



## **CS3219 Team Project Report for Peerprep**

Student	Alvin Chee	Marcus Tan	Ngoh Wei Yue	Rishi Ravikumar
Matric Number	A0201958B	A0149980N	A0205671J	A0206006X

### **Repository Link:**

<https://github.com/CS3219-SE-Principles-and-Patterns/cs3219-project-ay2122-2122-s1-g23>

<b>1. Introduction</b>	<b>5</b>
1.1 Peerprep	5
1.2 Motivation & Inspiration behind the choice	5
<b>2. Functional Requirements:</b>	<b>6</b>
User Account Management	6
Matchmaking System	6
Mock Interview System	7
Client Application	7
<b>3. Non-functional requirements:</b>	<b>9</b>
3.1 Performance requirements	10
Key points	10
Priorities	10
3.2 Usability requirements	11
Key points	11
Priorities	11
3.3 Availability Requirements	13
Key points	13
Priority	13
3.4 Scalability Requirements	14
Key points	14
Priority	14
<b>4. Architectural Design</b>	<b>15</b>
Architecture Diagram	15
Architecture Decisions	16
<b>5. Design Patterns</b>	<b>19</b>
5.1 MVC Pattern	19
5.2 Pub-Sub Pattern	19
5.3 Database per service Pattern	20
<b>6. DevOps</b>	<b>22</b>
6.1 Sprint Process	22
6.2 CI/CD	24
<b>7. Application Design</b>	<b>25</b>
7.1 Tech Stack	25
<b>8. Backend</b>	<b>27</b>
8.1 User Account Management Microservice	27

8.2 Match Microservice	34
8.3 Editor Microservice	37
8.4 Message Microservice	38
8.5 Questions Microservice	39
<b>9. Frontend</b>	<b>43</b>
<b>10. Application Screenshots</b>	<b>45</b>
10.1 User Authentication	45
10.2 User Home Page	47
10.3 Matching System	49
10.4 Interview	50
10.5 Feedback session	52
10.6 Friends List	<b>53</b>
<b>11. Remarks</b>	<b>56</b>
11.1 Challenges Faced	56
11.2 Potential Extension Features	56
1. Audio & Video communication	56
2. Code completion features	57
11.3 Reflections	57
<b>References</b>	<b>58</b>
<b>Appendix A</b>	<b>59</b>

## Contributions

Member	Technical contributions	Non technical contributions
Alvin Chee	Backend Development Implement Match Microservice GKE deployment	Requirements conceptualisation Project report
Marcus Tan	Implement Messaging Microservice Implement Frontend GCR (Google Cloud Run) deployment Integrate CI/CD	Requirements conceptualisation Project report
Ngoh Wei Yue	Implement Code Editor Microservice & User Microservice Set up Database	Requirements conceptualisation Project report
Rishi Ravikumar	Implement Frontend & Authentication Implement User Microservice (Friends) & Questions Microservice	Requirements conceptualisation Project report

# 1. Introduction

## 1.1 Peerprep

We will be embarking on the project Peerprep, a *peer support system* where users are engaged with *collaborative whiteboard-style programming* to practice for technical interviews. The app is expected to handle collaborative whiteboard-style programming. With IPP, everyone can code together on the same file at the same time. You can share with anyone, view edited code in real time, chat and comment for discussions. The app also supports some basic features as well e.g., user authentication and save and retrieve relevant data. Questions will be taken from leetcode (free questions), we will keep the sample size of our questions small as we are in the minimal viable product stage. The questions will be categorized based on difficulty (e.g easy, medium, hard).

## 1.2 Motivation & Inspiration behind the choice

Aspiring Software Engineering students often have trouble with technical interviews. Some may find it stressful to be assessed live coding, while others find it tough to articulate their thoughts properly while coding at the same time, including us too! In that perspective, we want to create an interview preparation platform where students can find peers to practice whiteboard style interview questions together, as well as fostering a local community for tackling technical challenges together.

We felt that the idea of PeerPrep was really an interesting one and that it can be purposeful for ourselves. We feel that the challenges faced aforementioned can be overcome through rigorous practicing, and we want to implement such a platform for students to engage in.

## 2. Functional Requirements:

1. User Account Management		
S/N	Functional Requirement	Priority
1.1	The user account management microservice shall allow the registration of accounts using a ".edu" email address with a password.	High
1.2	The user account management microservice shall allow users to login to the web application using their registered email account and password.	High
1.3	The user account management microservice shall allow users to view their profile, which contains previous mock interview details (date, interview partner, question) and their aggregated user XP (based on quantity and level of peer ratings & question difficulty).	Medium
1.4	The user account management system will allow users to view up to 6 past mock interviews on their profile.	Medium
1.5	The user account management microservice will allow users to add other users as a friend.	Medium
1.6	The user account management microservice will be able to display their list of friends.	Medium

2. Matchmaking System		
S/N	Functional Requirement	Priority
2.1	The matchmaking microservice will assign every user a default XP of 0 at the beginning.	High
2.2	Every user's XP will be an integer sum that is accumulated from their previous mock interviews.	High
2.3	The matchmaking microservice will match users with XP within a 100 range.	High

2.4	After every interview, the matchmaking microservice will allocate each user a range of XP based on the peer rating and the difficulty of the question attempted. (Easy: up to 130XP, Medium: up to 190XP, Hard: up to 300XP)	Medium
-----	---	--------

<b>3. Mock Interview System</b>		
<b>S/N</b>	<b>Functional Requirement</b>	<b>Priority</b>
3.1	The code editor microservice will allow both users to view and edit on a collaborative code editor in near real-time during the mock interview.	High
3.2	The chat microservice will allow users to communicate with each other through a chat box in near real-time during the interview.	High
3.3	The frontend UI will show a pop up screen for both users to rate each other based on a scale of 1 to 5 immediately after an interview ends.	Medium
3.4	The frontend UI will allow users to scroll and select a friend to request for a mock technical interview with a random question allocated for the interview.	Medium

<b>4. Client Application</b>		
<b>S/N</b>	<b>Functional Requirement</b>	<b>Priority</b>
4.1	The application UI will serve a landing page to introduce potential users to the purpose of our application.	Medium
4.2	The application will limit the time for users to respond to interview requests to 30 seconds.	Medium



### **3. Non-functional requirements** (Listed in order of priority):

- 1) Performance**
- 2) Usability**
- 3) Availability**
- 4) Scalability**

Since this is a collaborative web application where users have to interact in near real-time for optimal experience, we decided to have performance as our top priority.

Users should not be facing latency issues that will hugely affect their experience.

Next, we chose usability as our second most important priority because we want to onboard users to our application quickly and not face any frustrations or barriers when using it. This is important because we feel that this will encourage users to come back again to using our application.

Thereafter, we have availability because we understand the importance of having an application running when users need them, especially when their technical interviews are around the corner!

Lastly, we prioritise scalability last because we do not expect a large number of users in the initial stage of our development.

### 3.1 Performance requirements

#### Key points

1. The application will ensure at most 3 seconds response delay during live coding when there is a change in the content in the collaborative notepad.
2. The application will not take more than 5 seconds to change from one screen to another.
3. API calls from frontend to backend should have a response time of <3s during peak periods

#### Priorities

1. Performance		
S/N	Non-functional Requirements	Priority
1.1	At most 3 seconds response delay during live coding	High
1.2	The application UI will not take more than 5 seconds for switching of screens	Medium
1.3	The API calls from frontend to backend shall have a response time of lower than 3 seconds during peak periods.	Low

#### Load testing

For this rendition of PeerPrep, we utilised *hey*, a HTTP load testing tool to stress test our application with a barrage of HTTP requests.

```

[rishi@Rs-MacBook-Pro-2 ~ % hey -n 30000 -c 50 -z 3m -t 0 http://www.peerprelegends.com/api/user/

Summary:
  Total:      180.0159 secs
  Slowest:    0.3677 secs
  Fastest:    0.0047 secs
  Average:    0.0107 secs
  Requests/sec: 4675.2492

  Total data: 36189617 bytes
  Size/request: 43 bytes

Response time histogram:
  0.005 [1] |
  0.041 [835095] |
  0.077 [1939] |
  0.114 [618] |
  0.150 [3299] |
  0.186 [540] |
  0.223 [69] |
  0.259 [57] |
  0.295 [0] |
  0.331 [0] |
  0.368 [1] |

Latency distribution:
  10% in 0.0071 secs
  25% in 0.0079 secs
  50% in 0.0089 secs
  75% in 0.0107 secs
  90% in 0.0138 secs
  95% in 0.0173 secs
  99% in 0.0360 secs

Details (average, fastest, slowest):
  DNS+diapup: 0.0000 secs, 0.0047 secs, 0.3677 secs
  DNS-lookup: 0.0000 secs, 0.0000 secs, 0.0321 secs
  req write: 0.0000 secs, 0.0000 secs, 0.0601 secs
  resp wait: 0.0106 secs, 0.0046 secs, 0.3676 secs
  resp read: 0.0000 secs, 0.0000 secs, 0.0554 secs

Status code distribution:
  [200] 841619 responses

```

Figure: Load testing results

With a sample test of 50 users and 30000 requests, we are comfortably able to maintain the desired performance metrics of <3s per API call and up to 50 users accessing the application as part of the **availability** requirements too.

## 3.2 Usability requirements

### Key points

1. Intuitive, easy to use frontend - user flows through the application don't take much steps to execute/remember
2. Interactive leveling system (when gaining more exp and leveling up, users get cool virtual cosmetic badges)
3. Minimalistic, straightforward design

### Priorities

## 2. Usability

<b>S/N</b>	<b>Non-functional Requirements</b>	<b>Priority</b>
2.1	Intuitive and user friendly UI such that the whole mock interview process will be an easy flow for users.	High
2.2	Minimalistic, straightforward design where there are no unnecessary frontend components/animations/transitions distracting the user.	Medium
2.3	Token based authentication utilised using cookies so that users can easily access into the application without the need to key in their login details - cookie expires in 2 days.	Medium
2.4	Interactive leveling system - users gain more XP when rated more highly and attempt more challenging questions, out of which they are rewarded with various titles & stickers on the app UI.	Low

In order to set a benchmark for our website's usability, we utilised the industry standard's System Usability Scale, where we invited 50 NUS computer science students to use our platform and provide feedback through a series of 10 questions.

Below are the 10 questions, where surveyors can choose from 5 responses ranging from "Strongly Disagree" to "Strongly Agree".

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

Figure<sup>1</sup>: SUS's list of questions

Below are the results:

Minimum score: 65

Maximum score: 95

Average score: 82.5

This gives us an average 'A' grade.

### 3.3 Availability Requirements

Key points

1. The system should be up at least 99% of the time.

Priorities

3. Availability		
S/N	Non-functional Requirements	Priority
4.1	The application has a minimum of 99% uptime during peak hours.	High

Service/Node	Redundancy	Failure Detection	Fallover	Replication
Frontend	-	-	-	-
Kubernetes Work nodes	Multi-Region	Ping	Ingress Gateway	Cluster Autoscaler
Users MS	Horizontal Pod Scaling	Ping	Ingress Gateway	HPA
Match MS	Horizontal Pod Scaling	Ping	Ingress Gateway	HPA
Editor MS	Horizontal Pod Scaling	Ping	Ingress Gateway	HPA
Message MS	Horizontal Pod Scaling	Ping	Ingress Gateway	HPA
Questions MS	Horizontal Pod Scaling	Ping	Ingress Gateway	HPA
MongoDB	Horizontal Scaling	Ping	DB	Atlas Cluster Autoscaling

GCP and Mongo Atlas have stated multiple times in their sites over the high certainty of >99% uptime for their app using the relevant software's configurations - through which we are also able to ensure >99% uptime during peak hours for our app too.

### 3.4 Scalability Requirements

#### Key points

1. The system can host 5 concurrent interviews at any point in time
2. The system should support >50 concurrent users browsing PeerPrep

#### Priorities

4. Scalability		
S/N	Non-functional Requirements	Priority
3.1	The system can host 5 concurrent interviews at any point in time	Medium
3.2	The system should be able to support up to 50 concurrent users browsing the application	Medium
3.3	The system should be able to support more than 500 registered users in its database.	Low

Our Kubernetes cluster is set to have a minimum of 1 pod and a maximum of 3 pods, with Horizontal Pod Autoscaling. The Horizontal Pod Autoscaler automatically scales the number of Pods based on observed CPU utilization. Vertical Scaling is also enabled in the event that there are too many CPU requests.

1. The ingress nginx controller serves as an API gateway for application routing to the different microservices so that only one endpoint is required to be exposed externally.
2. If there is an excess of traffic to any particular service pod, the HPA recognises the raised load and increases the number of pods gradually.

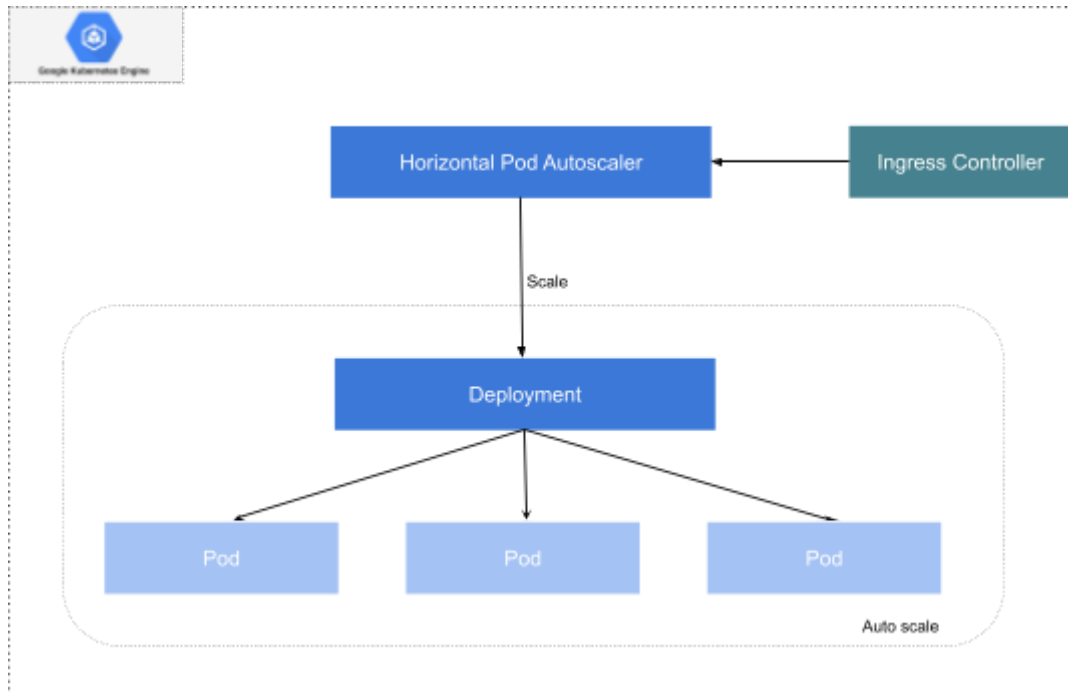


Figure: Scaling architecture implemented with HPA on GKE

## 4. Architectural Design

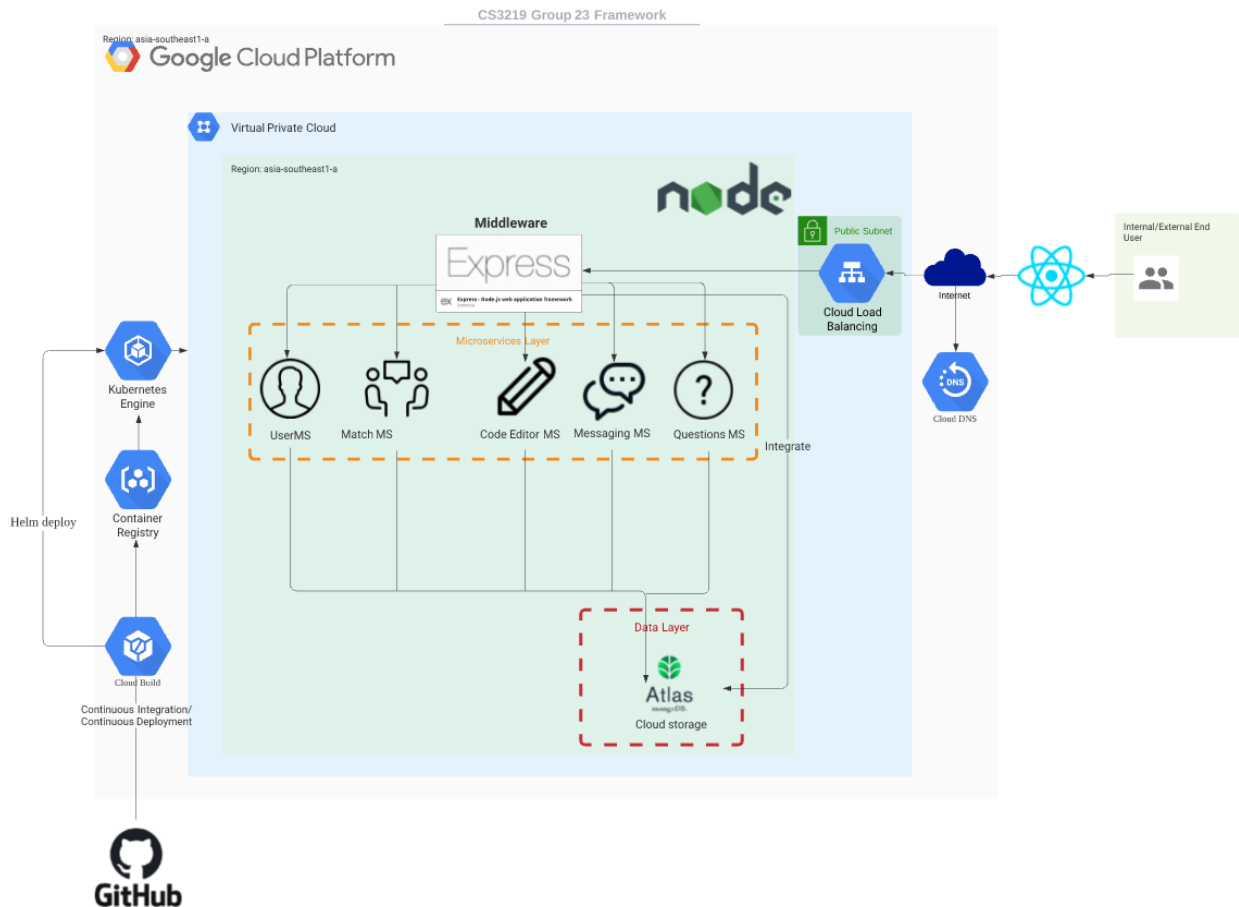


Figure: Architecture diagram

### Explanation:

1. Google Cloud Platform Load Balancer distributes traffic across multiple instances of applications.
2. As the user navigates through the application, requests are made to the REST api implemented at **www.peerprelegends.com/api**, which are then directed to the Google Load Balancer set up at the public subnet.
3. Google Load Balancer controls the traffic flow and directs client requests to the appropriate endpoints on the REST API, based on the service requested by the client
4. Inter-communications within microservices are handled by the internal application load balancer, staying within the boundary of the private subnet where the microservices are held
5. API endpoints are routed to fetch data from cloud database hosted on Mongo Atlas



## Architectural Decisions

Standpoint	Microservice architecture	Monolith architecture
Benefits	<ul style="list-style-type: none"><li>• Reduces the coupling of different logical components.</li><li>• Builds a manageable code base for each service, easing up the testing and troubleshooting of software</li><li>• Enables for the autoscaling of individual services based on load received.</li><li>• Easier to delegate responsibilities among team-mates</li></ul>	<ul style="list-style-type: none"><li>• More familiar, beginner-friendly architecture</li><li>• Easier to set up and deploy</li></ul>

### Decision: Microservice architecture

Given that we only have a few weeks to develop a working application, we felt that we should prioritise having an architecture that allows us to ramp up quickly. Having the microservice architecture allows us to split our team up so that each one of us can implement and test a part of the application without any dependencies from each other.

Since each module is independently deployed and scaled, it also allows us to ensure that scaling can be done faster and the application can still be largely unaffected in the case of a failure of a single module. This helps us achieve our non functional requirement of performance.

Standpoint	Kubernetes cluster	Docker compose
Benefits	<ul style="list-style-type: none"> <li>• Reduces the coupling of different logical components.</li> <li>• Builds a manageable code base for each service, easing up the testing and troubleshooting of software</li> <li>• Enables for the autoscaling of individual services based on load received.</li> </ul>	<ul style="list-style-type: none"> <li>• Simpler management since it can just be executed on 1 host</li> <li>• Easier to tap on the hot reload features by setting up volumes for each image created per service</li> </ul>

### Decision: Kubernetes for deployment

Kubernetes is easier to use for deployment with features like auto-scaling and we are able to tap on Google Kubernetes Engine to further expedite the deployment process onto the cloud, which isn't feasible & straightforward if docker-compose clusters were to be used.

Standpoint	Ingress controller	API gateway
Benefits	<ul style="list-style-type: none"> <li>• Only a singular entry point</li> <li>• Coordinate with GCP Load Balancer to easily distribute incoming load</li> </ul>	<ul style="list-style-type: none"> <li>• More control over the endpoints specified</li> <li>• Higher flexibility</li> <li>• Authentication features</li> </ul>

### Decision: Ingress controller

Considering the relatively small feature set and simplicity of our application, an ingress controller is more than necessary for our use cases with less effort required. An API gateway does provide more control over the distribution of traffic to our services but it is difficult to anticipate and manage with multiple entry points - which is unnecessary for our small application.

## 5. Design Patterns

### 5.1 MVC Pattern

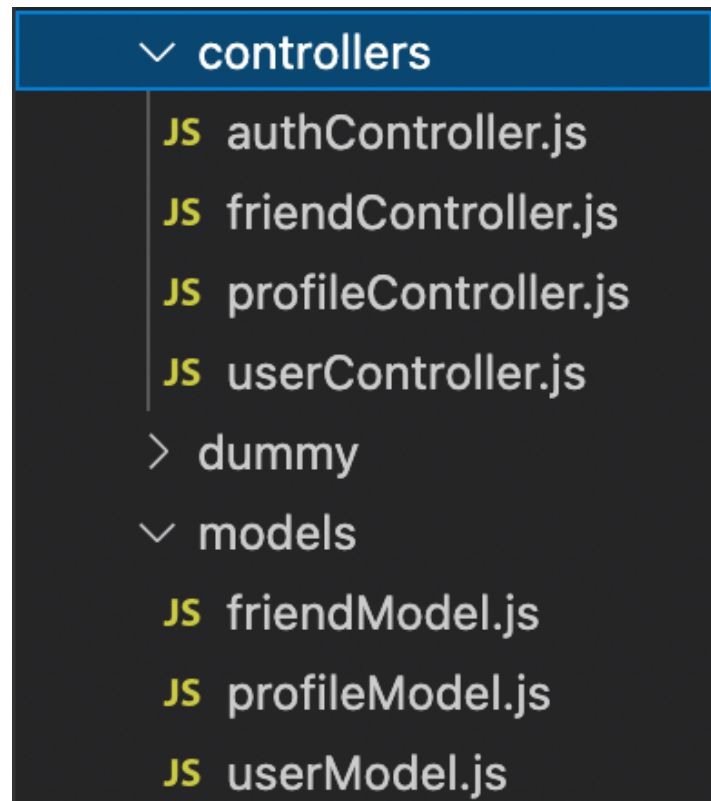


Figure: Code organisation

In our project, we utilised the MVC pattern on our User Account Management and Match microservices. This allows us to practise Separation of Concerns to increase modularity in our microservices.

- User output and input is now handled by the View component which is implemented in the frontend
- Each microservice's schema and data related logic is handled by the Model component
- Each microservice's business logic and incoming requests is handled by the Controller component

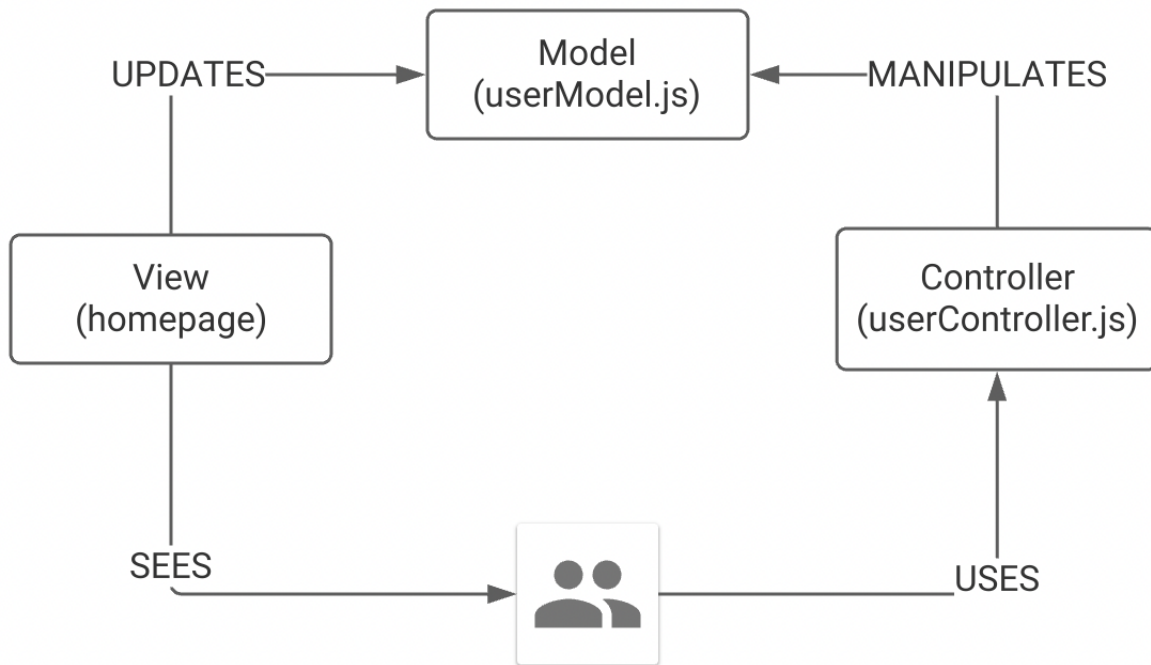


Figure: Example of a MVC pattern used in the User Account Management Microservice

## 5.2 Pub-Sub Pattern

To integrate the Match, Editor and Message microservices together, we employed the usage of Socket.io for real-time communication.

Editor Microservice and Message Microservice each contain a Server socket that allows for communication with the Editor and Message component in the frontend respectively. On any changes (typing a word for example) in the Editor or sending a message in the chat tab, the Client socket will emit a signal that is subscribed by the Server socket, and Server socket will emit to the respective target topic on receiving any signals

The Home page also has a Client socket that allows for receiving signals for any incoming practice session request from a friend. This signal is sent from the Server socket at Match Microservice. In the home page, a user can also send a practice request to a friend (target) on his friend list and the Client socket will manage the emitting of request signals to the Service socket, which will then redirect to the specific topic listened by the target.

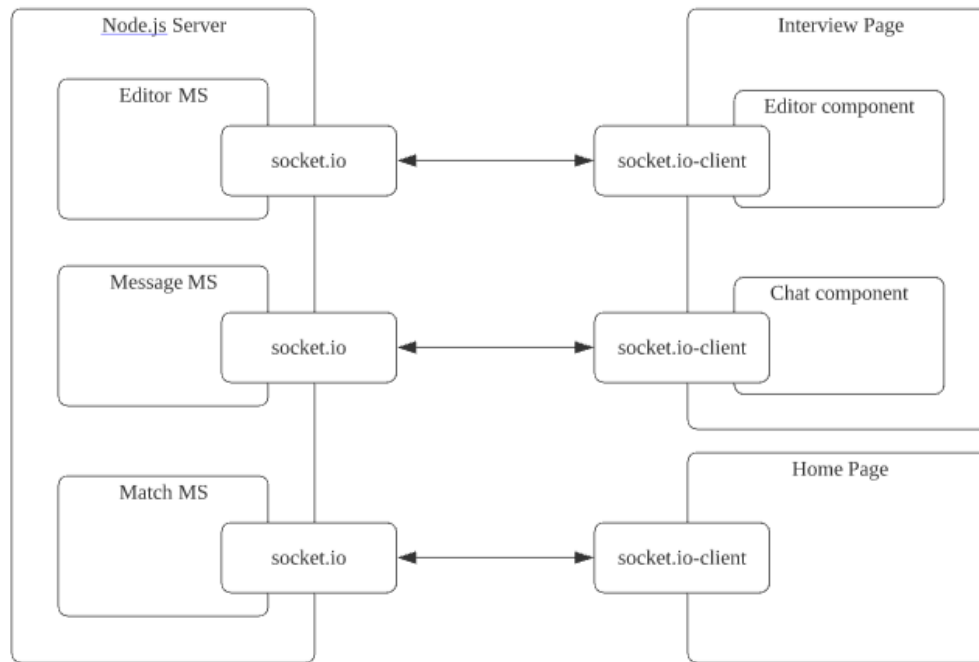


Figure: Pub-sub Pattern for Peerprep

### 5.3 Database per service Pattern

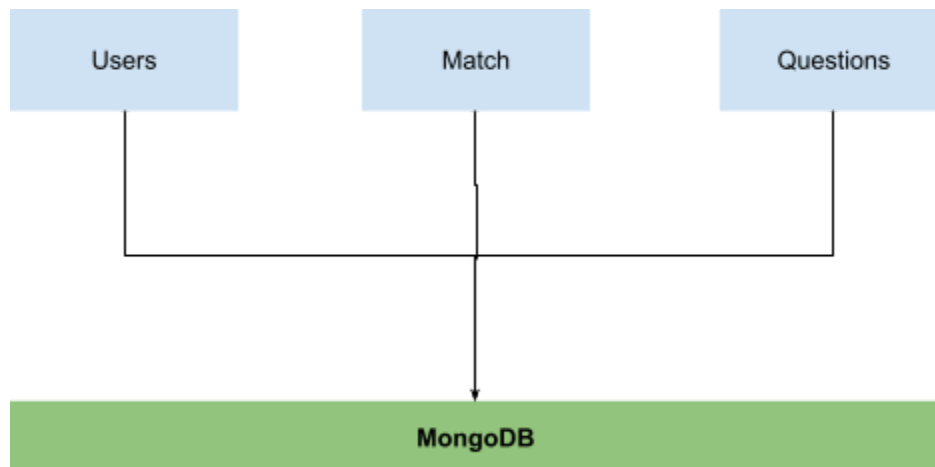
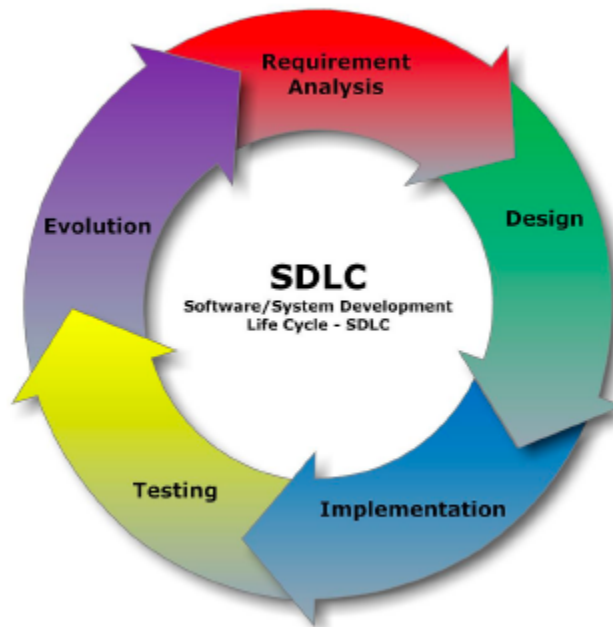
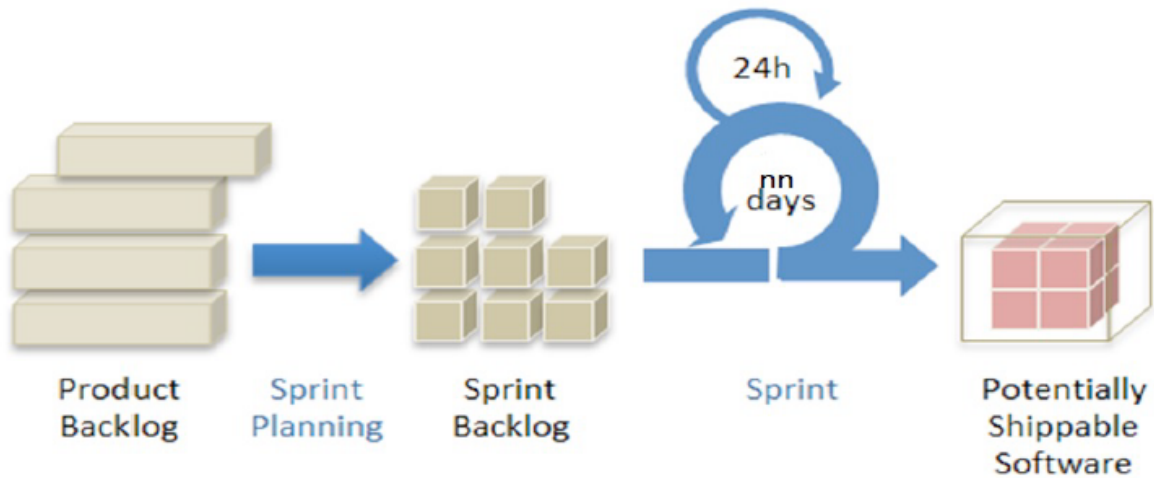


Figure: Database structure for microservices that use database

In our microservices, we adopted the database per service pattern to a high extent. The Match Microservice, Question microservice & Users are clear exemplifications of this, since they are relatively disparate from one another in functionality and data persisted.

## 6. DevOps

### 6.1 Sprint Process



Our team followed the agile development process when embarking on this project. This allows for iterative development, where we can build on to the application incrementally.

Our sprint cycle is within a week and each sprint goes through a whole cycle of planning, designing, developing, testing, review and deployment. We set up weekly scrum every Friday in

order to keep track of tasks, ensure accountability and to discuss what is to be done in the coming week in order of priority. We will first demonstrate what we have accomplished in that week. Thereafter, we will list out what features we aim to complete in the coming week and reassess any sprint backlogs from the previous sprint to be put into the product backlog or the next sprint. After reassessment, we will set out the goal for the coming week.

We met twice a week for standups over zoom meetings to resolve any blockers and update the team on the individual progress. We will also conduct a mini code review on the pull requests that are open if we have time.

## Milestone 1

Closed 22 days ago 100% complete

0 Open ✓ 12 Closed		
☐	✔ Design database schema <span>documentation</span>	👤👤
#3 by Rishi5154 was closed 22 days ago		
☐	✔ Set up deployment <span>High</span>	👤
#15 by Rishi5154 was closed 22 days ago		
☐	✔ Implement Auth frontend <span>High</span>	👤
#16 by Rishi5154 was closed 25 days ago		
⚡ ☐	✔ Set up frontend for registration & login <span>High</span>	👤 6
#19 by Rishi5154 was merged 25 days ago • Approved		
☐	✔ Create User Account Microservice <span>High</span>	👤
#8 by nweiyue was closed 27 days ago		
☐	✔ Setup MongoDB database <span>High</span>	👤
#2 by nweiyue was closed 27 days ago		
☐	⚡ Add User Account MS	14
#9 by nweiyue was merged 27 days ago		

Figure: Sample sprint review

We conduct sprint reviews at the end of each week via Zoom, as observed at the diagram above. We will merge all Pull Requests once approved (with review comments via Github/other communication platform) and ensure that deployment is successful.

## 6.2 CI/CD

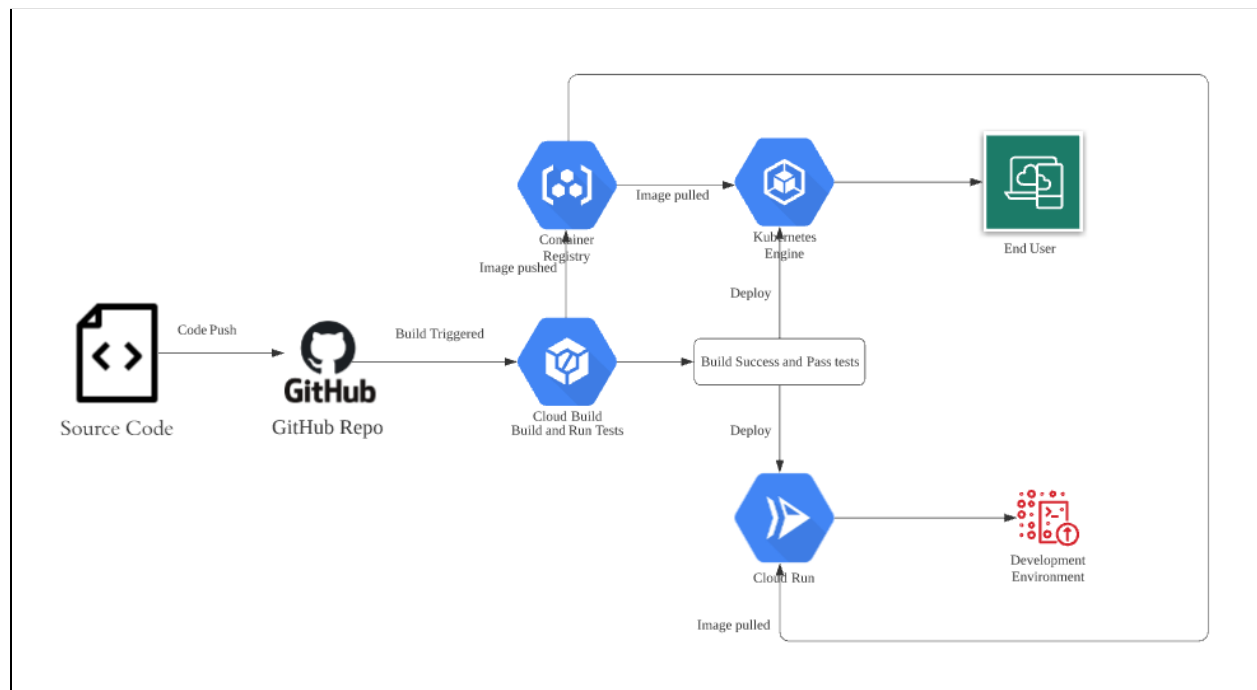


Figure: Simple CI/CD flow using Cloud Build

Our team chose Cloud Build as our CI/CD tool because it complements the usage of Google Kubernetes Engine and Cloud Run.

### Requirements:

1. Create the relevant cloudbuild.yaml files, telling Cloud Build what actions to take (build, test and deploy).
2. Create Cloud Build trigger that listens to push to branches (or alternatively push to tags or pull requests) and enter your environment variables.

Once our feature is completed and tested locally, we can push the source code to our target branch where it fires off the Cloud Build trigger. Cloud Build will then build the relevant containers as images and push to Google Container Registry. After that, it will run the unit tests. If tests failed, we will receive notification on GitHub that the CI/CD has failed. If tests passed, Cloud Build will deploy the built container images in Container Registry to Kubernetes Engine using Helm.

Development environments will be tested with Cloud Run before QA and Production on Kubernetes Engine. This is because Cloud Run is flexible and it supports Stateless containers that will be dramatically cheaper for testing purposes, since testing has low-traffic as our developer team is small.



## 7. Application Design

### 7.1 Tech Stack

Type	Choice	Explanation
FrontEnd	React React Bootstrap Material UI Prettier ESLint	<ul style="list-style-type: none"><li>• Virtual DOM feature that allows for quick rendering of UI</li><li>• Second most popular library used by developers, easy to get community support</li><li>• Some members of the team has prior experience in React</li><li>• Subtle performance differences were negligible in our application as compared to usage of VueJS</li><li>• React also supports variability in data as compared to AngularJS</li></ul>
BackEnd	Node.js Express Mongoose	<ul style="list-style-type: none"><li>• ExpressJS is a prebuilt NodeJS framework that can help us create a server-side web application faster and smarter.</li><li>• ExpressJS is also simple, minimalistic and flexible that inherits from Node.js, so it helps us to build the API in a short span of 6 weeks and the performance was also up to standards.</li><li>• Furthermore, prior to the project, our team has built a simple API using ExpressJS and Node.js, hence, we are familiar with this framework.</li><li>• Typically MERN stack where only we only need to employ JavaScript/TypeScript</li></ul>
Database	MongoDB (using cloud db with Mongo Atlas)	<ul style="list-style-type: none"><li>• Non-relational database like MongoDB have flexible data models allow us to change easily and quickly</li><li>• Data models parsed as JSON objects, easy to evolve and store data</li><li>• MongoDB Atlas monitoring tool</li><li>• Free to use</li></ul>
Cloud Providers	GCP	<ul style="list-style-type: none"><li>• Team has prior exp experience with using GCP for previous assignments or internships</li><li>• Free credits were given and used</li><li>• Extensive and large number of products (CloudBuild, CloudRun, GKE etc)</li><li>• Easy to use, secured</li></ul>

CI/CD	Google CloudBuild	<ul style="list-style-type: none"> <li>• Easy to implement with GCP</li> <li>• Simple procedure to setup CI/CD workflow using creation of build triggers</li> </ul>
Pub-Sub Messaging	Socket.io	<ul style="list-style-type: none"> <li>• Enables bidirectional and event based communication between client and server.</li> <li>• Supports reverse proxies and load balancers when deployed to GCP</li> <li>• Rooms feature compliments the practice session between two matched users</li> </ul>
Deployment	Google Kubernetes Engine  Google Cloud Run	<ul style="list-style-type: none"> <li>• Cloud Run allowed us to deploy backend microservices as Serverless containers and was suitable for development testing</li> <li>• Each revision in Cloud Run is also have automatically scaling to the number of container instances needed to handle all incoming requests</li> <li>• Google Kubernetes Engine allows for a managed, production-ready environment for running containerized application, suitable for production purposes with large number of users</li> <li>• GKE is also scalable with built-in resource for horizontal pod autoscaling</li> </ul>
Service Discover	GCP Load Balancing	<ul style="list-style-type: none"> <li>• Ability to scale application on Compute Engine from zero to full throttle</li> <li>• Distributes load-balanced compute resources in single region to meet high availability requirements</li> </ul>
Project Management Tools	GitHub Issues  GitHub Project (Kanban Board)	<ul style="list-style-type: none"> <li>• Convenient to use since organisation has created a Github repository for us to host the entire project</li> <li>• Ability to link GitHub issues to pull requests</li> <li>• Free to use</li> </ul>

## 8. Backend

Host/Deployment URL = <http://www.peerprelegends.com>

### 8.1 User Account Management Microservice

The User account management microservice consists of 3 main sections, namely, auth, users profile and friends.

Notable supported endpoints:

#### **Auth:**

##### Register

End-point: <HOST>/api/auth/register

Request method: POST

Sample body in JSON:

```
{
  "username": "admin",
  "email": "admin@u.nus.edu",
  "password": "12345678"
}
```

Sample successful response in JSON:

```
{
  "message": "New user created.",
  "data": {
    "username": "admin",
    "email": "admin@u.nus.edu",
    "_id": "618a76351326b76b210243a8",
    "__v": 0
  }
}
```

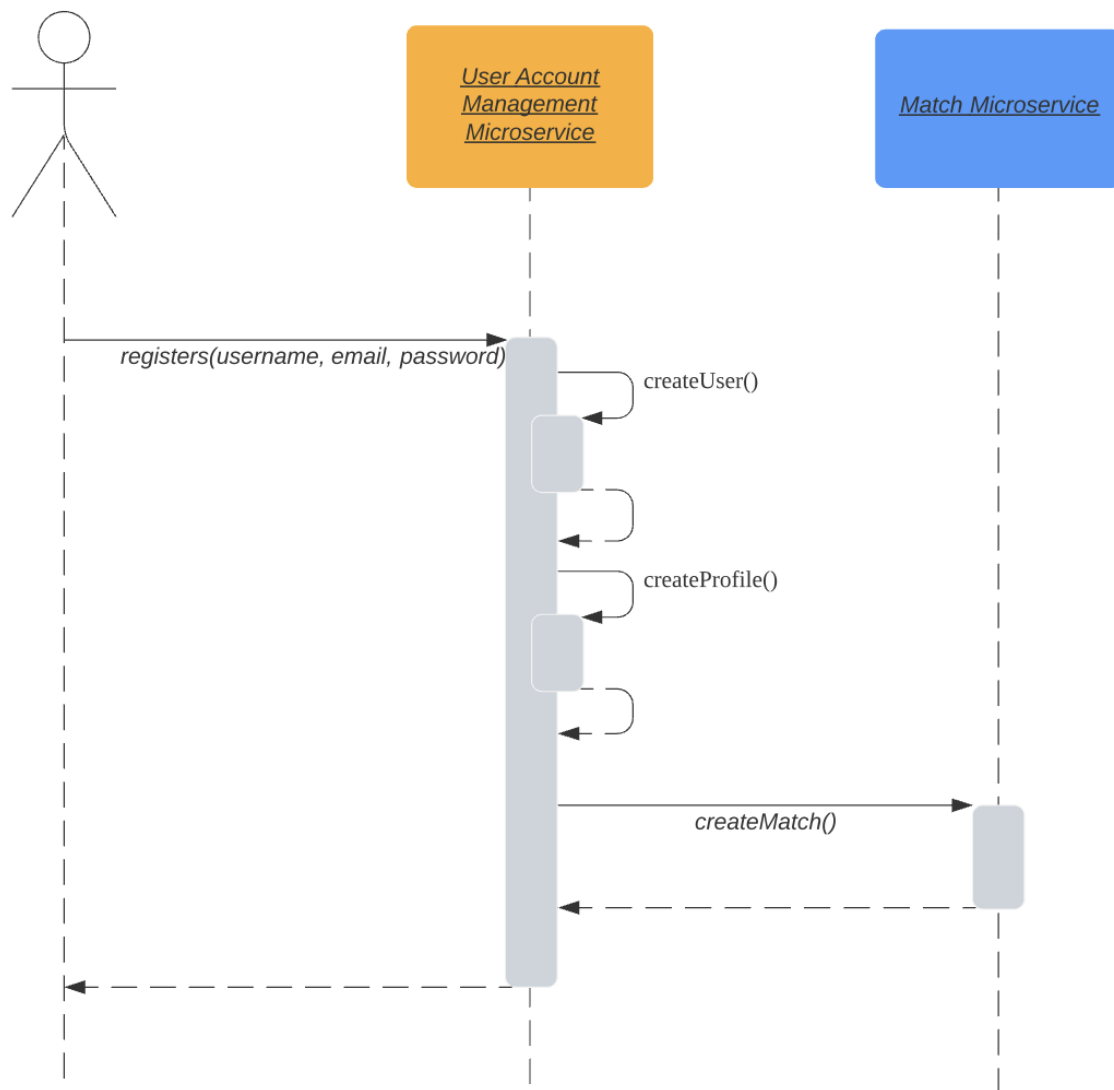


Figure: Sequence diagram demonstrating user's registration

### Login

End-point: <HOST>/api/auth/login

Request method: POST

Sample body in JSON:

```
{
  "username": "admin",
  "password": "12345678"
}
```

Sample successful response in JSON:

```
{
```

```

"token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2kljoiNjE3MmVkMTg4MzMzNDIiODc5ODhiYzk5IiwiaWF0Ijox
NjM2NDY0Mjg1LCJleHAiOiJlMzY3MjM0ODV9.HyIJybSqlKDdrGGXfww1zxvKgDO1zoYyUMTi4rgvZVs",
"user": {
  "_id": "6172ed18833349e87988bc99",
  "username": "admin",
  "email": "cs3219@u.nus.edu",
  "__v": 0
}
}

```

## **User:**

### Retrieve a user's details

End-point: <HOST>/api/user/{username}

Request method: GET

Bearer Token: Bearer <JWT>

Sample successful response in JSON:

```

{
  "message": "User admin retrieved successfully!",
  "data": {
    "_id": "6172ed18833349e87988bc99",
    "username": "admin",
    "email": "admin@u.nus.edu",
    "password": "$2a$10$iN00F5Ju4TAv3u4FNlgOPeSlicV.CD9VhnBDEhO8hfPJnm.0jPRzi"
    "__v": 0
  }
}

```

## **Profile**

### Add a new user profile

End-point: <HOST>/api/user/profile

Request method: POST

Bearer Token: Bearer <JWT>

Sample body in JSON:

```

{
  "username": "dev",
}

```

Sample successful response in JSON:

```

{

```

```

    "message": "New profile created!",
    "data": {
      "username": "dev",
      "interviews": [],
      "__v": 0
    }
  }
}

```

### Retrieve a user's profile

End-point: <HOST>/api/user/profile/{username}

Request method: GET

Bearer Token: Bearer <JWT>

Sample successful response in JSON:

```

{
  "message": "Profile of admin retrieved successfully!",
  "data": {
    "_id": "61891074f86a28f2125d56dc",
    "username": "admin",
    "interviews": [],
    "__v": 0
  }
}

```

### Add a new interview

End-point: <HOST>/api/user/profile/interview/{username}

Request method: POST

Bearer Token: Bearer <JWT>

Sample body in JSON:

```

{
  "partnerUsername": "admin",
  "question": "3 Sum",
}

```

Sample successful response in JSON:

```

{
  "message": "New interview created.",
  "data": {
    "partnerUsername": "admin",
    "question": "3 Sum",
    "_id": "618ab49a62b23db374e93bbe",
  }
}

```

```
"date": "2021-11-09T17:49:14.420Z"
}
}
```

## **Friend:**

### Retrieve a user's friends

End-point: <HOST>/api/user-friend/{username}

Request method: GET

Bearer Token: Bearer <JWT>

Sample successful response in JSON:

```
{
  "message": "Friends retrieved successfully.",
  "data": [
    {
      "_id": "618235b49797cb6897676cc9",
      "user_id": "6172ed18833349e87988bc99",
      "friend_id": "61672fcf35b12526cf0e0cc4",
      "friend_username": "Wei Yue",
      "__v": 0
    }
  ]
}
```

### Add a new friend

End-point: <HOST>/api/user-friend/user2

Request method: POST

Bearer Token: Bearer <JWT>

Sample body in JSON:

```
{
  "friend_username": "admin"
}
```

Sample successful response in JSON:

```
{
  "message": "New friend formed.",
}
```

```
"data": {  
  "user_id": "618ab38962b23db374e93bb1",  
  "friend_id": "61869a529ca99a9bae8e9959",  
  "friend_username": "test",  
  "_id": "618ab94a62b23db374e93bda",  
  "__v": "0",  
}
```



## 8.2 Match Microservice

The Match microservice stores every user's Experience Point (XP) and handles the matching of users based on their XP. It also supports updating of XP of users and

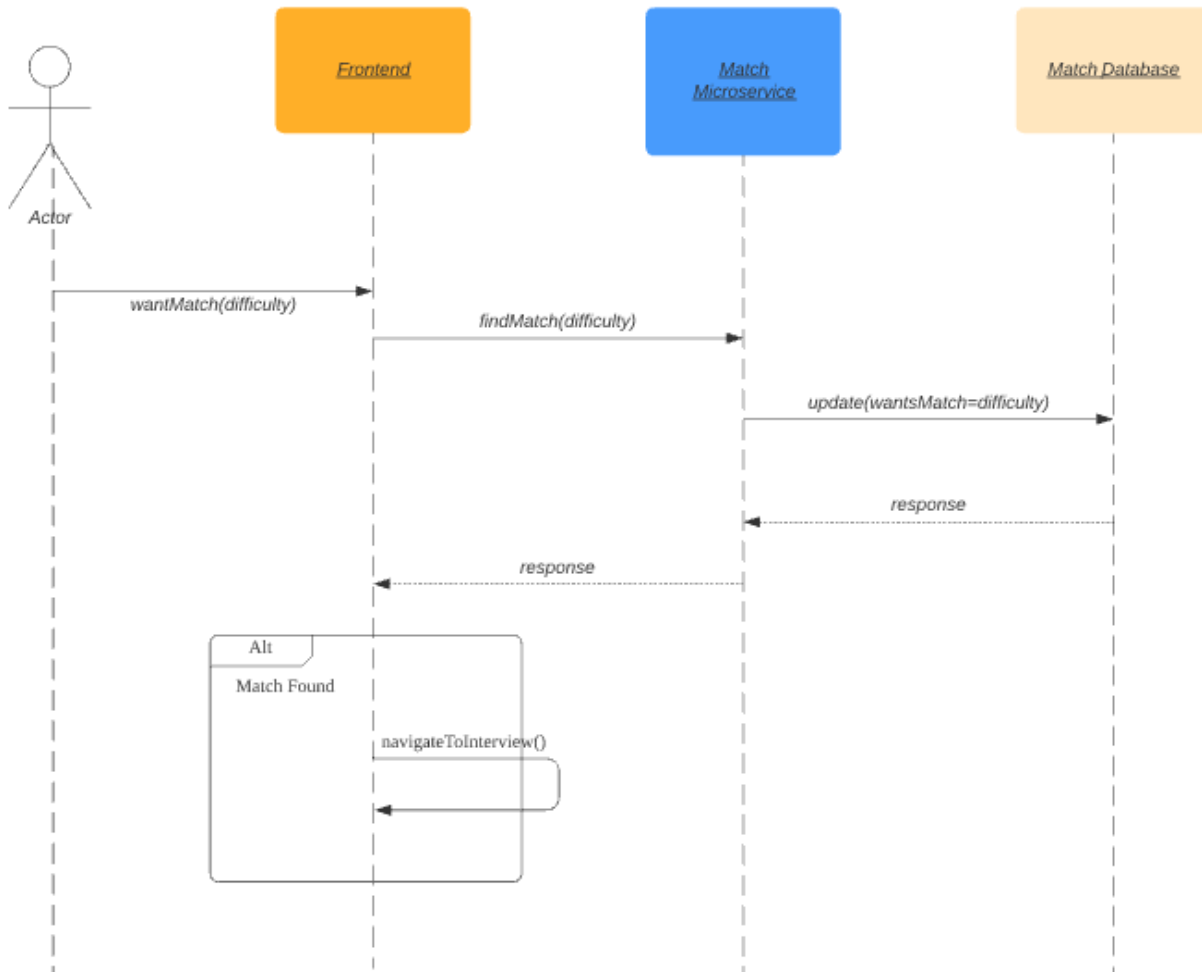


Figure: Sequence Diagram for Match Microservice

Notable Supported Endpoints:

### **Match:**

Create a new match for user (This is performed whenever a new user registers)

End-point: <HOST>/api/matches

Request method: POST

Sample body in JSON:

```
{
  "username": "admin",
  "password": "password",
  "email": "admin@u.nus.edu",
```

```
}
```

Sample successful response in JSON:

```
{
  "status": "success",
  "message": "New match created!",
  "data": {
    "_id": "618b361e6e3733e755e84c39",
    "username": "admin",
    "xp": 0,
    "match": null,
    "questionTitle": null,
    "questionDifficulty": null,
    "isOnline": true,
    "__v": 0
  }
}
```

#### Request for a new match

End-point: <HOST>/api/matches

Request method: PUT

Sample body in JSON:

```
{
  "username": "admin",
  "questionTitle": "3 Sum",
  "questionDifficulty": "Medium"
}
```

Sample successful response in JSON:

```
{
  "status": "Success",
  "message": "Set current user questionTitle and questionDifficulty status successfully",
  "data": "Current user: admin"
}
```

#### Delete a user's current match (sets the user's match to null)

End-point: <HOST>/api/matches

Request method: DELETE

Sample body in JSON:

```
{
  "username": "admin"
}
```

Sample successful response in JSON:

```
{
  "status": "Success",
  "message": "Deleted user match",
  "data": {
    "_id": "61891198f86a28f2125d56de",
    "username": "admin",
    "xp": 51197,
    "match": null,
    "questionTitle": null,
    "questionDifficulty": null,
    "isOnline": true,
    "__v": 0
  }
}
```

Updates a user's match details (e.g., updating xp, questionDifficulty, etc.)

End-point: <HOST>/api/matches/match

Request method: PUT

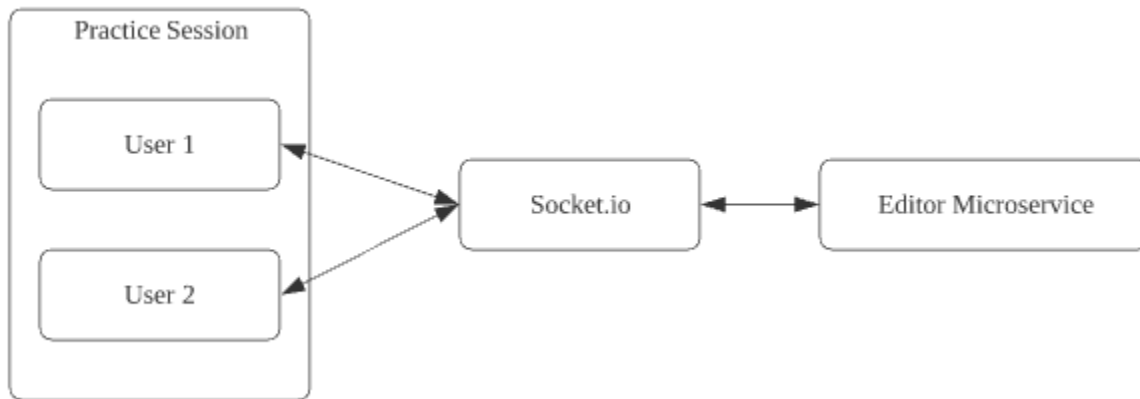
Sample body in JSON:

```
{
  "username": "admin",
  "isOnline": true,
  "xp": 100
}
```

Sample successful response in JSON:

```
{
  "status": "Success",
  "message": "Save current user status successfully",
  "data": {
    "_id": "61891198f86a28f2125d56de",
    "username": "admin",
    "xp": 100,
    "match": null,
    "questionTitle": "3 Sum",
    "questionDifficulty": "Medium",
    "isOnline": true,
    "__v": 0
  }
}
```

### 8.3 Editor Microservice



Pub-sub Pattern for Editor Microservice

The Editor Microservices uses Socket.io to enable real time communication between two users in a practice session. The communication channel for each practice session is identified by the `sessionId`, which is generated through concatenating the usernames of two users, starting with the smaller username.

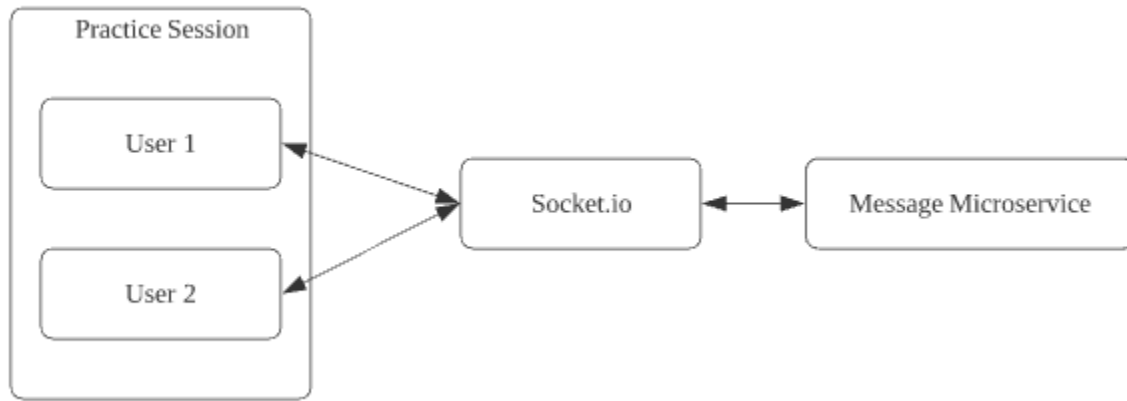
For example, “Alice-Bob” would be the `sessionId` between two users with username Alice and username Bob. This is because the string comparison is “Alice” < “Bob”. Since each username is guaranteed to be unique, we will guarantee a unique `sessionId` between two different users.

On the client, it listens to the socket using the `sessionId` generated above. Since the `sessionId` is unique, no other third party can listen to this channel.

The communication uses pub-sub message design pattern to send the correct data to the relevant parties shown in the diagram above. If User 1 has username “Alice” and User 2 has username “Bob”, then both User1 and User 2 will listen to “Alice-Bob”. The Editor microservice act as publisher to “Alice-Bob” and subscriber to “send-changes”. The Client socket will emit to “send-changes” so that the Server socket will receive this signal and redirect it to the “receive-changes” channel. The Client socket will listen to “receive-changes” and update the editor accordingly.

To allow the contents of the editor to be persistent after each practise session, the contents are being stored in a database with the `sessionId`. Whenever the same users are matched again (and subsequently routed to the same practise room), the contents of the editor will be fetched. If the users are matched for the same time, then a new entry will be created in the database. For example, if Java is used, the Main class and public static void main method can be kept for convenience sake. This feature is similar to other industry leading collaborative code editors like CoderPad and CodeBunk which many companies use.

## 8.4 Message Microservice



Pub-sub Pattern for MessageMicroservice

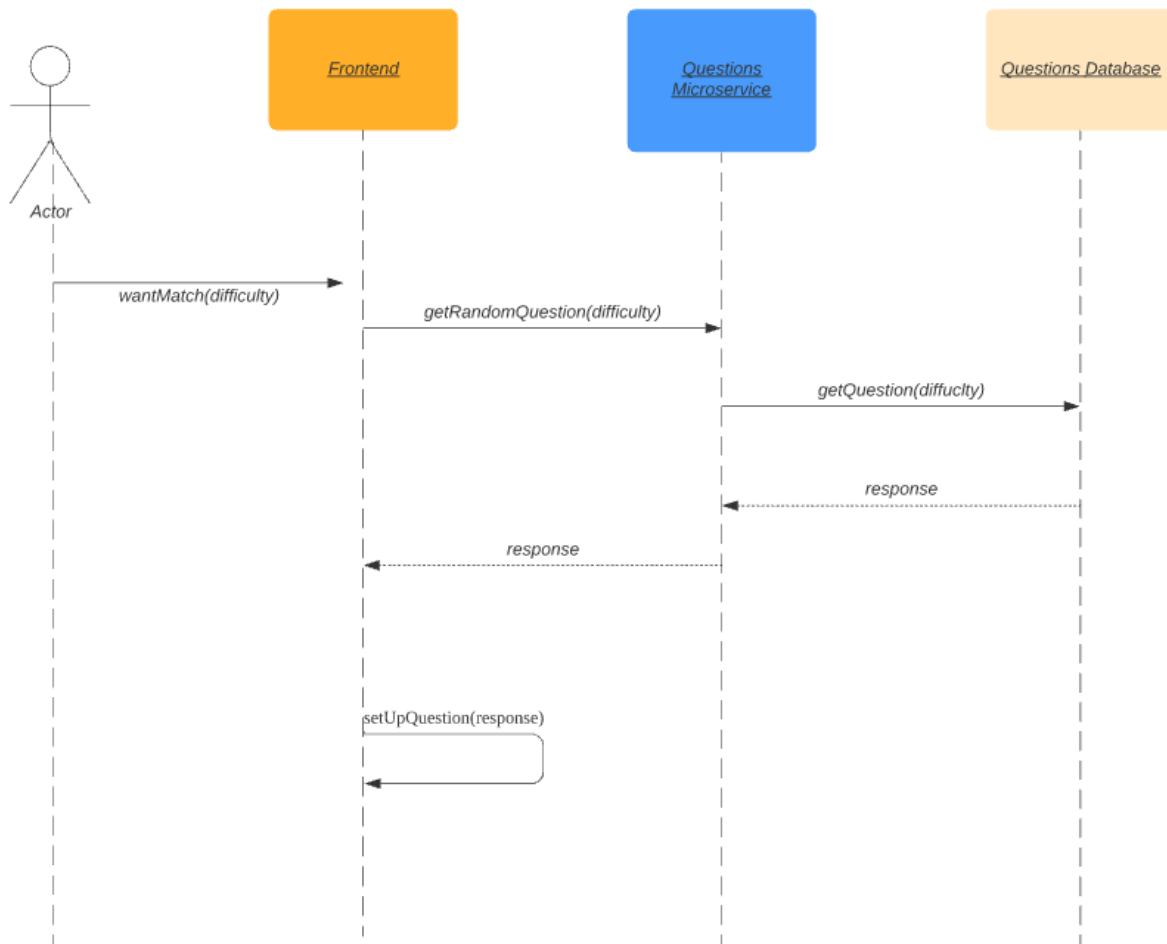
Similar to Editor Microservice, the Message Microservice uses Socket.io to enable real time communication between two users in a practice session. The communication channel for each practice session is identified by the `sessionId`, which is generated through concatenating the usernames of two users, starting with the smaller username.

For example, “Alice-Bob” would be the `sessionId` between two users with username Alice and username Bob. This is because the string comparison is “Alice” < “Bob”. Since each username is guaranteed to be unique, we will guarantee a unique `sessionId` between two different users.

On the client, it listens to the socket using the `sessionId` generated above. Since the `sessionId` is unique, no other third party can listen to this channel.

The communication uses pub-sub message design pattern to send the correct data to the relevant parties shown in the diagram above. If User 1 has username “Alice” and User 2 has username “Bob”, then both User1 and User 2 will listen to “Alice-Bob”. The Message microservice acts as publisher to “Alice-Bob” and subscriber to “new-message”. The Client socket will emit a “new-message” so that the Server socket will receive this signal and redirect it to “Alice-Bob” channel. The Client socket will listen to “Alice-Bob” and update the chat component with the latest messages. No third user should be able to access this “Alice-Bob” unless manual script and connection is done.

## 8.5 Questions Microservice



Sample Sequence Diagram for Questions Microservice

A standalone microservice from the rest, the Questions microservice merely serves only 1 purpose - as a repository for questions with respect to user requirements. In our case, this microservice possesses its own database as abided by the database per service pattern.

Notable supported endpoints:

Get a question by title

End-point: <HOST>/api/questions/{title}

Request method: GET

Sample successful response in JSON:

```
{
  "message": "Question: Two Sum retrieved successfully!",
  "data": {
    "username": "617ea53b57a9331e50e7ed59",
    "title": "Two Sum",
    "description": "Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.\nYou may assume that each input would have exactly one solution, and you may not use the same element twice.\nYou can return the answer in any order.",
    "difficulty": "Easy",
    "image": "",
    "testcases": [
      {
        "input": "nums = [2,7,11,15], target = 9",
        "output": "[0,1]",
        "_id": "617a971b5b193718ed6a1ddc"
      },
      {
        "input": "nums = [3,2,4], target = 6",
        "output": "[1,2]"
        "_id": "617a971b5b193718ed6a1ddd"
      }
    ]
  },
  "__v": 0
}
```

Get a question randomly by difficulty

End-point: <HOST>/api/questions/difficulty/{level}

Request method: GET

Sample successful response in JSON:

```
{
  "message": "Question: Three Sum retrieved successfully!",
```

```

"data": {
  "username": "617ea53b57a9331e50e7ed59",
  "title": "Three Sum",
  "description": "Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k,
and j != k, and nums[i] + nums[j] + nums[k] == 0.\nNotice that the solution set must not contain duplicate triplets.",
  "difficulty": "Medium",
  "image": "",
  "testcases": [
    {
      "input": "nums = [-1,0,1,2,-1,-4]",
      "output": "[[-1,-1,2],[-1,0,1]]",
      "_id": "617a971b5b193718ed6acddc"
    },
  ]
  "__v": 0
}
}

```

### Get a question randomly

End-point: <HOST>/api/random-question

Request method: GET

Sample successful response in JSON:

```

{
  "message": "Question: Three Sum retrieved successfully!",
  "data": {
    "username": "617ea53b57a9331e50e7ed59",
    "title": "Three Sum",
    "description": "Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k,
and j != k, and nums[i] + nums[j] + nums[k] == 0.\nNotice that the solution set must not contain duplicate triplets.",
    "difficulty": "Medium",
    "image": "",
    "testcases": [
      {
        "input": "nums = [-1,0,1,2,-1,-4]",
        "output": "[[-1,-1,2],[-1,0,1]]",
        "_id": "617a971b5b193718ed6acddc"
      },
    ]
    "__v": 0
  }
}

```



}  
}

## 9. Frontend

- React framework
- React-Bootstrap
- Material UI
- CSS

Our team has used the **React** framework to build our application. React is an open-source JavaScript framework that is used for building fast, scalable user interfaces simply for both single-page applications & progressive web apps. Additionally with the addition of React Hooks, we could leverage on powerful hooks and ES6 higher order functions/syntax to create our own web applications with easy state management & data passing between web components.

The fundamental frontend library we worked with was **React-Bootstrap**. Our components, such as Modals, Buttons and Alerts are integrated using React-Bootstrap. React-Bootstrap has many customisable components as well template css styles that allows us to reduce boilerplate codes that we need to create Buttons or Form elements as well as saving us the need to write huge chunks of CSS code to style the components properly, whilst ensuring a decent level of responsiveness too.

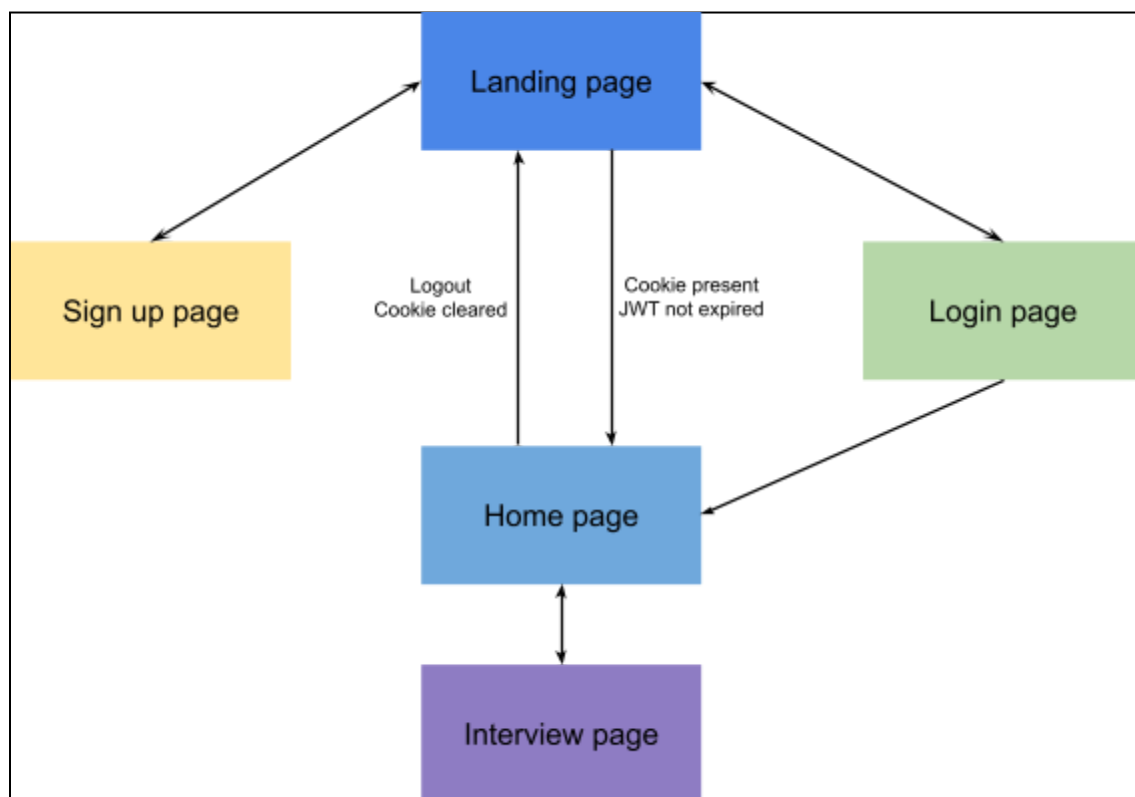
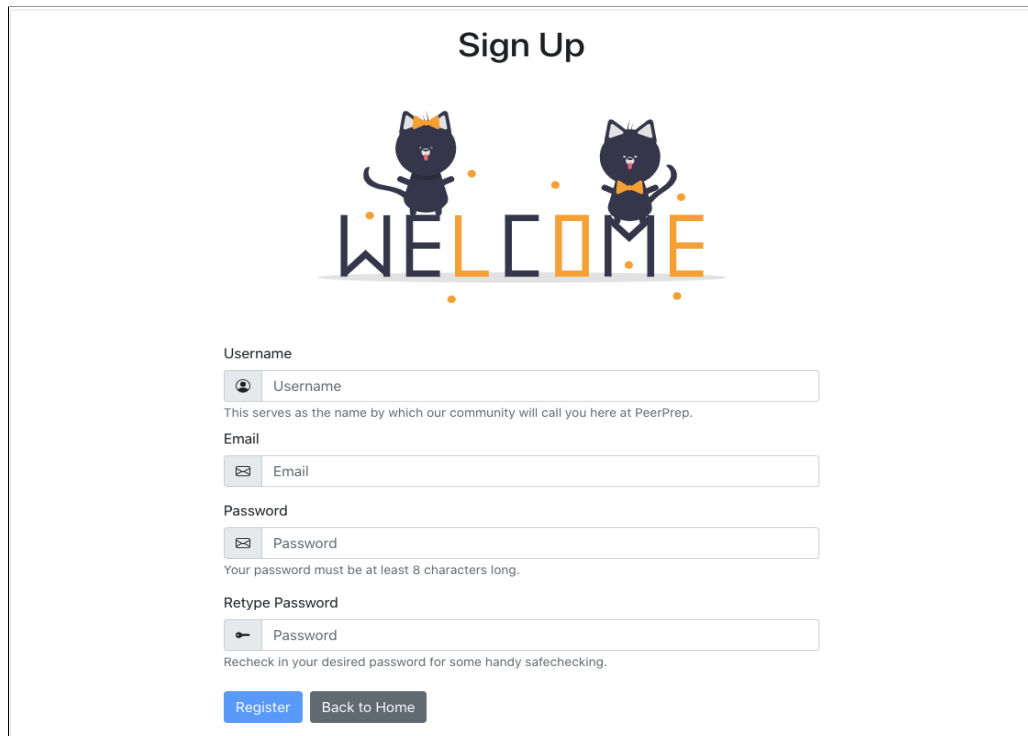


Figure: Frontend UI pages and flow

The App component in our React code acts as the **controller** class which contains routes to 3 main pages/components of our application: Home Page, Interview Page and Login Page, Sign Up page. These 3 main pages/components also rely on smaller, modularised components in the components folder to build onto its functionality. On a side note, the Landing Page serves a simple role in prefacing the application to potential users, as well as explaining the purpose of our application.

## 10. Application Screenshots

### 10. 1 User Authentication



The 'Sign Up' form features a header with the title 'Sign Up' and a decorative illustration of two black cats flanking the word 'WELCOME' in large, colorful letters. Below the illustration, the form contains four input fields: 'Username', 'Email', 'Password', and 'Retype Password'. Each field has a small icon to its left (person, envelope, and key respectively). The 'Username' field is followed by a descriptive sentence. The 'Password' field is followed by a requirement note. At the bottom, there are two buttons: 'Register' (blue) and 'Back to Home' (grey).

Sign Up

WELCOME

Username

This serves as the name by which our community will call you here at PeerPrep.

Email

Password

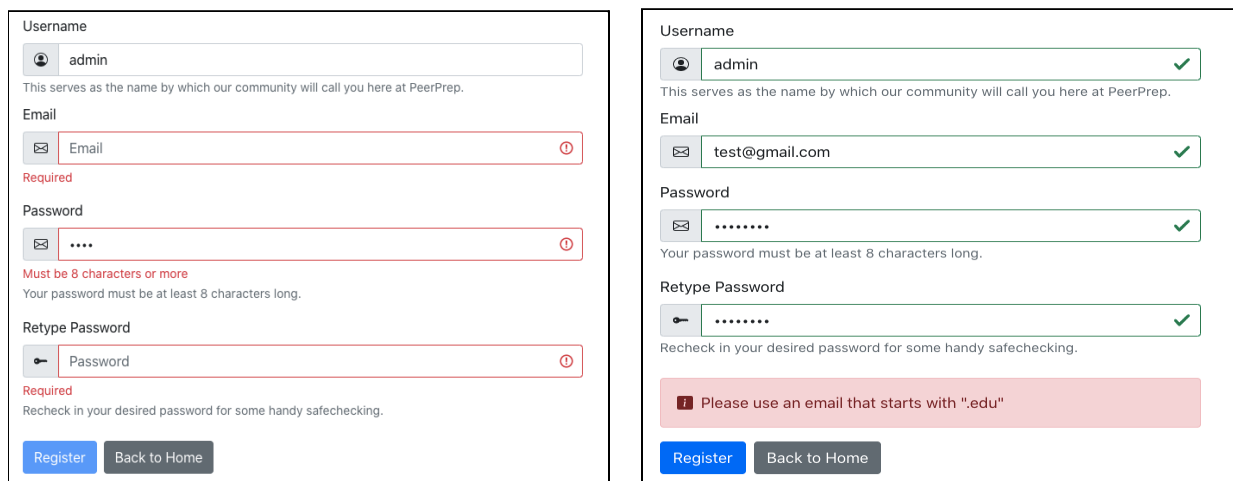
Your password must be at least 8 characters long.

Retype Password

Recheck in your desired password for some handy safechecking.

[Register](#) [Back to Home](#)

Figure: Registration page



This figure shows two side-by-side versions of the registration form to illustrate validation states. The left version shows invalid inputs: 'admin' for Username, 'Email' for Email, '....' for Password, and 'Password' for Retype Password. Each field has a red border and a red error icon. The right version shows valid inputs: 'admin' for Username, 'test@gmail.com' for Email, and '.....' for both Password and Retype Password. Each field has a green border and a green checkmark icon. A red error message 'Please use an email that starts with ".edu"' is displayed below the Email field in the right version. Both versions have 'Register' and 'Back to Home' buttons at the bottom.

Username

This serves as the name by which our community will call you here at PeerPrep.

Email

Required

Password

Must be 8 characters or more

Your password must be at least 8 characters long.

Retype Password

Required

Recheck in your desired password for some handy safechecking.

[Register](#) [Back to Home](#)

Username

This serves as the name by which our community will call you here at PeerPrep.

Email

Password

Your password must be at least 8 characters long.

Retype Password

Recheck in your desired password for some handy safechecking.

Please use an email that starts with ".edu"

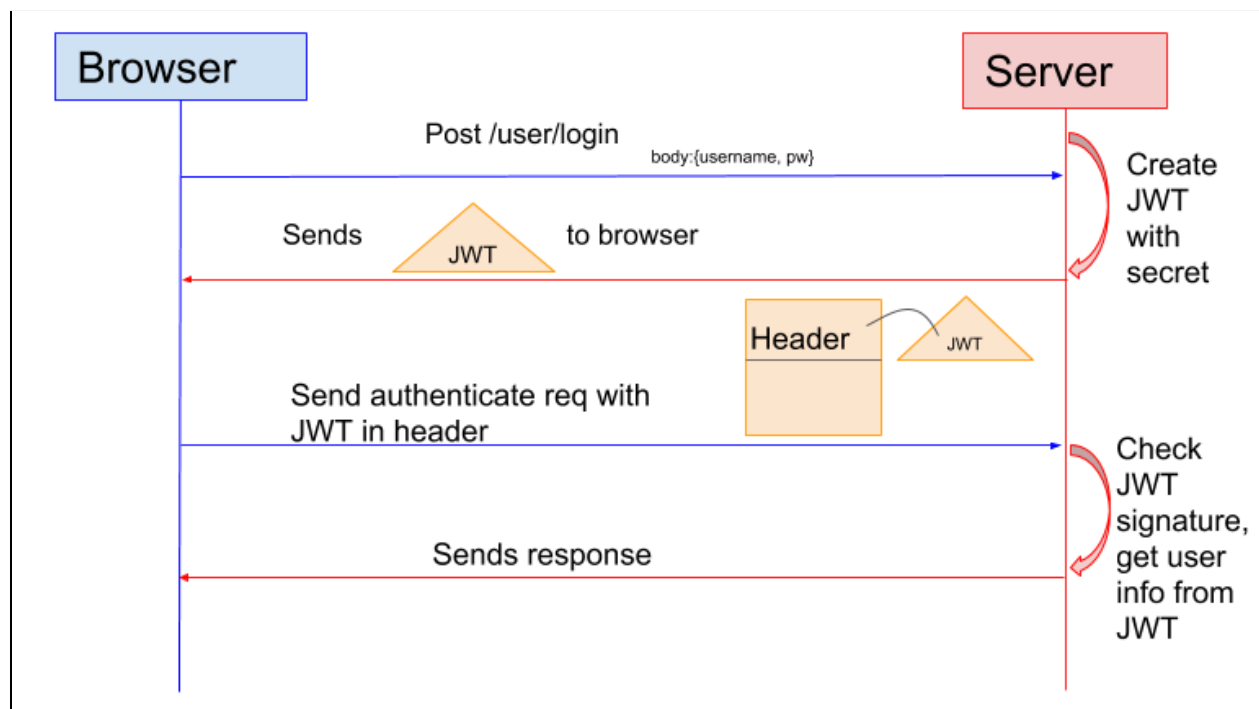
[Register](#) [Back to Home](#)

Figure: Validations

Registration page simply consists of an email, **unique** username & password for creation of account. During login, only the username & password are needed. Validations are also included to catch any user mistakes when parsing input (i.e users can't post empty data, password of at least 8 characters, ...).

Authentication is done with a simple JWT token system. Noting that security is NOT one of our main targets in terms of NFRs, we aim to offer a minimum level of security whilst ensuring a **high ease of use** for our users as part of the usability NFR.

In that regard, we use a simple cookie to store the JWT token and verify requests made to the backend microservices - each token expires in 2 days time. The diagram below offers a quick high level illustration of this.



Figure<sup>2</sup>: JWT Cookie based authentication

## 10.2 User Home Page

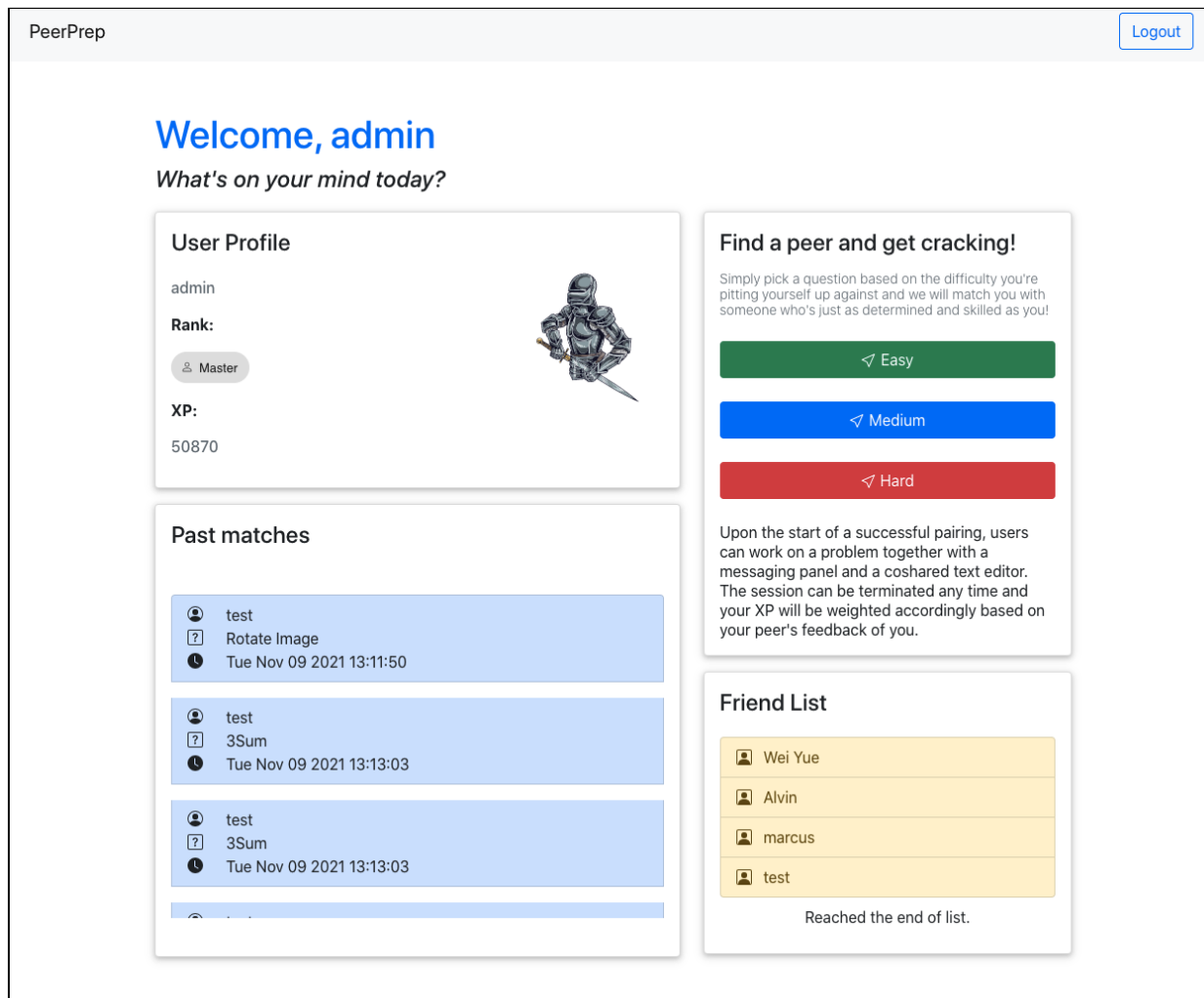
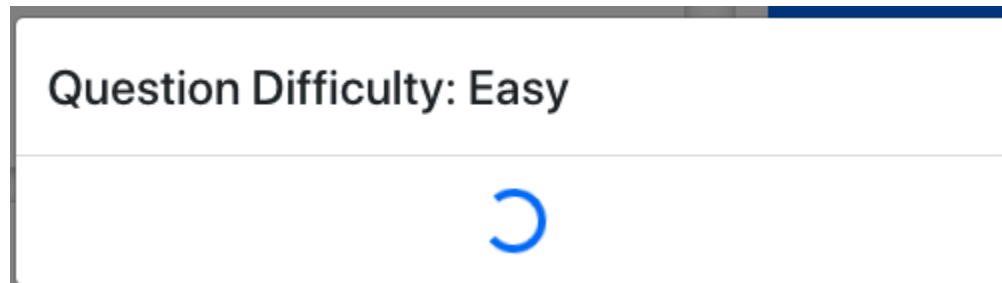


Figure: User Home page

## 10.3 Matching System



When attempting to find a new peer to match up with and tackle a question of a selected difficulty from **Easy**, **Medium** & **Hard**, the user will be directly prompted with the matching modal as seen above. The loader indicates that the match microservice is currently attempting to find a peer for the current user. To cease the matching service, the user simply needs anywhere outside the modal, within the browser screen.

Users who are within 100 xp of each other will be paired accordingly. For a quick high level understanding of the sequential flow, do refer to the UML activity diagram below for the overview of the matching flow.

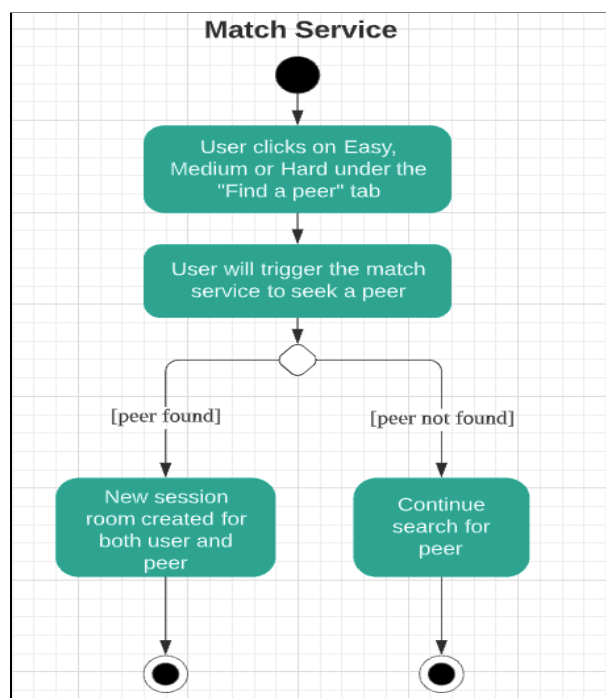


Figure: Activity diagram for Match service

## 10.4 Interview

Once the match has been done successfully, the pair of users will be routed to the Interview page, whereby they will be assigned a random question based on the difficulty selected during the match. The page consists of a **question tab**, real-time textbox editor (mimicked the notepad style to further closely match up to the actual technical or whiteboarding interviews), as well as a real-time messaging functionality as well. (Of course the editor is empty on initialisation.)

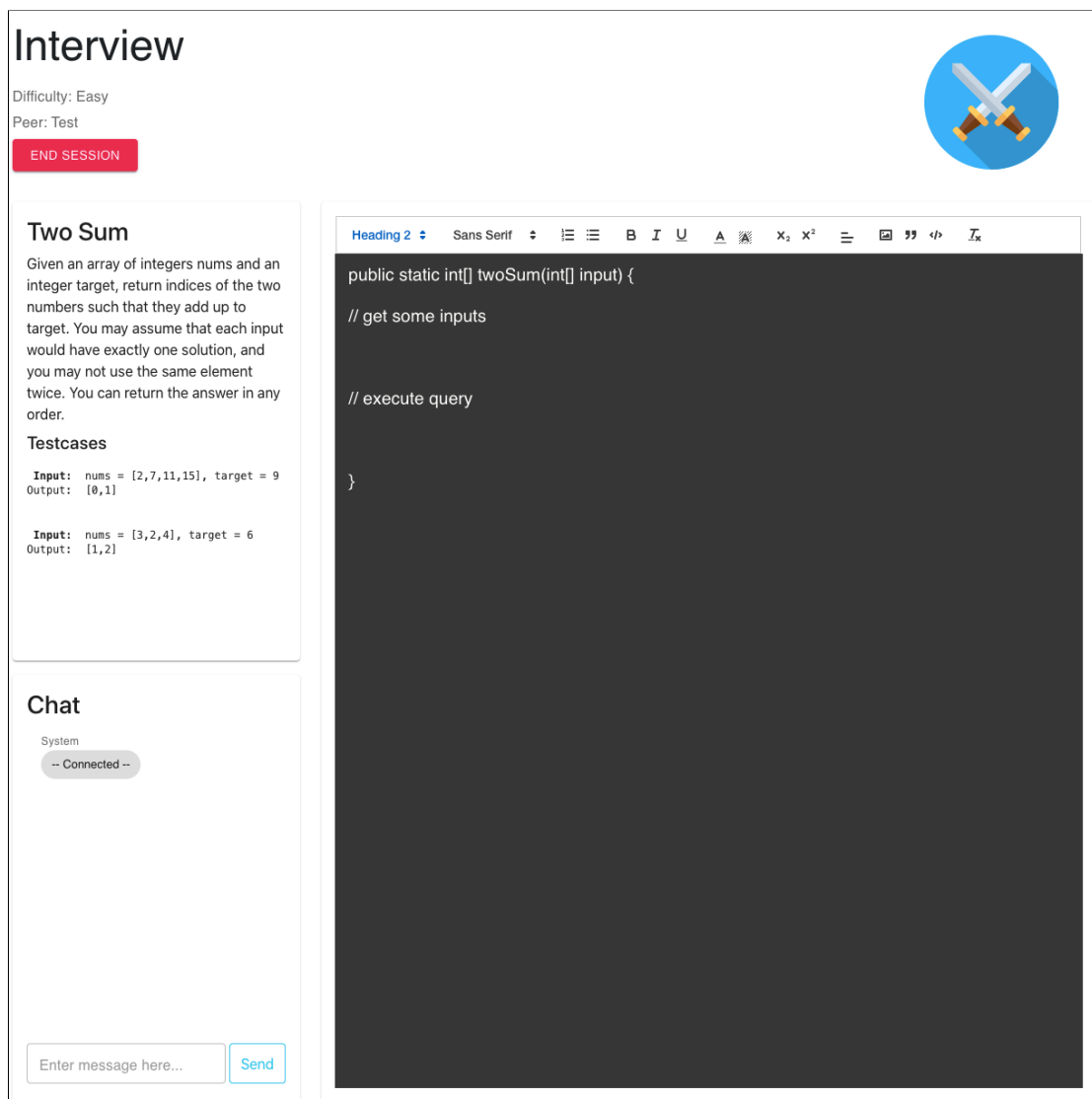


Figure: Sample Interview page for User **Admin** in action



# Interview

Difficulty: Easy  
Peer: Admin  

END SESSION

## Two Sum

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target. You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order.

### Testcases

**Input:** nums = [2,7,11,15], target = 9  
**Output:** [0,1]

**Input:** nums = [3,2,4], target = 6  
**Output:** [1,2]

## Chat

System  
-- Connected --

Enter message here...

Send

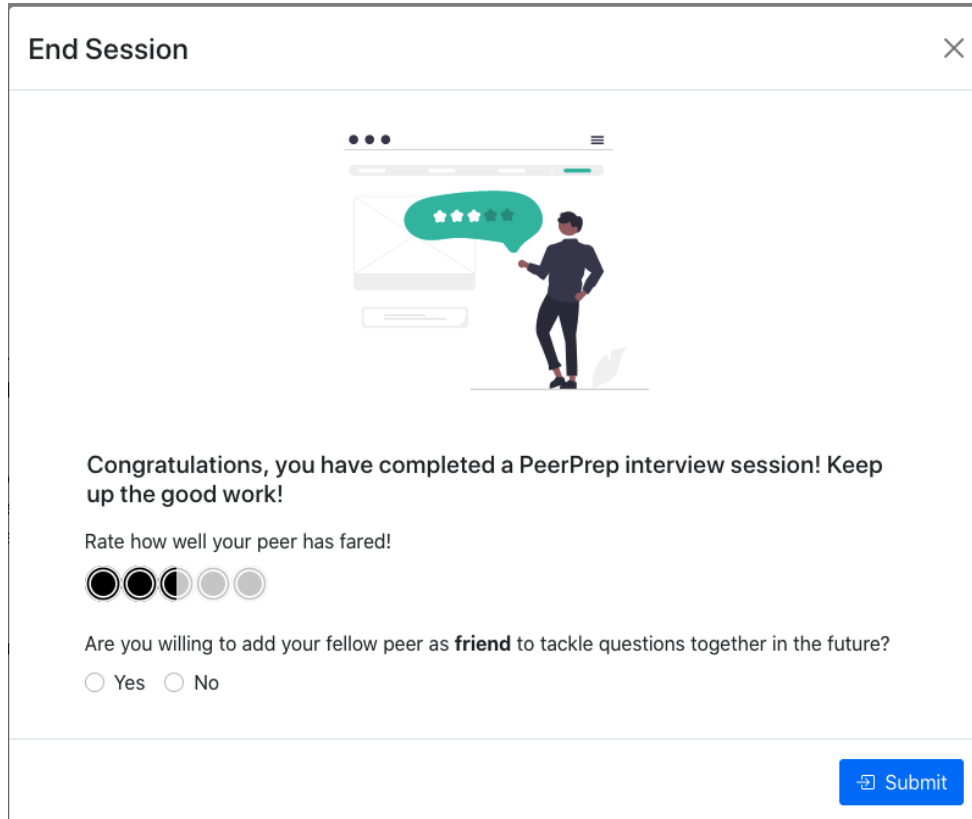
Heading 2 Sans Serif B I U A X<sub>2</sub> X<sup>2</sup> [Icons]

```
public static int[] twoSum(int[] input) {  
  
    // get some inputs  
  
  
    // execute query  
  
  
}
```

Figure: Sample Interview page for User **Test** in action

## 10.5 Feedback session

When a user decides to terminate the session, he can simply choose to click on 'Terminate session' button on the top of the Interview page, on which he will be prompted with this modal to push in his feedback.

A modal window titled "End Session" with a close button (X) in the top right corner. The modal contains an illustration of a person in a suit standing next to a laptop, with a speech bubble containing five stars above the laptop. Below the illustration, the text reads: "Congratulations, you have completed a PeerPrep interview session! Keep up the good work!". This is followed by the prompt "Rate how well your peer has fared!" and a rating scale of five circles, where the first two are filled and the third is half-filled. Below the rating scale is the question "Are you willing to add your fellow peer as **friend** to tackle questions together in the future?" with radio buttons for "Yes" and "No". A blue "Submit" button is located at the bottom right of the modal.

End Session

Congratulations, you have completed a PeerPrep interview session! Keep up the good work!

Rate how well your peer has fared!

☒ ☒ ☒ ☐ ☐

Are you willing to add your fellow peer as **friend** to tackle questions together in the future?

☐ Yes ☐ No

Submit

Figure: Sample Modal when user wants to end session

This consists of 2 possible inputs for the user:

1. Rating: Rate the experience of the session, as well as the skill of the other user. Higher rating = more xp for the peer, **default** at 2.5 out of 5.
2. Option to add friend: The user can choose to add the current peer as a friend to directly seek interviews with in the future, **default** at false.

The XP awarded are weighted according to the rating given to the peer, as well as a randomised integer added to the rating based on the difficulty of the question selected.

The computation of the XP can be observed below in the code snippet below.  
Higher rating = more XP, more difficult questions = more *randomised* XP awarded

```
// Assign XP randomised by difficulty level
const assignXp = (difficulty, rating) => {
  var xp = 0;
  if (difficulty === "Easy") {
    xp = rating * 20 + Math.floor(Math.random() * 30) + 1
  } else if (difficulty === "Medium") {
    xp = rating * 30 + Math.floor(Math.random() * 40) + 1
  } else {
    xp = rating * 50 + Math.floor(Math.random() * 50) + 1
  }
  return xp;
}
```

Figure: Calculation of XP

## 10.6 Friends List

As mentioned earlier, PeerPrep also supports a social network element in the form of friends, where users can meet peers and add friends upon the end of a session, after which they can simply click on a peer's name to set a session upon clicking their name tab on the list.

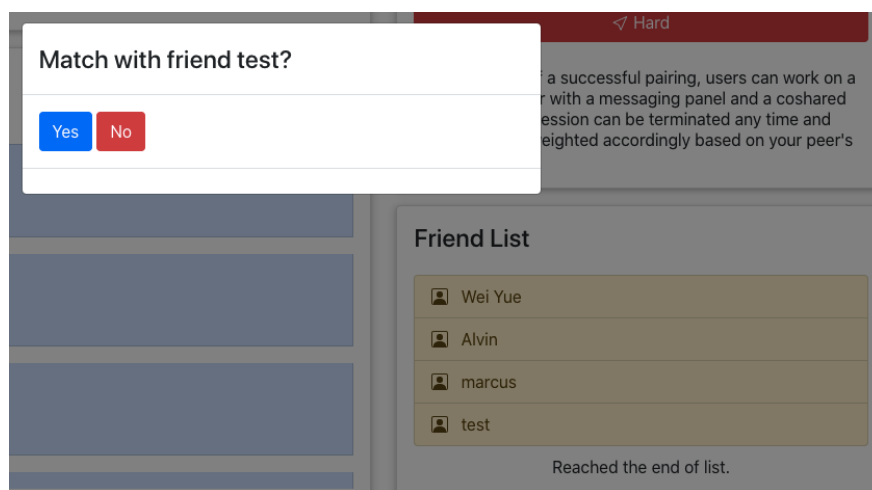


Figure: Upon a simple click on the user **test** in the friends list of user **admin**

Suppose the user *admin* decides to send an interview request to his friend, *test*. Now, user *test* receives a request from *admin*.

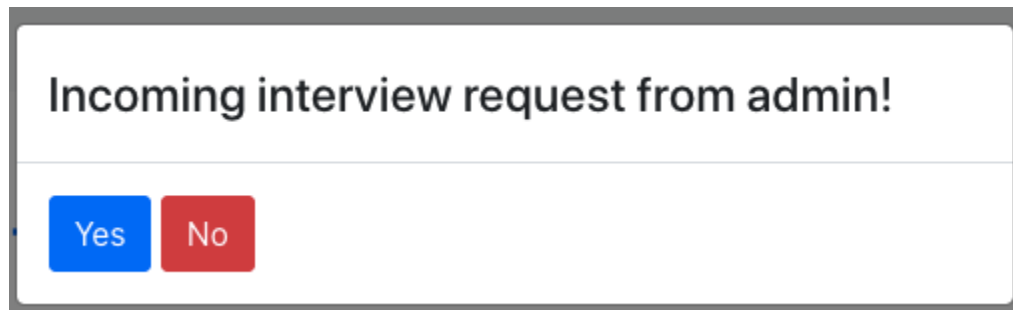


Figure: Interview request from friend *admin* to user

User *test* now receives a prompt from *admin* to accept an interview request. Upon acceptance, both users will be slotted into a similar Interview page as seen earlier, but with a random question of **any** difficulty. This was done to raise the '**game**' factor of Peerprep, whereby users can choose to interview with known peers for an unknown difficulty of a question, or they can choose a difficulty but have an unknown peer for the interview.

If user *test* (or any recipient for that matter) **declines** to the request within 30s, the request modal will be terminated and the requester *admin* will be presented with the toast below on the top right hand corner of the screen.

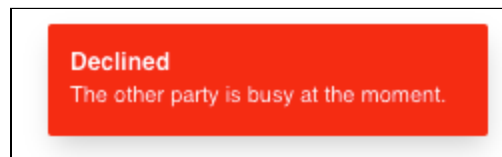


Figure: Rejection toast for *admin*

Otherwise, if *test* were to fail to respond within 30 seconds, the request modal will automatically close on both user *admin* and user *test*.

Likewise, here's the high level overview of the flow involved with the friend list feature as seen here

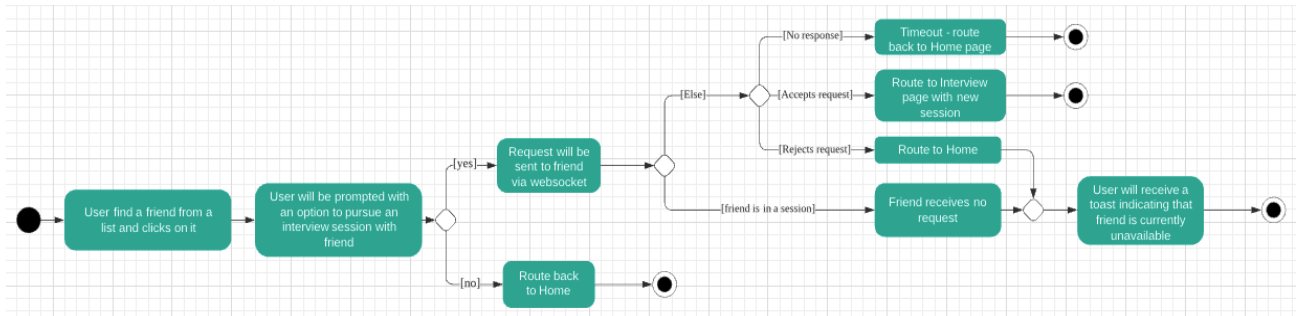


Figure: Activity diagram for Friends

## 11. Remarks

### 11.1 Challenges Faced

- Faced challenges on the frontend logic involved in data handling and matching of peers real-time
- Backend structure in storage of data and determining what's necessary and what unnecessary
- Faced challenges on the back end trying to update the match partner of two matched users
- Deployment logic involved in .yaml configuration for GKE and docker
- Learning the new technology and implementing Pub-sub pattern using Socket.io

### 11.2 Potential Extension Features

#### 1. Audio & Video communication

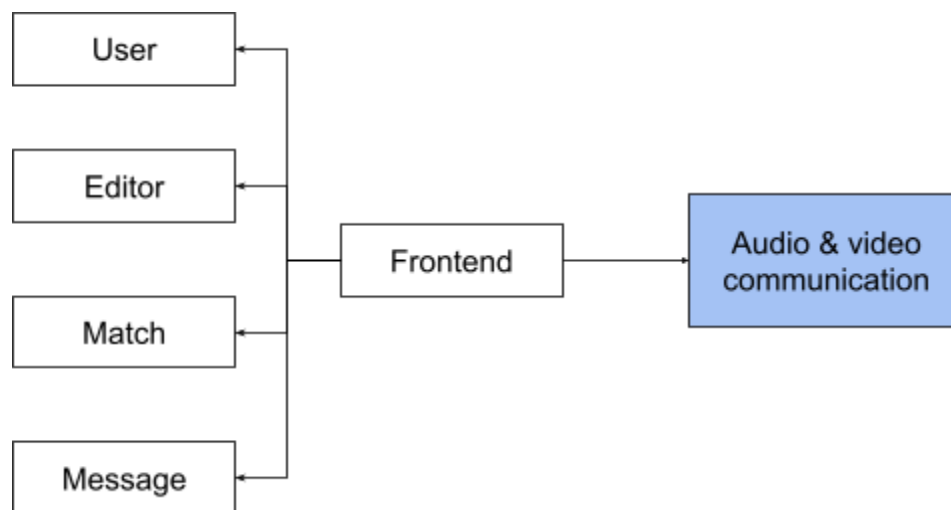


Figure: High-level architecture diagram with audio & video communication feature

A useful feature to insert into Peerprep would be audio & video communication. This aids in facilitating a far more robust technical interview session as a normal technical interview doesn't normally require any form of worded communication but rather emphasises on precise & clear verbal communication. Through the use of microservices architecture, we can extend the current application to include an audio, video communication service using more performant languages like Java, C++. We can also make use of React WebRTC libraries to implement this realtime communication feature.

## 2. Code completion features

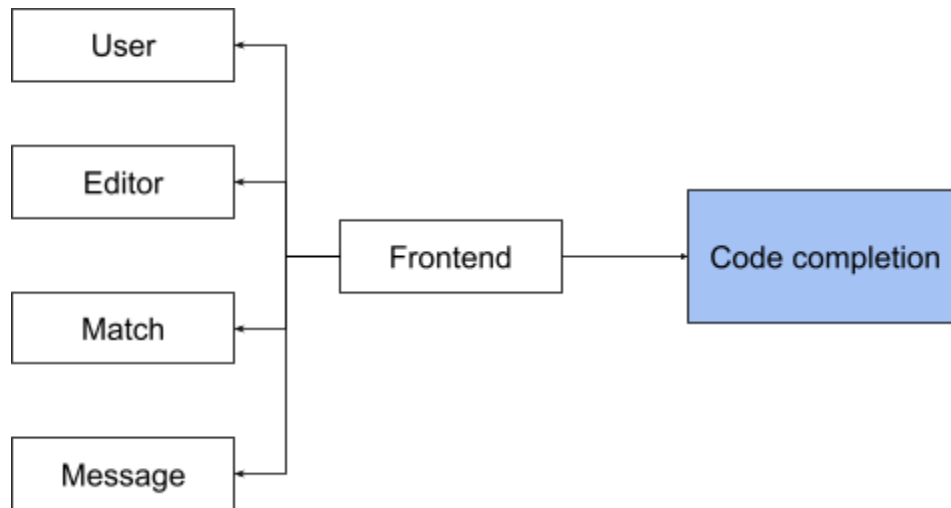


Figure: High-level architecture diagram with code completion feature

Another useful feature would be to allow for auto-completion of code and running against the test cases. This would simulate a similar usage like LeetCode, HackerRank etc, so that users will know whether their code is correct against the test cases. Even though minor syntax errors are rather negligible in actual technical interviews, practicing with correct syntax and fully compilable code would be impressive. Peerprep will aim to

## **11.3 Reflections**

In a nutshell, this project turned out to be harder and more time-consuming than expected as we were given exposure to the entire SDLC in a short span of 6 weeks, all the way from requirement gathering, documentation, prototyping, coding, testing and finally deployment.

Nonetheless, it was a fulfilling one and one in which we would have taken more seriously with enough time, as we faced issues on the deployment end on the tail-end of the project and hence had to burn a couple of midnights in resolving these pertinent issues.

## References:

Figures/references used:

1. Figure<sup>1</sup>:  
<https://measuringu.com/sus/>
2. Figure<sup>2</sup>:  
<https://sherryhsu.medium.com/session-vs-token-based-authentication-11a6c5ac45e4>
3. CS3219 AY2021/22 Semester 1 Lecture 1 notes



## Appendix A: Sample meeting minutes

**Date:** 24/09/2021

**Venue:** Zoom

**Date:** 1900

Present: Rishi, Alvin, Marcus, Wei Yue

Agenda: Weekly check in

1. Decide on future project management schedule: Agile-scrum
2. Duration of sprint: 1 week
3. Decide on architecture
4. Allocation of tasks

Index	Description	By	Due date
1	Set up frontend code and landing page	Rishi	01/10/2021
2	Flesh out basic level of user microservice	Wei yue	01/10/2021
3	Set up CI/CD tools and cloud database	Marcus	01/10/2021
4	Flesh out basic level of match microservice	Alvin	01/10/2021

The meeting was completed at 8.30 pm. These documents will be disseminated and adopted if there are no amendments reported in the next 2 days.

**Crafted by:**

Rishi

**Vetted by:**

Marcus