong community support and we can trust that the framework will continue to be supported and maintained by contributors. Additionally, begin a popular framework, it is easy for us to find solutions online which will help us in our development process.

Secondly, Express works with Node which has many open source packages developed by the community that we can leverage for our project to increase the productivity and efficiency of development. This will cut down that time required for development and at the same time provide a more refined end product.

Lastly, Express is a framework that our teammates are familiar with. This will allow us to begin development of PeerPrep early without having to spend additional time to familiarise with our techstack.

Socket.io
Our team used Socket.io for live communication between clients and matching service.

Firstly, Socket.io provides real time event based communication between our client and matching service. This allows the client to receive live updates on the status and progress of their match search. This makes it a better option over providing the frontend with API endpoints which it would have to call to receive updates.

Secondly, Socket.io is a fully developed library which uses websockets. By using Socket.io for our backend, this removes the need for us to develop using plain websockets which has drastically sped up development.

MongoDB
MongoDB is a document based non-relational database, that our team used for database storage for the matching service.

Firstly, non-relational databases like MongoDB have faster read and write times compared to popular SQL databases like MySQL and PostgreSQL. This is especially important for our matching service, where matches are regularly being created and closed which result in read and write operations on our database. Hence, the fast read and write operations of MongoDB allow users to match other users quickly.

Secondly, MongoDB document based storage is more flexible compared to SQL databases and easier to work with. MongoDB allows us to be flexible with our database schema, which makes it easy for us to make possible future updates unlike SQL databases which require us to define a strict schema at the start.

PostgreSQL
We used a relational database to store information such as the user's account, blacklisted JWT tokens (in the user-service) and the questions (in the question-service). One of the advantages of using the relational database is due to the improved query performance via database indexing. For question service, we indexed the questions based on the difficulty of the questions. This gives us an improved database performance when we query the database for the questions of a certain difficulty. In our application, there is likely to be frequent queries to the database for the questions when users match into a codepad. Hence, using the index can speed up the query processing time and provide better performance for our application.

6.1.3 External Tools

Firepad
Our team chose to leverage on Firepad for the real-time collaborative editor. Firepad allows the easy integration of Ace editor for the code pad, which includes extensive code editing features such as syntax highlighting. Since these features are available to us out of the box, we can improve the user experience when peer programming without convoluting the code base. In addition, Firepad does not require any server code, reducing the code base. Finally, given that we have a relatively short amount of time for development, using Firepad would help us to reduce the development time needed.

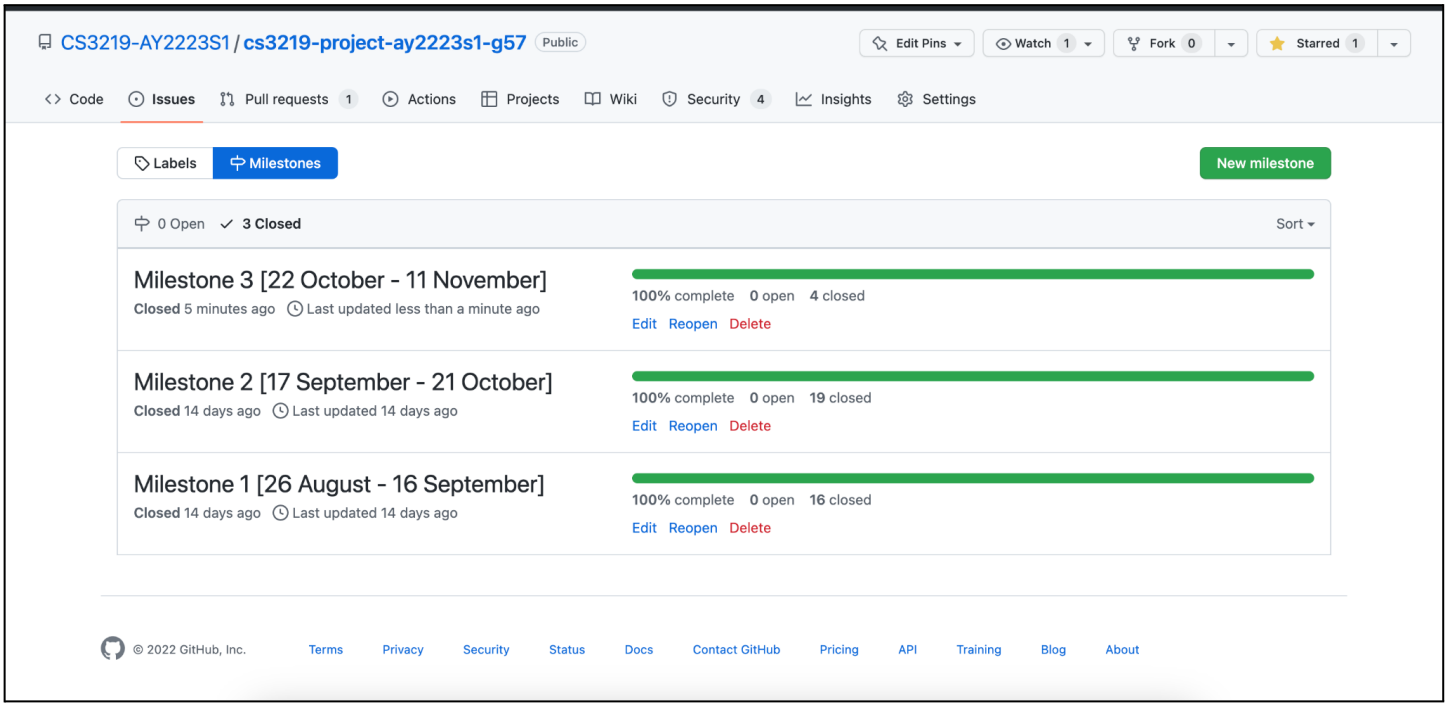
Pub Nub
Our team decided to use PubNub for the real-time communication chat between users. PubNub provides a chat box with low latency messaging system out of the box, which makes it easy for us to integrate into our PeerPrep application and speed up out development process. An alternative to PubNub that we considered

was Twilio, which would have provided the same features. However, we decided to go with PubNub due to its cleaner documentation and ease of integration.

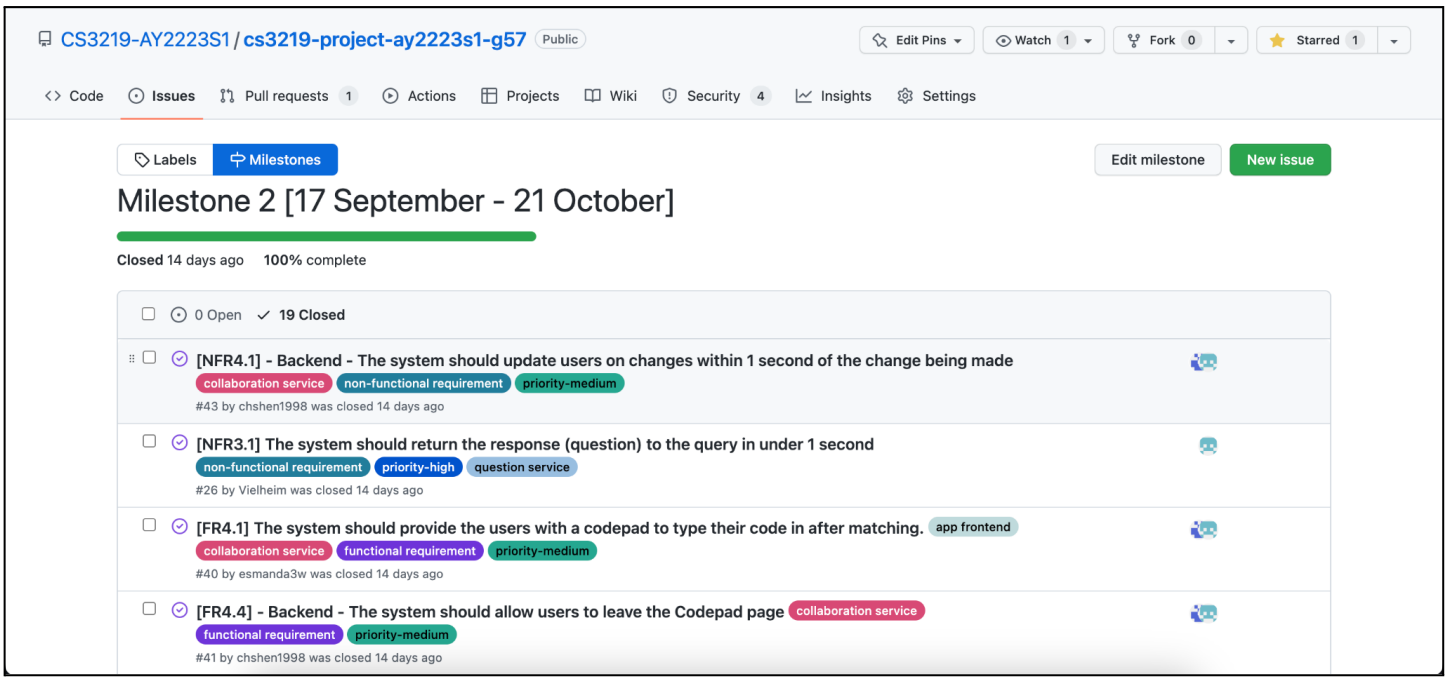
7. DevOps

7.1 Sprint Management

We made use of GitHub issues to track our tasks for each sprint, splitting them into milestones according to the project requirements.



Under each milestone, we split the tasks according to the functional requirements and features that we need to build. Each member is assigned their tasks and occasionally there might be overlapping tasks.



For the tasks within the milestone, we planned our work on a weekly sprint basis. We followed the **agile** development process. On a weekly basis, the team meets for a weekly scrum to share and discuss everyone's engineering efforts over the week. We then assessed our product backlog and evaluated the tasks that we

intend to complete for the upcoming week. Subsequently, we assigned the tasks accordingly and set our goals for the coming week.

In addition, our team would meet up once a week for a stand-up in order to resolve any blockers, communicate any concerns or collaborations between members (if necessary). Asynchronously, we would review each other's pull requests and merge them when completed. We would also use the stand-up as an opportunity to discuss any critical or urgent pull-requests. The stand-ups are also an opportunity for us to test each other's code and ensure that the components are working before merging into main. If any bugs were discovered, we would take the opportunity to resolve them as a team, or assign them to be resolved asynchronously.

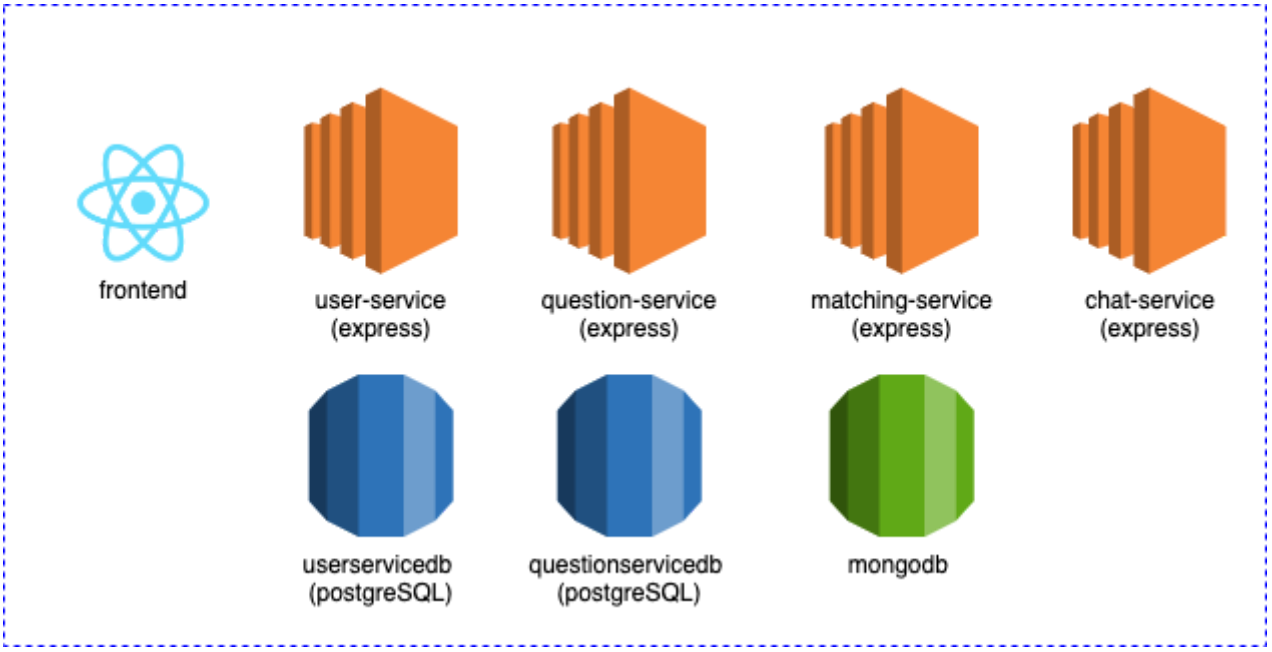
7.2 Local Deployment

Visit our project repository for instructions on how to deploy our application. We have two main methods for local deployment depending on whether we are executing development work or simulating production deployment.

7.2.1 Docker Compose

To aid with development, we containerised each service and orchestrated the deployment locally so that we can run the entire application with a single command line (**docker-compose up –build**). Moreover, we also dockerized our databases to remove additional configurations and standardise the environment for every developer.

These are the list of images created and orchestrated using docker-compose:



After running the command, all the images are created and deployed:

```
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
Starting chat-service      ... done
Starting questionservicedb ... done
Starting userservicedb     ... done
Starting mongodb           ... done
Starting question-service  ... done
Starting matching-service  ... done
Starting user-service      ... done
Starting frontend          ... done
```

We can also teardown the entire application in one place:

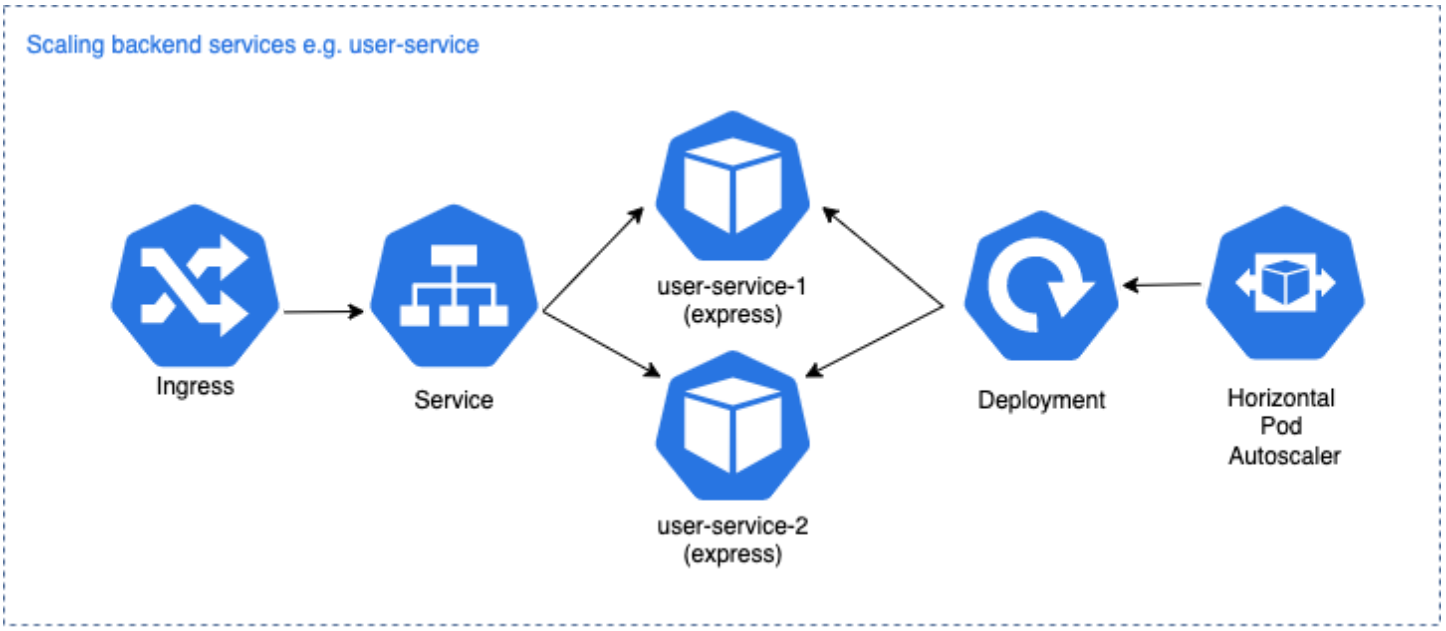
```
^CGracefully stopping... (press Ctrl+C again to force)
Killing frontend           ... done
Killing user-service       ... done
Killing userservicedb      ... done
Killing matching-service   ... done
Killing question-service   ... done
Killing chat-service       ... done
Killing mongodb            ... done
Killing questionservicedb  ... done
```

7.2.2 Docker + Kubernetes

In order to simulate the intended behaviour of the application for possible cloud deployment, we made use of Kubernetes. Kubernetes provides a framework to manage distributed systems resiliently, with service discovery, container orchestration, load balancing, and scaling.

We made full use of these kubernetes objects for our application:

1. Pods - We used pods to house our backend services containers
2. Deployments - An object that manages the instances of pods that hold a containerized application
3. Horizontal Pod Autoscaler - We used this object in order to scale up or down for our deployments depending on the current application load.
4. Service - Enables network access to a set of pods. It represents a set of pods running an application of functional component
5. Ingress - An API object that manages external access in a cluster. We used this to expose specific service endpoints within the cluster externally




```
jim@Jims-MacBook-Pro-2 k8s % k get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/chat-service-7fb5f84dc6-4ps4x       1/1     Running   0          5s
pod/chat-service-7fb5f84dc6-jtdgc       1/1     Running   0          5s
pod/chat-service-7fb5f84dc6-mcnmg       1/1     Running   0          12m
pod/matching-service-f94c795bf-rnqk8    1/1     Running   0          12m
pod/mongodb-56b8cfbfc8-szvxw            1/1     Running   0          12m
pod/question-service-54b5f98459-f96kn   1/1     Running   0          12m
pod/question-service-54b5f98459-gkn8s   1/1     Running   0          7s
pod/questionservicedb-6f4968ddb4-fr9zz  1/1     Running   0          12m
pod/user-service-6bbb99b489-9rdwz       1/1     Running   0          12m
pod/user-service-6bbb99b489-mrrgf       1/1     Running   0          10s
pod/userservicedb-7d4855f6fb-h9z6k      1/1     Running   0          12m
```

As shown from the diagram, we are able to scale the number of specific backend services replicas depending on the backend service utilisation in order to match the current load from the users. The Horizontal Pod Autoscaler controls the number of replicas created / deleted based on the CPU metrics and triggers the deployment process.

```
jim@Jims-MacBook-Pro-2 k8s % k get all
NAME                                        READY   STATUS    RESTARTS   AGE
pod/chat-service-7fb5f84dc6-mcnmg           1/1     Running   0          6m32s
pod/matching-service-f94c795bf-rnqk8        1/1     Running   0          6m33s
pod/mongodb-56b8cfbfc8-szvxw                1/1     Running   0          6m33s
pod/question-service-54b5f98459-f96kn       1/1     Running   0          6m34s
pod/questionservicedb-6f4968ddb4-fr9zz      1/1     Running   0          6m35s
pod/user-service-6bbb99b489-9rdwz           1/1     Running   0          6m37s
pod/userservicedb-7d4855f6fb-h9z6k          1/1     Running   0          6m38s

NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
service/chat-service          ClusterIP   10.96.195.229   <none>        8003/TCP    6m32s
service/kubernetes            ClusterIP   10.96.0.1       <none>        443/TCP     13m
service/matching-service      ClusterIP   10.96.65.161    <none>        8001/TCP    6m33s
service/mongodb               ClusterIP   10.96.32.173    <none>        27017/TCP   6m33s
service/question-service      ClusterIP   10.96.160.146   <none>        8002/TCP    6m34s
service/questionservicedb     ClusterIP   10.96.148.147   <none>        5432/TCP    6m35s
service/user-service          ClusterIP   10.96.244.105   <none>        8000/TCP    6m37s
service/userservicedb         ClusterIP   10.96.122.253   <none>        5432/TCP    6m38s

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/chat-service           1/1     1            1           6m32s
deployment.apps/matching-service       1/1     1            1           6m33s
deployment.apps/mongodb                1/1     1            1           6m34s
deployment.apps/question-service       1/1     1            1           6m34s
deployment.apps/questionservicedb      1/1     1            1           6m35s
deployment.apps/user-service           1/1     1            1           6m37s
deployment.apps/userservicedb          1/1     1            1           6m38s

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/chat-service-7fb5f84dc6           1         1         1       6m32s
replicaset.apps/matching-service-f94c795bf        1         1         1       6m33s
replicaset.apps/mongodb-56b8cfbfc8                1         1         1       6m34s
replicaset.apps/question-service-54b5f98459       1         1         1       6m34s
replicaset.apps/questionservicedb-6f4968ddb4      1         1         1       6m35s
replicaset.apps/user-service-6bbb99b489           1         1         1       6m37s
replicaset.apps/userservicedb-7d4855f6fb          1         1         1       6m38s

NAME                                                                  REFERENCE                         TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/chat-service-hpa                  Deployment/chat-service           <unknown>/80%    1         3         1          6m32s
horizontalpodautoscaler.autoscaling/matching-service-hpa              Deployment/matching-service       <unknown>/80%    1         1         1          6m32s
horizontalpodautoscaler.autoscaling/question-service-hpa              Deployment/question-service       <unknown>/80%    1         3         1          6m34s
horizontalpodautoscaler.autoscaling/user-service-hpa                  Deployment/user-service           <unknown>/80%    1         3         1          6m36s
```

The picture above shows the kubernetes objects created during the deployment. We also have an ingress controller and objects.

8. Microservices

Microservice	Description
User Service	User authentication and authorization
Matching Service	Allows users to match with others based on question difficulty
Question Service	Collection of questions for users to solve
Collaboration Service	Codepad for matched users to collaborate their work
Communication Service	Chat for matched users to communicate

8.1. User Service

The user service handles the user account management. This includes the account creation, logging in and out of the web application, changing of passwords and account deletion. In addition, it manages authentication by creating JWT tokens and keeping track of blacklisted tokens.

8.1.1. Models

8.1.1.1 Tokens

Field	Type	Purpose
token	String	- Primary Key - JWT tokens that have expired or blacklisted

8.1.1.2 Users

Field	Type	Purpose
username	String	- Primary Key - Username of the the user
password	String	- Hashed password of the user

We used the bcrypt library to hash and store the password securely using the bcrypt hashing function. This adds an additional layer of confidentiality in the case where a malicious party gains access to the database layer, since the password is not stored in plaintext.

8.1.2. APIs

8.1.2.1 Signup

Description	Endpoint to sign-up as a user
Endpoint	**[POST]** api/user-service/signup
Request	Body:

	{
	"username": "",
	"password": ""
	}

Response	201 - user is created 400 - db error in creating user 409 - username is already taken

8.1.2.2 Login

Description	Endpoint to login
Endpoint	**[POST]** api/user-service/login

Request	Body:
	{
	"username": "",
	"password": ""
	}

Response	200 - login is successful and access token is returned
	{
	"accessToken": "",
	}
	
	400 - error produced by the server
	401 - invalid username and password

8.1.2.3 Logout

Description	Endpoint to log out
Endpoint	**[POST]** api/user-service/logout
Request	Headers: JWT Token
Response	200 - access token is blacklisted 401 - unauthorised 403 - token has expired

8.1.2.4 Change password

Description	Endpoint to change password
Endpoint	**[POST]** api/user-service/change-password
Request	Body: { "oldPassword": "",

	"newPassword": "" } Headers: JWT Token
Response	200 - password changed 400 - invalid old password or db error 401 - unauthorised 403 - token has expired

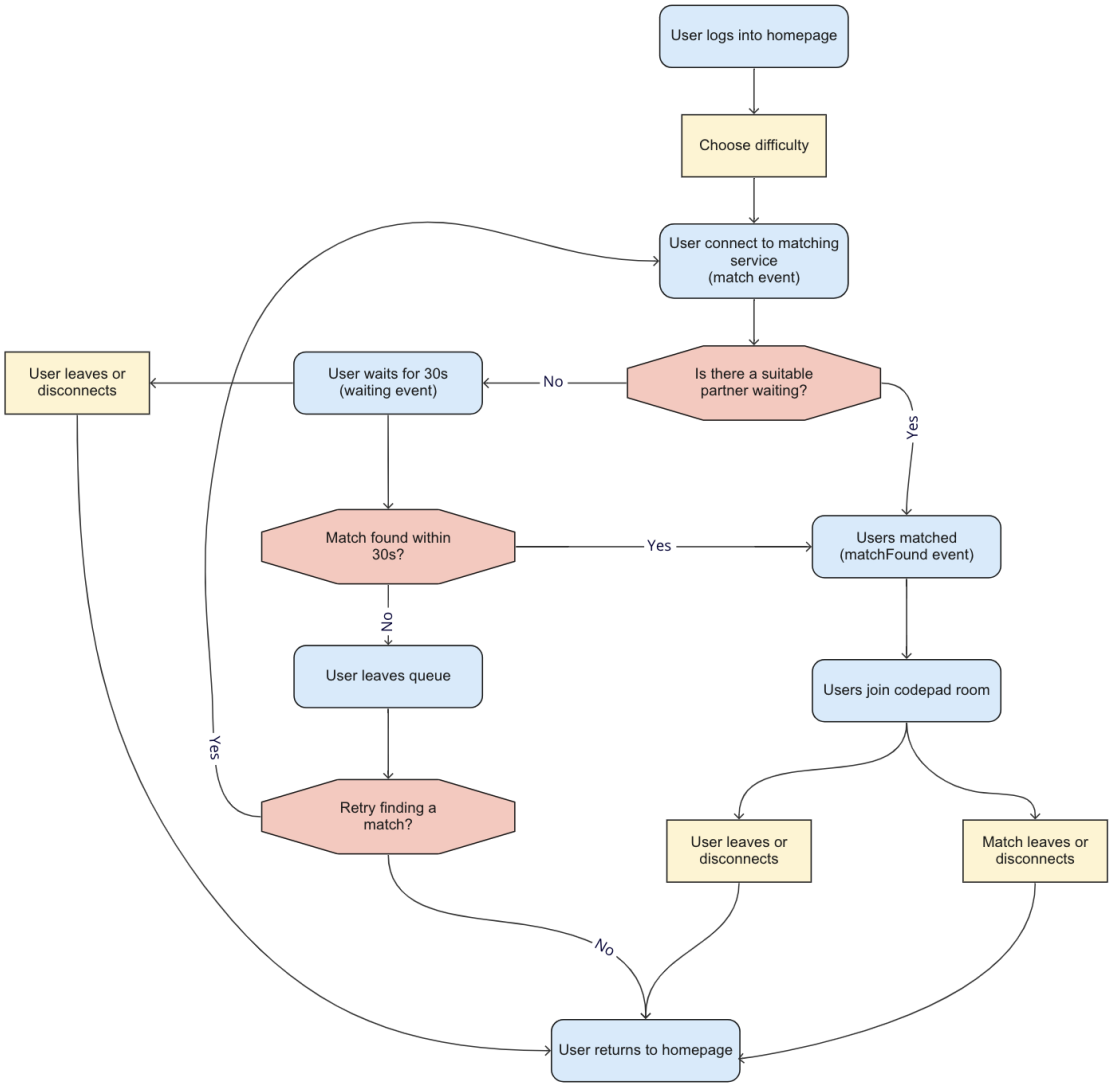
8.1.2.5 Delete account

Description	Endpoint to delete account
Endpoint	**[POST]** api/user-service/change-password
Request	Headers: JWT Token
Response	204 - user deleted 400 - error in deleting user or user does not exist 401 - unauthorised 403 - token has expired

8.2. Matching Service

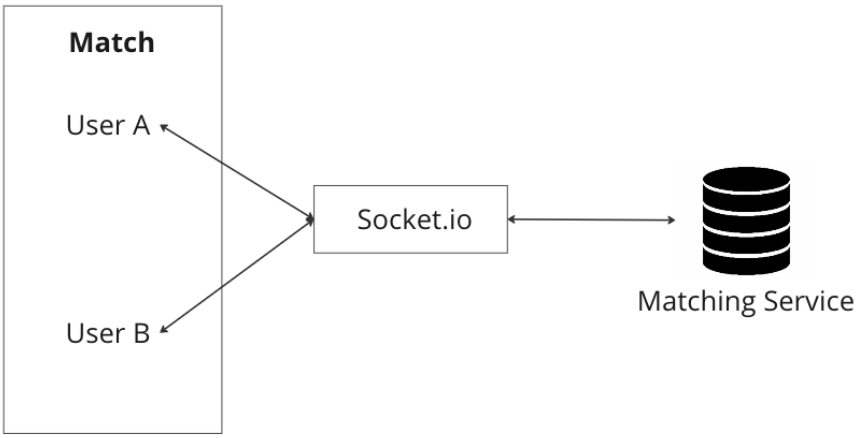
The Matching microservice is responsible for providing the users with the ability to pick a desired question difficulty and be matched with a suitable partner to work on the question together. The Matching microservice makes use of websockets to provide users live updates on the progress of their matching process and uses mongoDB to store pending matches and user socket connections.

The activity diagram below depicts the workflow for matching users. Users are matched according to the question difficulty selected, where two users who select the same difficulty will be matched with each other. Upon selecting a difficulty, users will be added to a queue, where they will wait for a maximum of 30 seconds. After the 30 seconds are up, they will be removed from the queue.

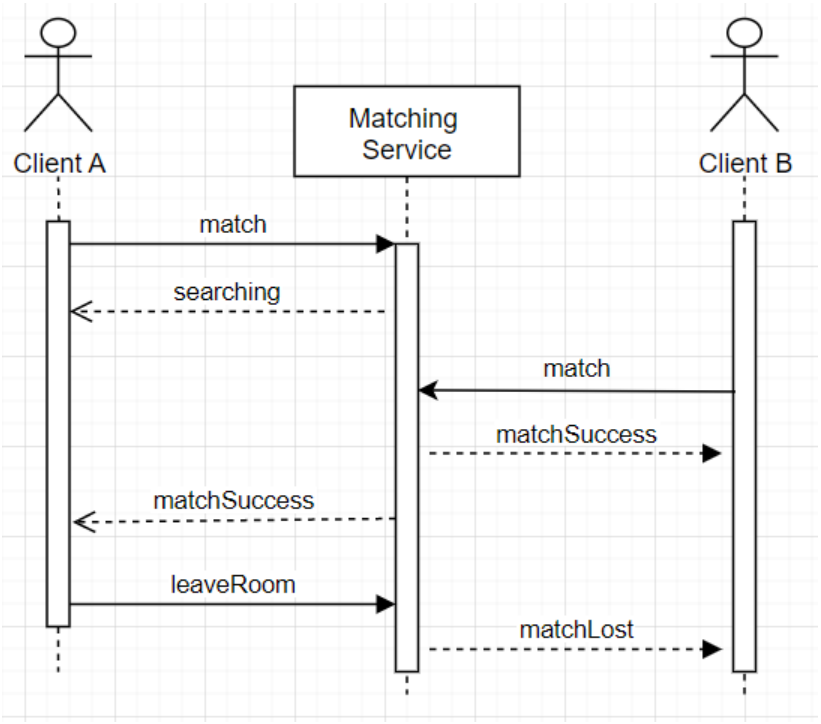


Matching Service Activity Diagram

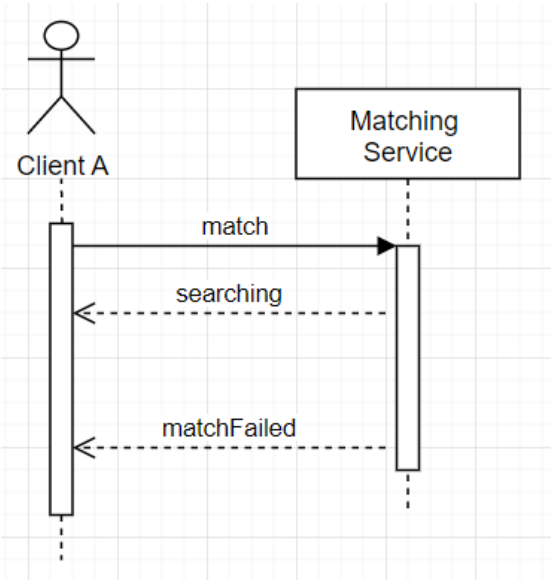
The server contains a server socket to communicate with the client sockets for different users. The users and the servers are both publishers and subscribers for different events. On entering the lobby, or leaving the matched room, the client publishes a “match” event or “leaveRoom” event respectively, which the server subscribes to. On finding a match between two different users, the server socket publishes a “matchSuccess” event, to which the two different users are subscribed to.



Publish-Subscribe Pattern for Matching Service



Sequence Diagram for Match Service



Sequence Diagram for Matching Service (Match not found within 30s)

8.2.1 Models

Pending Match		
Field	**Type**	**Purpose**
id	Integer	Primary Key
user	String	Username of waiting user
roomId	String	Room ID of the match room
difficulty	Enum	Question difficulty used for matching
created_at	datetime	Time that user started searching for match

User Connection		
Field	**Type**	**Purpose**
id	Integer	Primary Key
user	String	Username of connected user
socketId	String	Socket ID associated with user

8.2.2 Events

8.2.2.1 Client Events

8.2.2.1.1 Match

Description	Search for a match
Event	**match**
Arguments	1. UserID 2. Difficulty (Easy/Medium/Hard)
Response Events	**searching** - Match not found immediately **matchSuccess** - Match found immediately or within 30s **matchFailed** - Match not found within 30s

8.2.2.1.2 Leave

Description	Leave search or match
Event	**leaveRoom**
Arguments	1. UserID 2. RoomID
Response Events	

8.2.2.2 Server Events

8.2.2.1.1 Searching

Description	Inform user that match has not been found
Event	**searching**
Arguments	1. RoomID

8.2.2.1.2 Match Found

Description	Inform user that match has been found
Event	**matchSuccess**
Arguments	1. RoomID

8.2.2.1.3 Match Failed

Description	Inform user that match has not been found within the 30 second limit
Event	**matchFailed**
Arguments	1. Message

8.2.2.1.4 Match Lost

Description	Inform user that his/her matched partner has left/disconnected
Event	**matchLost**
Arguments	1. Message

8.3. Question Service

The Question microservice is responsible for providing users with questions of varying difficulties (easy, medium and hard). The questions are sourced from Leetcode and stored in a Postgres database. The database indexes the questions based on their difficulty to improve the efficiency of the database query. The following is the schema and information stored in the database.

8.3.1. Models

8.3.1.1 Question

Question		
Field	**Type**	**Purpose**
id	Integer	Primary Key
title	String	Question Title
url	String	URL to Leetcode question
difficulty	Enum	Question difficulty
prompt	String	Question prompt to describe the question
examples	Array(string)	Array of examples given by Leetcode to aid user's understanding of the questions
constraints	Array(string)	Array of constraints for the question
related_topics	Array(string)	Array of topics that are related to this question (e.g. Trees, Hash Maps)
similar_questions	Array(json object)	Array of JSON objects that contain the title, url and difficulty Given by leetcode for users to find similar questions to practise

8.3.2. APIs

8.3.2.1 Get question by difficulty

Description	Endpoint for a codepad room to retrieve a question based on a given difficulty
Endpoint	**[GET]** api/question-service/questions/difficulty/{difficulty}/{roomId}

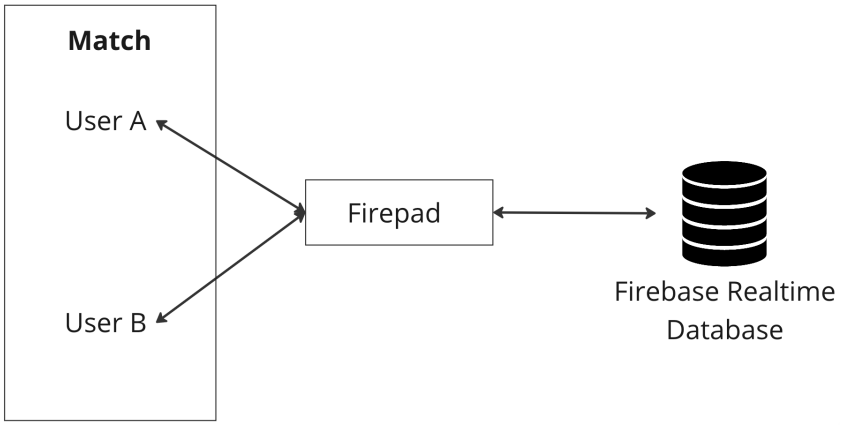
Request	Query parameters:
	{
	"difficulty": "easy" | "medium" | "hard"
	"roomId": "roomId"
	}

Response	200 - question successfully retrieved 400 - invalid difficulty provided or db error in retrieving the question

8.4. Collaboration Service

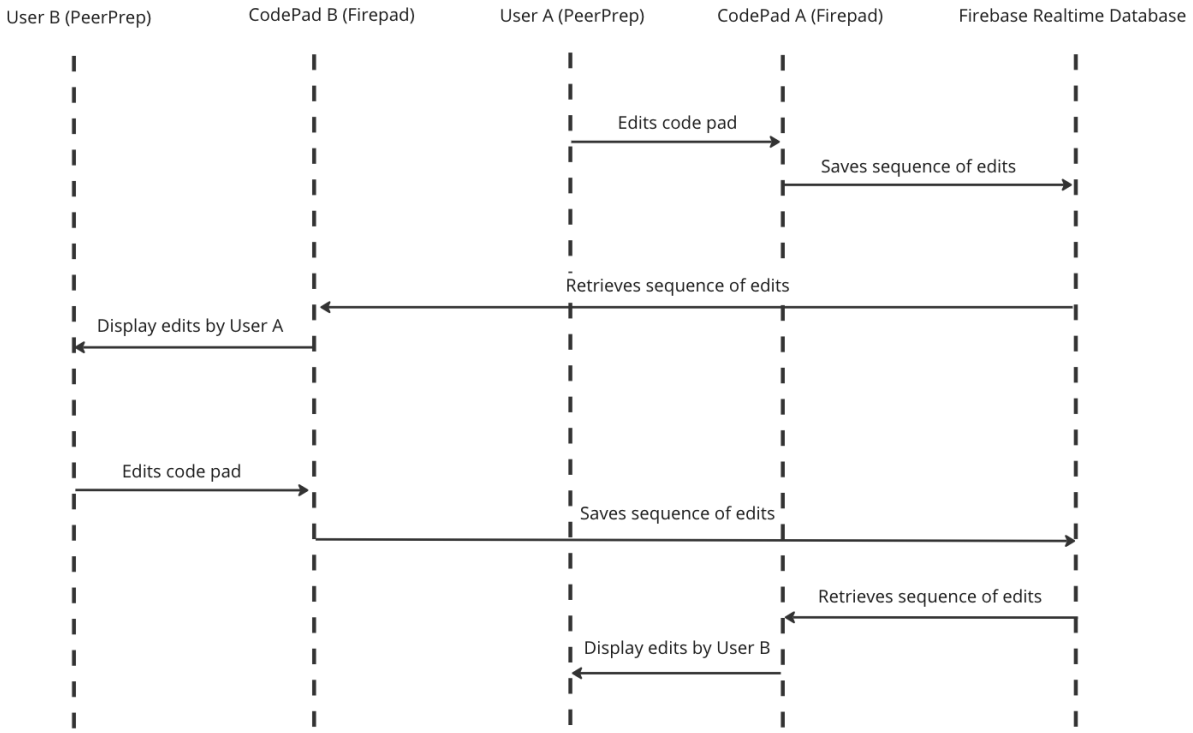
The Collaboration microservice provides users with a codepad for real-time collaboration on technical interview questions. The collaboration service leverages Firepad, which provides a real-time collaborative code editor without any server code.

Using the publish-subscribe pattern, the two users that are matched to each other are both publishers and subscribers to the same topic. When User A makes a change in the codepad, the edits made are published to that topic. User B, who is subscribed to the same topic, will receive the edits made by user A. In the same way, changes made by User B will be received by User A.



Publish-Subscribe Pattern for Collaboration Service

The sequence diagram below describes the process of users editing on the codepad. When a user makes edits in the codepad, the sequence of edits are captured by Firepad and saved into the Firebase Realtime Database. This sequence of edits are published to all subscribers, namely the other matched user in the same codepad page. The edits are then displayed, giving a near real-time collaborative experience.

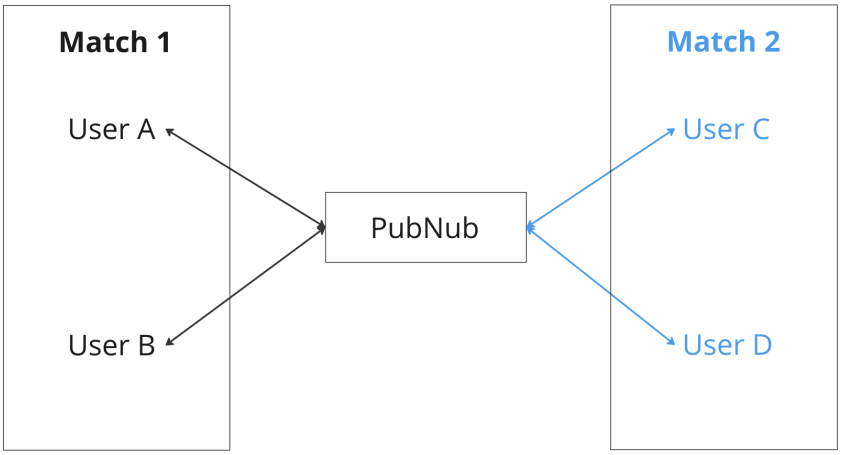


Sequence Diagram for the Collaborative Codepad

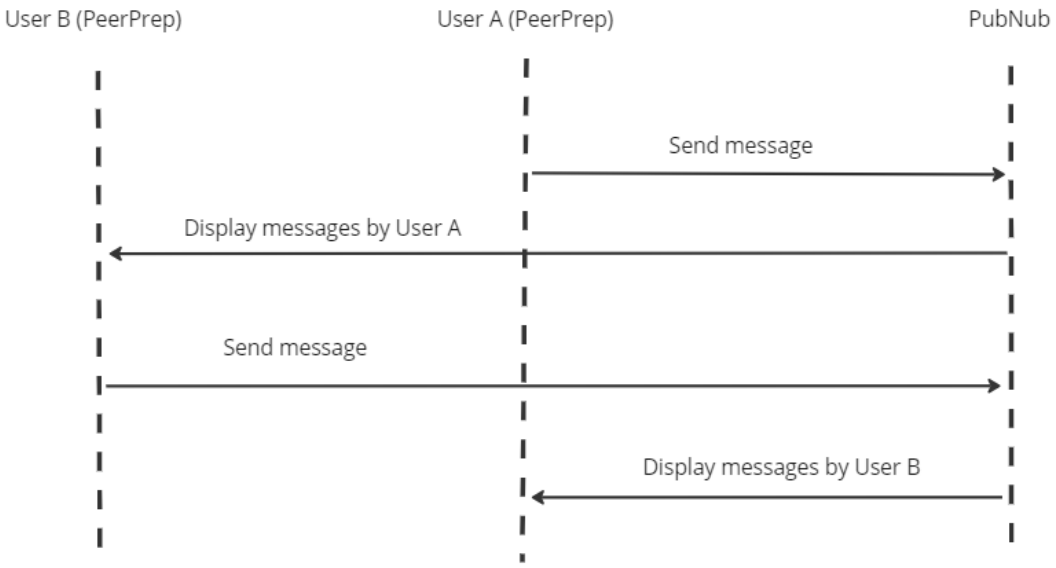
8.5. Communication Service

The Communication microservice provides matched users with a chat service for them to communicate through text during collaboration in the codepad room. The communication service makes use of PubNub, a real time chat and communication platform to provide users with the chat functionality. This provides for a better collaboration experience for matched users.

For the communication service, only users that are matched to each other will publish and subscribe to messages from the same topic. User A publishes a chat message to a topic, to which only User B is subscribed to, since they are matched together. Users C and D are subscribed to a different topic as they are in a different match.



Publish-Subscribe Pattern for Communication Service



Sequence Diagram for Communication Service

9. Suggested Improvements

9.1 Video Chat

One of the potential improvements that we felt could have been implemented is a video-chat communication service. Currently, our chat service provides users with a messaging component that they could use to write messages. In the long term, it would be harder for users to explain their thought process and elaborate quickly to their partner in the codepad. Having a video chat allows the user to speak while they are solving the question, as to allow the partner to understand their thought process (which is one of the things interviewers look out for in technical interviews)

9.2 Match/Question History

Another potential improvement we hope to extend is to save the history of matches and questions that a user has gone through. Having the match history allows us to generate statistics about the user's interview preparation. For example, we can show the users in the form of a graph, the frequency of them using the application, the amount of time they take to solve a question and use it to track their progress. The users can then identify the type of interview questions that they require more practice on and work on them accordingly.

In addition, by saving the Question History (and potentially the user's answers), we can allow the users to identify new questions that they can try and revise on their past solutions or mistakes.

9.3 Filters for Matching & Dedicated Questions + Suggested Solutions

In order to further enhance the user's experience when practising questions, we can add filters for the types of questions users wish to practise with. For example, the user can choose to practise Medium Graphs and Trees for their codepad interview, while another user can choose to practise Medium String-based questions. When the users are matched and placed in the codepad, the codepad displays two questions according to the filters set by the users. This gives each user a question dedicated to their preferences. For the users, we can also display suggested solutions so that they can understand the approach to solve the question. The interviewers can use it to guide their partners when they are practising their questions. We believe these would enhance the experience of the users when they are practising using our application.

10. Reflections & Learning Points

10.1 Design deliberately with the end-goal in mind

One of our pain points during our development process was to adapt our existing microservices to a docker/cloud based deployment. Initially, we designed and developed the microservices to work in a local environment. This went well as we are able to create the microservices and integrate them well together.

As the project progresses, we felt that it would be useful to dockerise them and shift them onto a container-based (and subsequently cloud-based) deployment. Hence, we started to dockerise the microservices, adding NGINX ingress and Kubernetes clusters to contain the Docker images. However, we had to identify how to properly integrate the application and resolve bugs which required substantial effort given that the microservices are already mature (it was hard to tell if the bugs come from the microservices or the deployment configuration/.yml files). Hence, we believed we could have done better by planning and designing our project more intently with the end goal of a cloud-based deployment in mind.