CS3219 Project Report (Peerprep)

Group 11

Group Members:

GOH JUN WEI (A0201970M)

LEE WEI YI, SAMUEL (A0202032H)

LOW EN HAO (A0200239U)

MURUGESAN KARTHIKA (A0191135L)

Code Repository URL:

https://github.com/CS3219-SE-Principles-and-Patterns/cs3219-project-ay2122-2122-s1-g11

Table of Contents

1. Project Background

1.1 Project Description

Peerprep is an online code collaboration platform where users can collaborate to solve programming questions of varying difficulties. Peer prep features a landing page where users can learn more about the functionalities we have to offer. Users can register and attempt programming questions by choosing various difficulties and categories through the platform. Users who have chosen the same question difficulty and category will be matched together and attempt to solve the question through our built-in text editor that supports linting of different programming languages to suit the users' needs.

1.2 Purpose of Project

Peerprep is designed to allow programmers of various proficiency to make use of our platform to improve their technical skills through code collaboration. Users will be able to write and discuss code on our platform with ease which allows them to improve their problem solving and communication skills, both of which are key to a successful tech interview.

2. Project Contributions

The below table summarizes the individual contributions towards this project.

Name	Project Contributions
GOH JUN WEI	**Technical Contribution:** - Worked on Frontend code for user matching, web page flow, login, register and forgot password page - Worked on authentication of logged in user - Created matching service (Microservice) for the backend - Worked on Redis for matching service **Non-Technical Contributions:** - Worked on different parts of the report and presentation slides - Regularly review PR for group members - Regularly ensure that the project is on the right track
LEE WEI YI, SAMUEL	**Technical Contribution:** - Worked on Frontend code for code collaboration with a text editor and chat system - Created collaboration service (Microservice) for the backend - Kubernetes and Docker deployment to AWS for frontend, collaboration, matching and question service with API gateway - CI/CD pipeline to AWS using GitHub Action **Non-Technical Contributions:** - Worked on different parts of the report and presentation slides - Regularly review PR for group members - Testing of functionality after deployment to AWS
LOW EN HAO	**Technical Contribution:** - Worked on Frontend code for landing page, question selection page, and navigation bar - Created question service (Microservice) for the backend - Created test cases for question service - Created Postgres database for question service **Non-Technical Contributions:** - Worked on different parts of the report and presentation slides - Regularly helps to write meeting discussion notes - Research on various programming questions for the question service database

MURUGESAN KARTHIKA	**Technical Contribution:** - Created user service (Microservice) for the backend - Created test cases for user service - Created Postgres database for user service - Kubernetes and Docker deployment for matching and user services **Non-Technical Contributions:** - Worked on different parts of the report and presentation slides - Regularly review PR for group members - Regularly ensure that the project is on the right track

3. Prioritized Requirement List

After various discussions, our group has come up with the updated prioritized requirements.

Priority (Color Coded)	Meaning
High	Essential Feature Required for MVP
Medium	Good To Have Feature
Low	Bonus / Extension

3.1 Functional Requirements

3.1.1 Authentication

Identifier	Description	Priority
F1.1	Users should be able to log in with email and password into the system.	High
F1.2	Users should be able to sign up with a username, email address, and password.	High
F1.3	Users should be able to reset their passwords.	Medium
F1.4	Users should be able to log out once they're done with their session.	Medium
F1.5	The system should be able to differentiate admins and users.	Low

Authentication requirements F1.1 to F1.4 have been completed as the key feature of the application. The team has decided to omit requirement F1.5 because of its low priority as the users can still enjoy the full functionality of the application without the admin functionality in-built in the application.

3.1.2 Code Collaboration

Identifier	**Description**	**Priority**
F2.1	Users should be able to enter the solution to the text field in near-real-time for both users.	High
F2.2	Users should be able to view the coding question.	High
F2.3	Users should be able to instant message one another.	Medium
F2.4	Users should be able to exit code collaboration once his/her partner is done and be sent back to the question difficulty selection page.	Low
F2.5	Users should be able to re-connect to the collaboration room if either user is disconnected.	Low
F2.6	Users should be able to see their written code even after refreshing the page by storing these data in the cache.	Low

Code Collaboration requirements F2.1 to F2.6 have all been completed.

3.1.3 Peer Matching System

Identifier	**Description**	**Priority**
F3.1	Users should be able to select the difficulty of the question. (Easy, Medium or Hard)	High
F3.2	Users are matched with another user based on their selected difficulty level of Easy, Medium, or Hard.	High
F3.3	Users would be timed out after 30 seconds if not successfully matched and sent back to the difficulty selection page.	High

Peer Matching System requirements F3.1 to F3.3 have all been completed.

3.1.4 Programming Question System

Identifier	**Description**	**Priority**
F4.1	The programming questions should have an associated proficiency level of Easy, Medium, and Hard.	High

F4.2	Users should be able to have their questions selected randomly.	Medium
F4.3	The programming questions should have an associated category of programming questions such as Data Structures, Algorithms, Runtime associated with it.	Low
F4.4	Users should be attempting questions that they have not done before.	Low
F4.5	Admin should be able to add questions to the question pool.	Low
F4.6	Admin should be able to delete questions from the question pool.	Low
F4.7	Admin should be able to edit questions from the question pool.	Low

Programming question system requirements F4.1 to F4.3 have been completed. Our team has decided to not work on F4.4 after discussion because of the decently large number of questions available, the chances of getting a new question is still reasonable. F4.5 to F4.7 is also omitted in this phase of the project as the question pool could still be added by the programmers manually and will not affect the application use cases for the users.

3.1.5 Profile

Identifier	**Description**	**Priority**
F5.1	Users should be able to view their personal profile information.	Low
F5.2	Users should be able to view past attempted questions.	Low
F5.3	Users should be able to change their username and password.	Low

Profile requirements are omitted in this project phase as our team feels that it does not affect the usability of the application. Further iterations can consider implementing this for quality of life improvements.

3.2 Non-functionality Requirements (NFRs)

The NFRs below are listed in their order of priority:

1. Security
2. Usability
3. Scalability

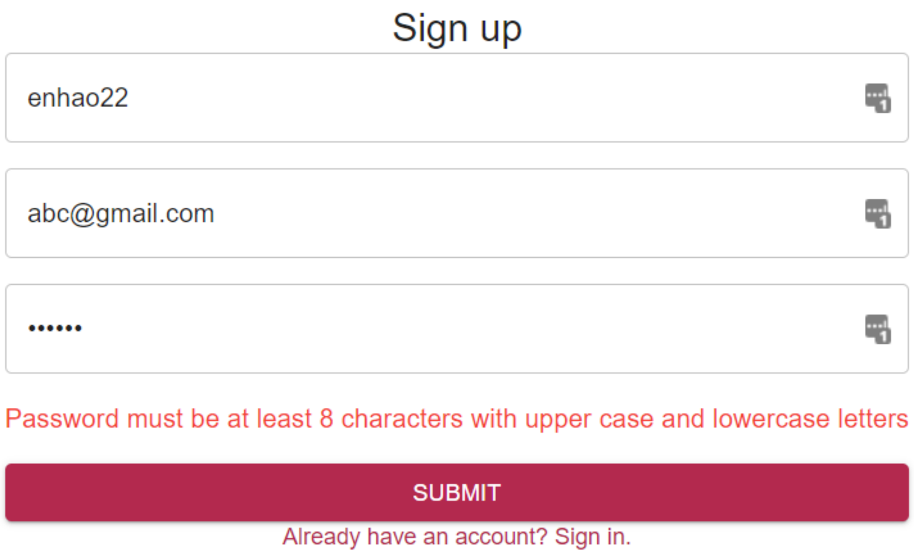
3.2.1 Security Features

Identifier	Description	Priority
NFR1.1	Passwords should have at least 8 characters, with a combination of lower and upper cases.	High
NFR1.2	Users should be authenticated using JWToken	High
NFR1.3	Passwords should be hashed (SHA-256) before storing in the database	Medium

3.2.1.1 Complex Password

In order to have better security, passwords are an essential touchpoint for our users. From NFR1.3, we have already hashed the passwords for security results, however, in the event that hackers are able to get hold of these hashed passwords, we want to make it infeasible for the hackers to brute force and retrieve the plaintext password from the hashed password.

Checks are done on both the frontend and the backend to cover both groups.



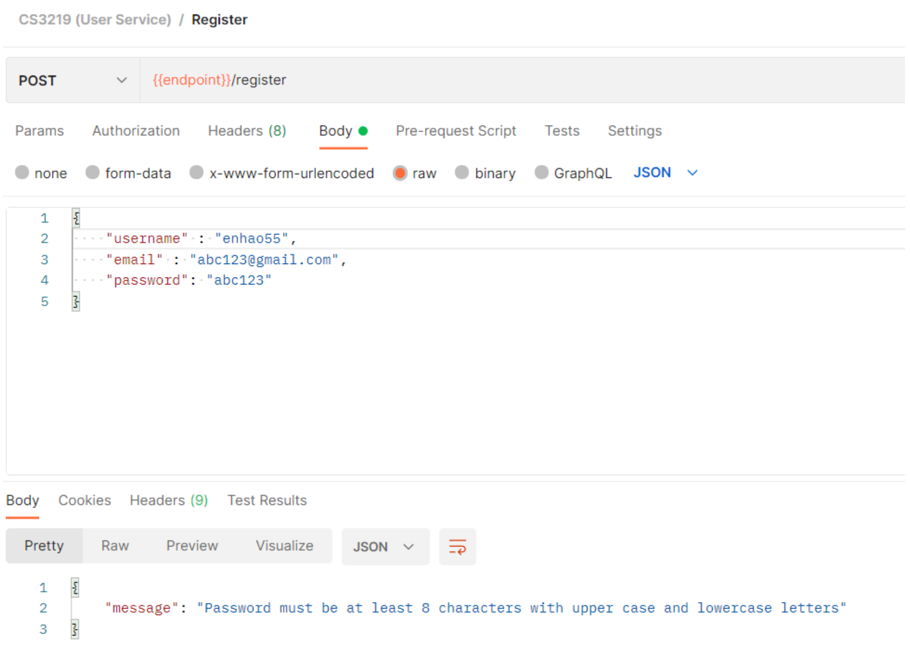
Sign up

enhao22

abc@gmail.com

••••••

Password must be at least 8 characters with upper case and lowercase letters


CS3219 (User Service) / Register

POST {{endpoint}}/register

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "username" : "enhao55",
    "email" : "abc123@gmail.com",
    "password": "abc123"
}
```

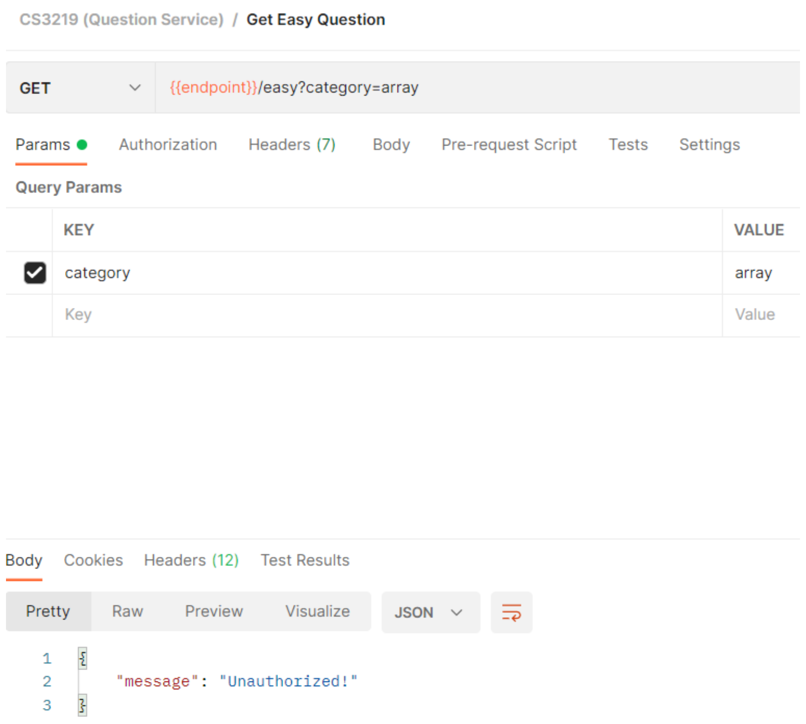
Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "message": "Password must be at least 8 characters with upper case and lowercase letters"
}
```

3.2.1.2 JWToken Authentication

JW Token Authentication ensures that the user is who it claims to be. JWT Token Authentication code is required across ALL essential user, matching and question services to ensure that only authorized users are able to interact with the services. In the event that the token is missing or invalid, an error message will be displayed accordingly.


CS3219 (Question Service) / Get Easy Question

GET {{endpoint}}/easy?category=array

Params ● Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

	KEY	VALUE
✓	category	array
	Key	Value

Body Cookies Headers (12) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "message": "Unauthorized!"
}
```

3.2.1.3 Hashed Password

In order to ensure that hackers are not able to see the password in plaintext, peerprep uses the external library **bcrypt** to perform hashing of the passwords using SHA256. In addition to regular hashing, this library supports the use of cryptographic salts.
Without salts, the same password would still generate the exact same hash, which makes it very vulnerable when malicious users get hold of our database.
With salts, the hash would be unique even in the instance two users choose the same passwords. This would force attackers to re-compute each hash using the salts for each user, which would greatly increase the difficulty of cracking the users passwords.
Combined with NFR1.1, this greatly increases the protection of our users’ passwords.

3.2.2 Scalability Features

Identifier	Description	Priority
NFR2.1	The system should be able to scale up with the number of users and handle the load of up to 100 users at any one time.	High

3.2.2.1 On-demand Scalability

The ability to scale on demand was one of the NFR identified to ensure that our system will be able to cope with any sudden spikes in visitors. By using microservices, we were able to make use of a Kubernetes cluster to individually scale up any microservice when required.

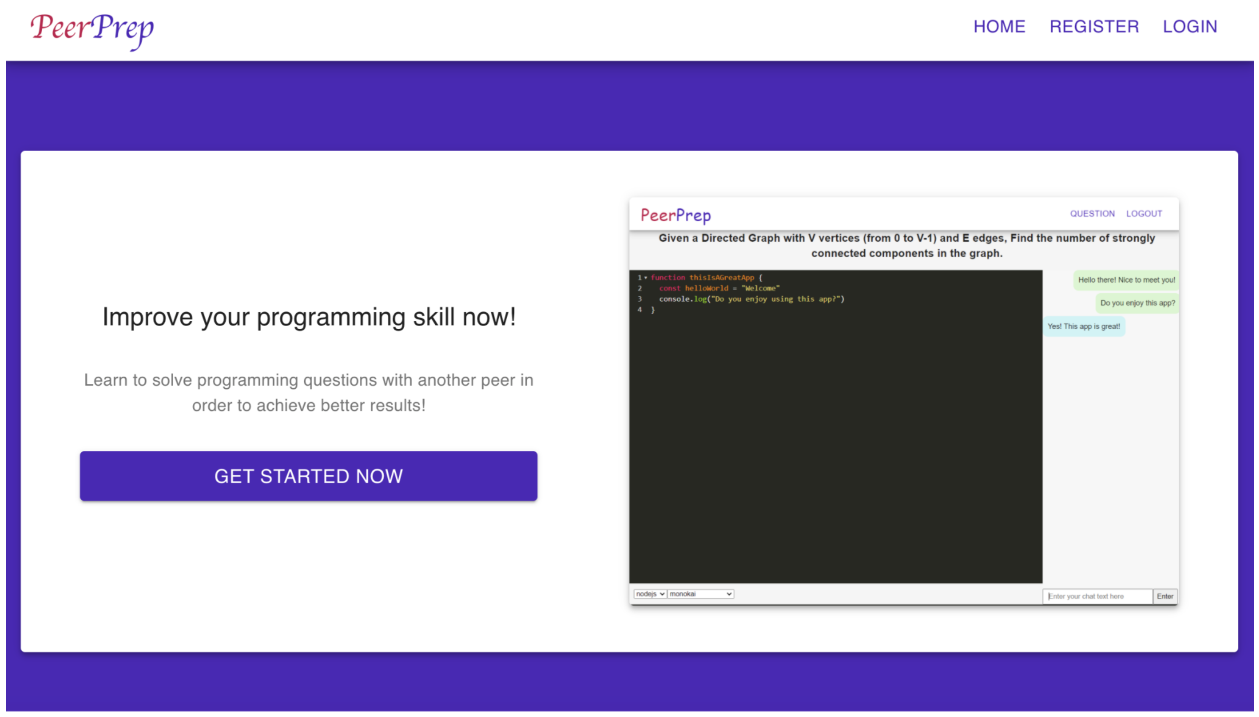
In the Kubernetes cluster, a horizontal pod autoscaler is employed, which has the ability to scale the number of pods up or down, depending on CPU usage. Metrics server is used to track CPU usage for the horizontal pod autoscaler to auto-allocate more pods when CPU usage rises.

3.2.3. Usability Features

Identifier	Description	Priority
NFR3.1	Users should be able to understand and get started on using the application within 5 minutes.	Medium
NFR3.2	Logged-in users should not take more than five clicks to select their questions and get matched.	Low
NFR3.3	Users should receive meaningful alerts in the application flow to aid user experience.	Medium

3.2.3.1 Self-Sufficient Application

To ensure that the application is self-sufficient and can be used without any of us explaining the feature, we have created an informational landing page that allows users to understand the functionality of the application.



Improve your programming skill now!

Learn to solve programming questions with another peer in order to achieve better results!

Given a Directed Graph with V vertices (from 0 to V-1) and E edges, Find the number of strongly connected components in the graph.

```
function thisIsAGreatApp {
  const helloWorld = "Welcome"
  console.log("Do you enjoy using this app?")
}
```

Hello there! Nice to meet you!

Do you enjoy this app?

Yes! This app is great!

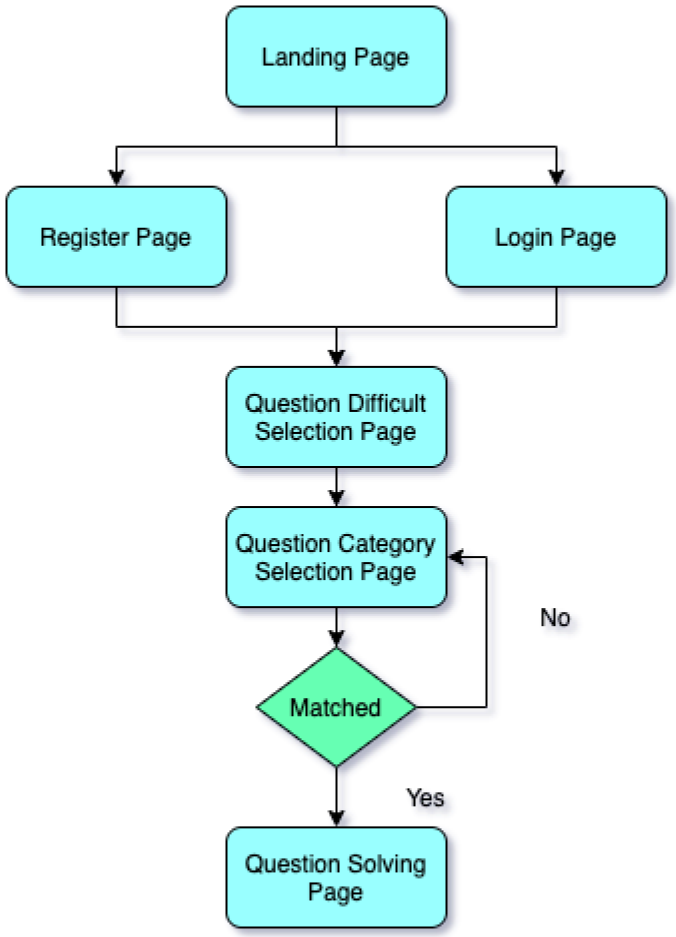
(Note: The landing page template is referenced from https://reactsaastemplate.com)

3.2.3.2 Application can be used within 5 clicks

To ensure that our users have a good experience using the application, we seek to ensure that our users are able to use the application within 5 clicks.

From the flowchart below, we can see that from the landing page to the question-solving page where the users attempt to solve the problem, it can be done within 5 clicks.

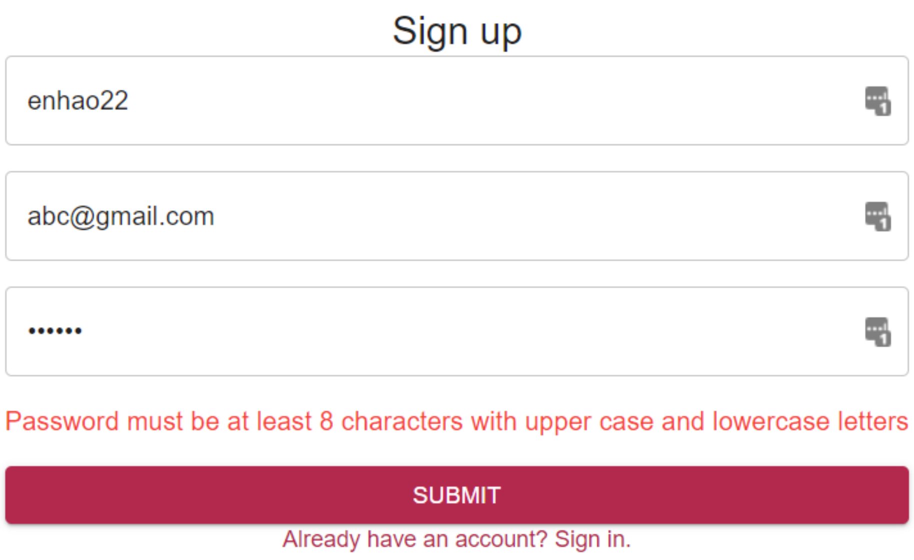
Consideration: We made sure that after successful registration, we would log the user in to ensure that they have a much better experience using the application.



3.2.3.3 Meaningful Alerts

To ensure that the user has a better experience using the application, we have made an effort to ensure that error messages are shown directly to the user in the frontend.

Below are some of the examples where error messages are shown.


Sign up

enhao22

abc@gmail.com

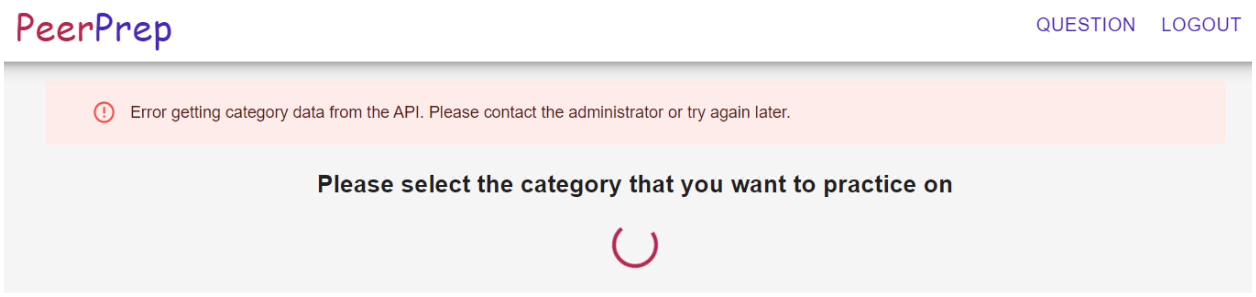
••••••

Password must be at least 8 characters with upper case and lowercase letters

SUBMIT

Already have an account? Sign in.

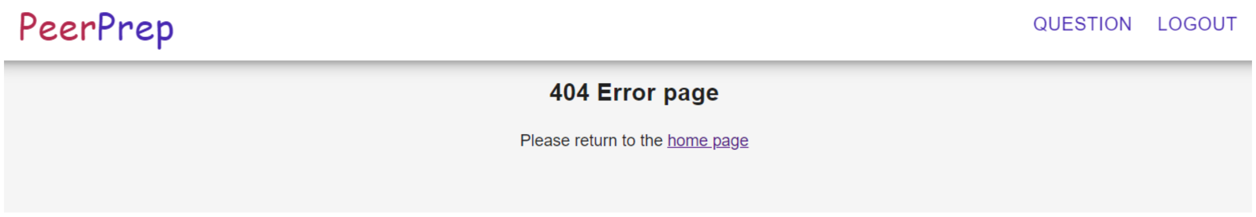
(Password error)


PeerPrep QUESTION LOGOUT

Error getting category data from the API. Please contact the administrator or try again later.

Please select the category that you want to practice on

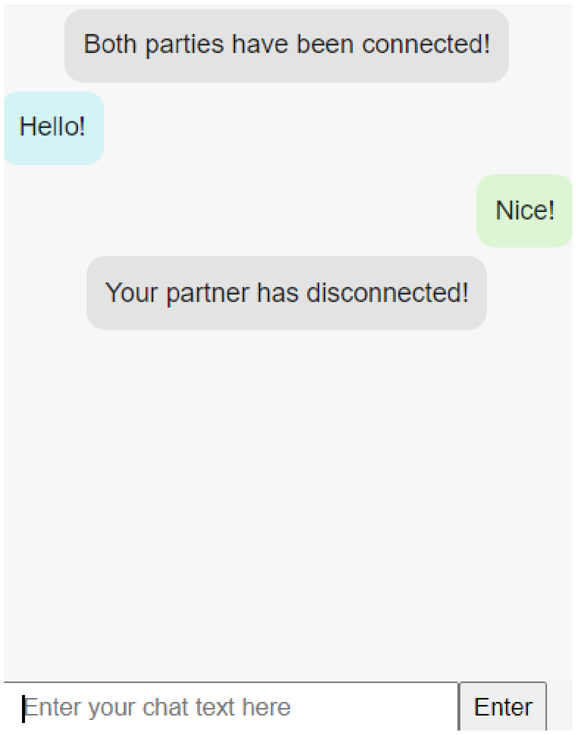
(Error alert during question selection and loading indicators)


PeerPrep QUESTION LOGOUT

404 Error page

Please return to the home page

(404 Error Page for pages that are not found w/ redirection included)


Both parties have been connected!

Hello!

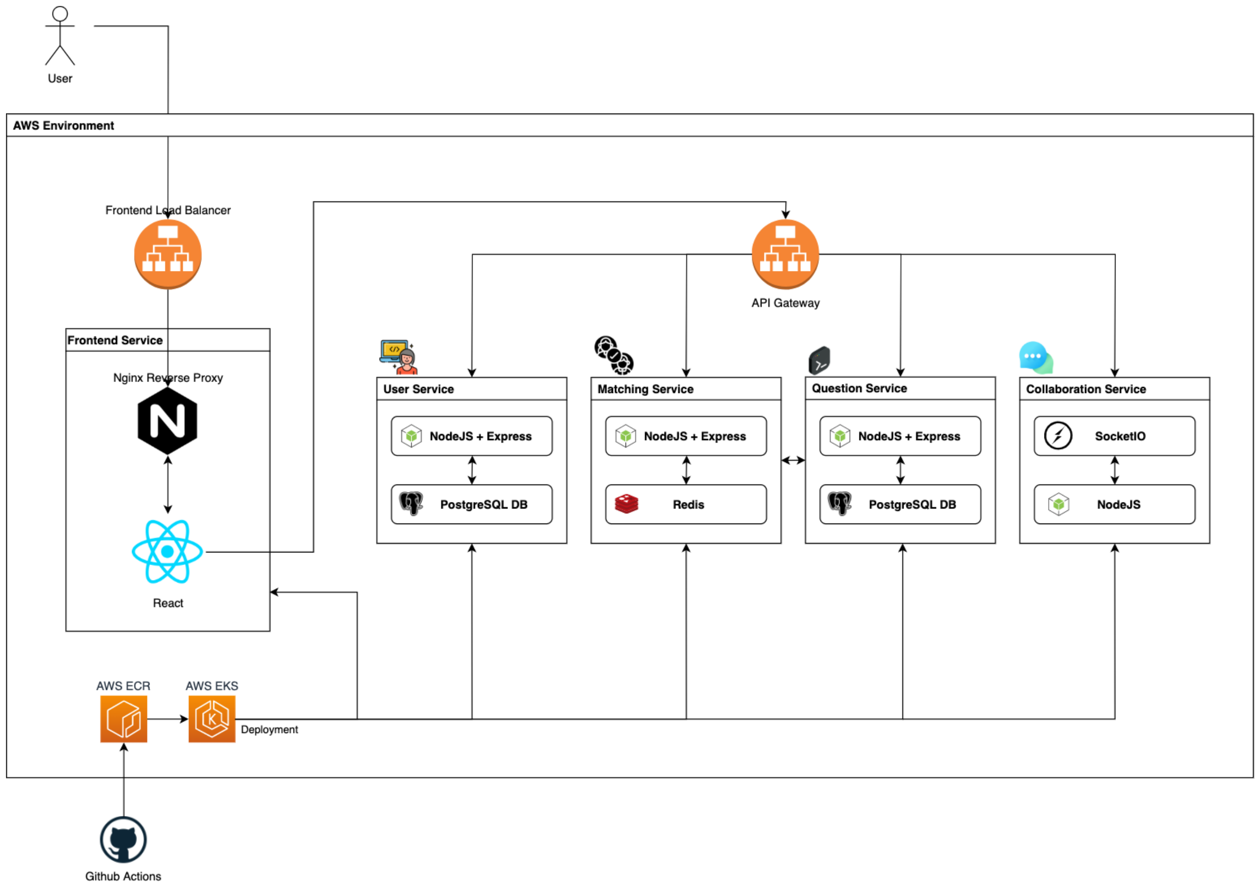
Nice!

Your partner has disconnected!

(Connection status for the users)

4. Architecture Design

4.1 Architecture Diagram



4.2 Design Decisions

4.2.1 Microservices

Benefits	• Able to employ separation of concerns where each Microservice can handle one small portion of the application • Freedom of different implementation within each Microservice • Scalability of individual Microservices instead of the entire application
Associated Requirements	NFR2: Scalability
Justification	With the use of Microservices Architecture, we are able to employ the “Separation of Concerns” principle in Software Engineering where each

	Microservices handles a different function of the application. This allows the team to make use of different implementations or sub-architecture which are best suited for each of the Microservice. With Microservices, we can also scale each Microservice individually when needed. In situations where one of the services such as the question service has a higher network load, we can only scale up the question service rather than scale up the entire application which is a more cost-effective approach.

4.2.2 Isolated Databases for Each Service

Benefits	● No single point of failure ● Reduce Coupling
Associated Requirements	Availability
Justification	Databases are potential points of failure for an application. When there is a single database for all the services, it means that when the database fails, the entire application will not be usable anymore. This largely reduces the usability of the application. However, we decide to use different databases for each Microservices to ensure that even when one of the databases fails, it does not affect the availability of the other Microservices as well. This reduces coupling as each Microservices only has a dependency on its own database, rather than towards a global database.

4.2.3 Authentication in Microservices

Benefits	● JWToken Authentication allows only authorized users to perform actions in the application
Associated Requirements	NFR1: Security
Justification	The Server generates a JWT on successful Client login and returns it to the client. The Client saves the JWT and from now on, every subsequent request from the Client to access routes, services, and resources have the JWT attached to the headers. When receiving JWT from the client, the Server gets the Signature, verifying that the signature is correctly hashed by the same algorithm and Secret. If it matches the Server's signature, the JWT is validated. Moreover, JWT is a good way of securely transmitting information between parties. As JWTs can be signed, one can be sure the senders are who they claim

	to be. Additionally, as the signature is calculated using the header and the payload, one can also verify that the content has not been tampered. As JWTs are self-contained and it is very hard to revoke them once issued, tokens are set to have a short expiration time. For example, reset password token expiration is set to 10 minutes.

4.3 Design Principles and Patterns

4.3.1 React Flux

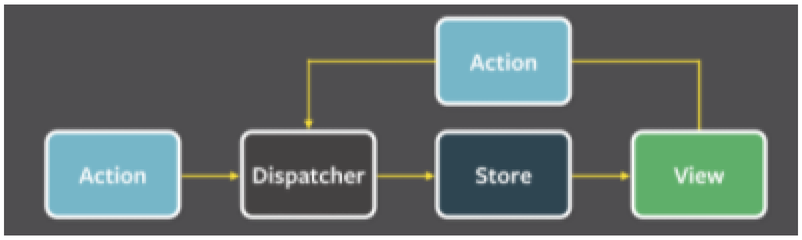


Figure 4.3.1.1 React Flux Pattern

The React frontend makes use of a flux pattern to update its views.

Example use case:
Let us take a look at the text field on the sign in page. When a user enters a character into the text field, a method setState() is called that helps us see the character we entered on the text field. When the setState() method is called, it fires an action to the dispatcher. The store has already registered a callback that listens on to this character update action. When this action reaches the dispatcher, it runs through a switch statement and executes the callback that updates the related store. React also uses a controller-view whereby the view listens for events that are broadcasted by the stores it depends on. When the store that it is listening to gets updated, the data from the store is passed down the chain of descendants that allows us to see our entered character on the text field.

4.3.2 SocketIO Publish-Subscribe Pattern

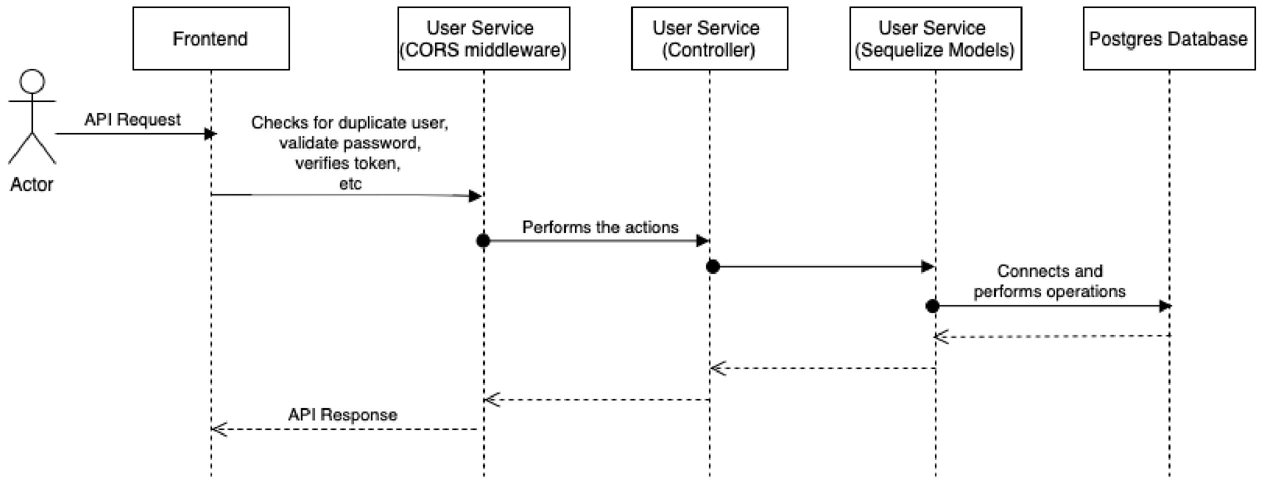
Communication of users through the chat functionality is accomplished using Socket.IO publisher/subscriber pattern. Please refer to section 5.4 for more information.

5. Microservices

5.1 User Service

The user service handles the login, registration, forget password, reset password and log out functions for users.

Below is the sequence diagram demonstrating how users interact with the User Service.


Frontend
User Service (CORS middleware)
User Service (Controller)
User Service (Sequelize Models)
Postgres Database
Actor
API Request
Checks for duplicate user, validate password, verifies token, etc
Performs the actions
Connects and performs operations
API Response

5.1.1 Supported Endpoints

Below are the endpoints supported by the user service.

Endpoint	**POST** /auth/register
Description	Users sign up with an username, email address, and password to create a new account.
Sample Request Body	`{ "username": "karthika", "email" : "mdkkarthika@gmail.com", "password": "TestTest" }`
Sample Response	`{ "message": "User was registered successfully!" }`

Endpoint	**POST** /auth/login
Description	Users can login with their registered username and password.

Sample Request Body	`{`
	    `"email": "mdkkarthika@gmail.com",`
	    `"password": "TestTest"`
	`}`

Sample Response	`{`
	    `"id": 1,`
	    `"username": "karthika",`
	    `"email": "mdkkarthika@gmail.com",`
	    `"roles": [`
	        `"ROLE_USER"`
	    `],`
	    `"accessToken":`
	`"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNjM2MjgwMDQ1LCJleHAiOjE2MzYyODM2NDV9.Zp00txM1bdtCKFKrsfK7Rauhxq1AW5ANZbNkwVSwAWk"`
	`}`

Endpoint	**PATCH** /auth/forgotPassword
Description	Users who have forgotten their password get a one time reset password link sent to their email address. This link will only be valid for 10 minutes.

Sample Request Body	`{`
	    `"email": "mdkkarthika@gmail.com"`
	`}`

Sample Response	`{`
	    `"message": "Check your email"`
	`}`

Subsequent interactions require Bearer token authentication to ensure that only registered users can perform the operations.

Endpoint	**GET** /auth/checkValidUser
Description	For frontend to verify that the user who wishes to reset their password is valid.
Sample Request	/auth/checkValidUser?token={token}

Sample Response	`{ "message": "User is valid", "user": { "id": 1, "username": "karthika", "email": "mdkkarthika@gmail.com", "password": "$2a$08$6Rkm1Pxwg97xjPBDUo00OOzWtfagMNUAD8Ln6OsJbJNC8GQoVB7oi", "resetLink": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNjM2MjgwODgxLCJleHAiOjE2MzYyODE0ODF9.dEz5aD7wmN-VtJjtqPy0Y-ayahGCProgLpZLIjhjRfw", "createdAt": "2021-11-07T10:09:20.544Z", "updatedAt": "2021-11-07T10:28:01.466Z" } }`

Endpoint	**PUT** /auth/resetPassword
Description	After the user is verified, user can change to a new password
Sample Request	/auth/resetPassword?token={token}
Sample Request Body	`{ "password" : "MyNewPassword" }`
Sample Response	`{ "message": "Password updated" }`

Endpoint	**POST** /auth/logout
Description	Users log out of the application and get redirected to Homepage.
Sample Response	`{ "message": "You have been logged out", "userId": 1 }`

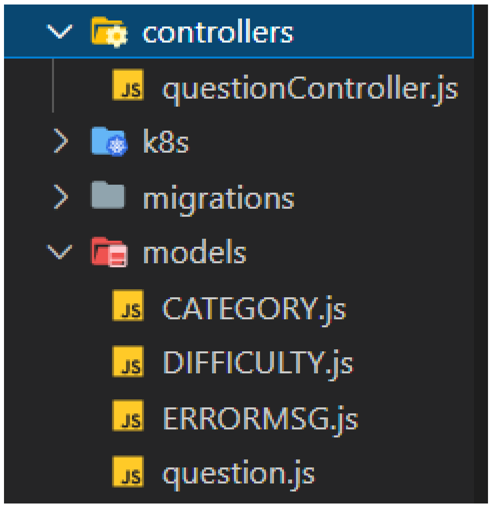
5.2 Question Service

The question service handles the storing and retrieving of random questions after users have successfully been matched. It also handles the category retrieval based on the given difficulty.

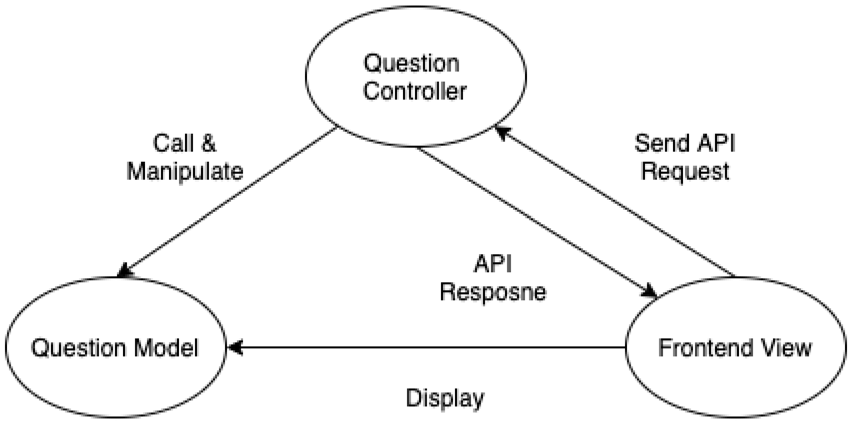
5.2.1 MVC Pattern

The question service is created by following the MVC pattern.

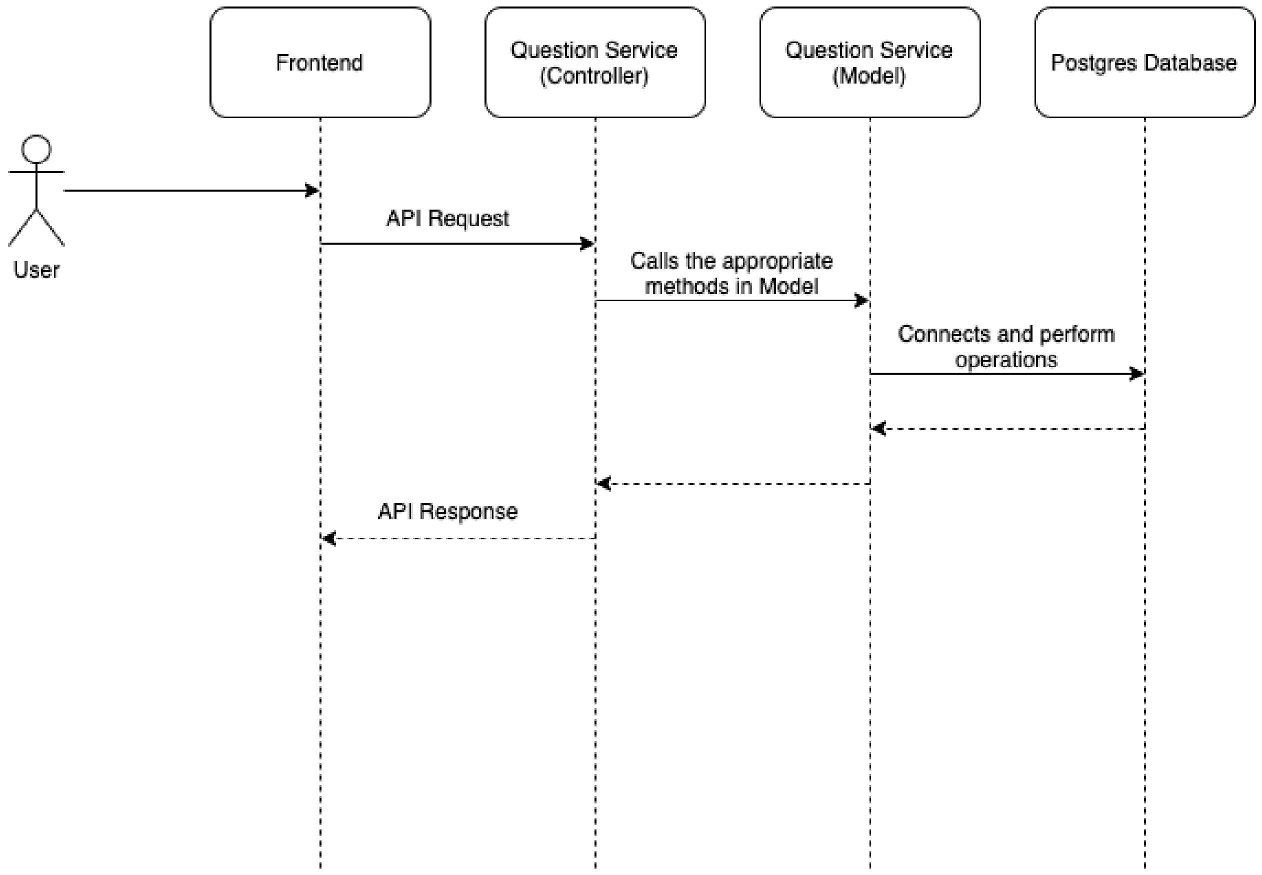
Controllers: Controls the flow of the endpoints after receiving a request from view (frontend) and renders data back to the view. Manipulate the models such as questions for the endpoints.
Models: Models an entity such as a question, category, and difficulty.



```
controllers
    questionController.js
k8s
migrations
models
    CATEGORY.js
    DIFFICULTY.js
    ERRORMSG.js
    question.js
```



Generally, the sequence of events when the user uses the application to interact with the question service is as follows.


Frontend
Question Service (Controller)
Question Service (Model)
Postgres Database
User
API Request
Calls the appropriate methods in Model
Connects and perform operations
API Response

5.2.2 Supported Endpoints

Below are the endpoints supported by the question service. All interaction requires Bearer token authentication to ensure that only registered users can perform the operations.

Endpoint	**GET** /{difficulty}?category={category}
Description	Returns a random question based on the given difficulty and category.
Sample Request	/question/easy?category=array

Sample Response	
	```
	{
	    "questions": [
	        {
	            "id": 24,
	            "difficulty": "easy",
	            "category": "array",
	            "question": "Test question",
	            "externallink": null
	        }
	    ]
	}
	```

Endpoint	**GET** /category/{difficulty}
Description	Returns all the available categories of the given difficulty
Sample Request	/question/category/easy
Sample Response	`{ "categories": [ { "category": "array" }, { "category": "others" }, { "category": "string" } ] }`

Endpoint	**POST** /question/add
Description	Add a given question into the database
Sample Request Body	`{ "difficulty" : "easy", "category": "", "question" : "Find all pairs of an integer array whose sum is equal to a given number.", "link": "" }`
Sample Response	`"Question created"`

Endpoint	**DELETE** /question/delete/{id}
Description	Delete the given question id from the database
Sample Request	/question/delete/3
Sample Response	`"Question #3 deleted"`

Endpoint	**UPDATE** /question/update/{id}
Description	Delete the given question id from the database
Sample Request	/question/update/2
Sample Request Body	`{ "difficulty" : "easy", "category": "string", "question" : "Edited question", "link": "" }`
Sample Response	`"Question #2 updated"`

5.3 Matching Service

MatchingService handles the matching of users before they are put into the same lobby.

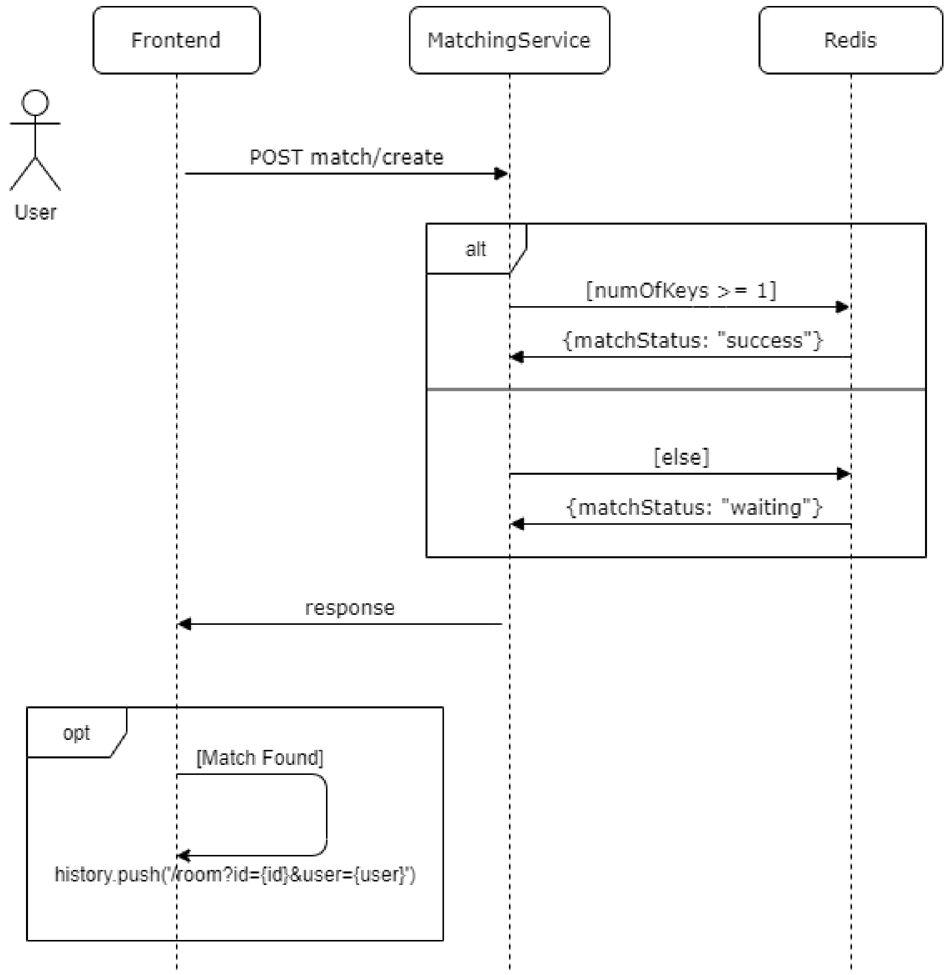


Figure 4.1.3.1 Sequence Diagram for matching users

A key is created in the MatchingService Redis store with an expiry time of 30 seconds whenever a user tries to connect. The key is a combination of the difficulty type, question type, and user id to allow matching of users of selecting different difficulty levels and question types.

5.3.1 Supported Endpoints

Endpoint	**POST** /match/create
Description	Creates a match for users and matches if there are greater or equal to one user in the store with the same question type and difficulty level
Sample Request Body	`{ "user": "user1", "difficulty" : "easy", "category": "array" }`
Sample Response	`{ "matchStatus" : "easy", "matchId": "467a5af3-09b8-4f2e-9e3e-cc87a28a007b", "partnerId" : "1", "question": "Remove all duplicates from a given array" }`

Endpoint	**POST** /match
Description	Polling endpoint for the frontend to call every 5 seconds to check for the status of the match.
Sample Request Body	`{ "user": "user1", "difficulty" : "easy", "category": "array" }`
Sample Response	`{ "matchStatus" : "easy", "matchId": "467a5af3-09b8-4f2e-9e3e-cc87a28a007b", "partnerId" : "1", "question": "Remove all duplicates from a given array" }`

Endpoint	**POST** /match/delete

Description	Deletes the key in the store when the user decides to exit matching. Returns the deleted key.

Sample Request Body	`{`
	`"user": "user1",`
	`"difficulty" : "easy",`
	`"category": "array"`
	`}`

Sample Response	`{`
	`"value" : "easyarrayuser1"`
	`}`

Endpoint	**POST** /match/deleteZombie
Description	Deletes zombie key in the store when the user disrupts frontend flow by refreshing the page.

Sample Request Body	`{`
	`"user": "user1",`
	`"difficulty" : "easy",`
	`"category": "array"`
	`}`

Sample Response	`{`
	`"value" : "easyarrayuser1"`
	`}`

5.4 Collaboration Service (Real Time Communication)

5.4.1 Code Collaboration

Code collaboration ensures real-time code sharing, where both parties are able to work on the same piece of code concurrently with confidence. This includes the ability to resolve conflicts and serialize concurrent modifications.

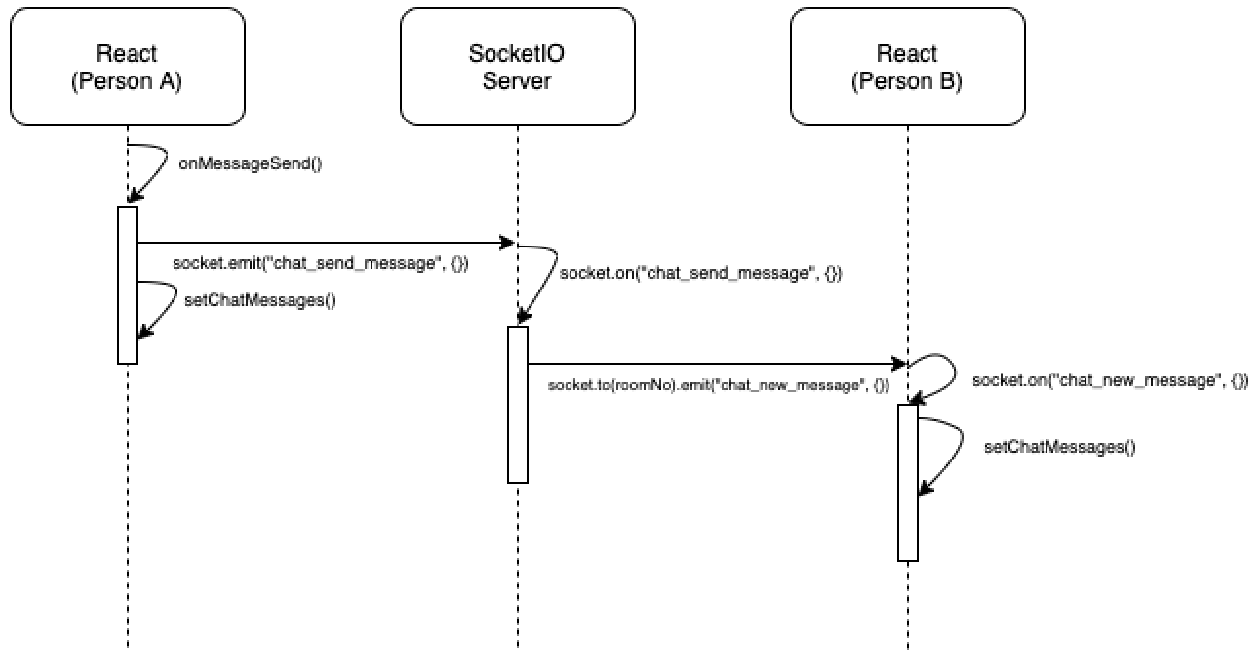
For this, we made use of conflict-free replicated data types (CRDTs), a data structure, which ensures that concurrent edits are updated independently and concurrently without coordination between both parties. This allows for consistent data (code in the text editor) between both parties.

We used Yjs, a CRDT library for nodeJS, and paired it with codemirror, a text editor library, which allows for binding with Yjs right out of the box. To reduce complexity in the implementation, we used a public signalling server.

5.4.2 Chat Room

The collaboration service handles the backend of the “chat room” function that exists in the code collaboration page, which ensures real-time communication between both parties in the coding room.

To enable real-time communication, we made use of the Socket.IO Javascript library, which enables real time, bi-directional communication between the client and server. The diagram below illustrates how real-time communication between two persons (Person A and B) is being processed by the server.


React (Person A)
SocketIO Server
React (Person B)
onMessageSend()
socket.emit("chat_send_message", {})
setChatMessages()
socket.on("chat_send_message", {})
socket.to(roomNo).emit("chat_new_message", {})
socket.on("chat_new_message", {})
setChatMessages()

When both users are matched by the matching service (5.3), they are given a unique identifier (ID) as their room number. Upon being redirected to the code collaboration page in the frontend (i.e. /room/id={roomid}), it will make a connection with the SocketIO server residing in the collaboration service, sending a request to join a room with the specified room ID.

Once the SocketIO server receives the room join request, it will “enrol” the user into that particular room ID. Going forward, any messages that are typed on the chat box will be sent as a request to the SocketIO server, processed, and then emitted to other persons in the room (since the room only consists of a pair, it’s the other person).

5.5 DevOps

5.5.1 Development Process

The development of the project is done following the Agile sprint development process.

Weekly standup meeting is conducted with the following goals in mind:

1. Review what was done previously
2. Discuss project blockers
3. Discuss the goals and functions to achieve before the next weekly standup meeting

5.5.2 CI/CD Pipeline

Our CI/CD pipeline, both in the frontend and backend, are both using Github Actions.

5.5.2.1 Frontend CI/CD

The frontend CI/CD is invoked when code changes have been made to the frontend folder, and is pushed to the master branch. The workflow is as follows:

1. Building the “frontend” docker image
2. Pushing it AWS Elastic Container Registry (ECR)
3. Swap the old image with the new “frontend” image on AWS Elastic Kubernetes Service (EKS)

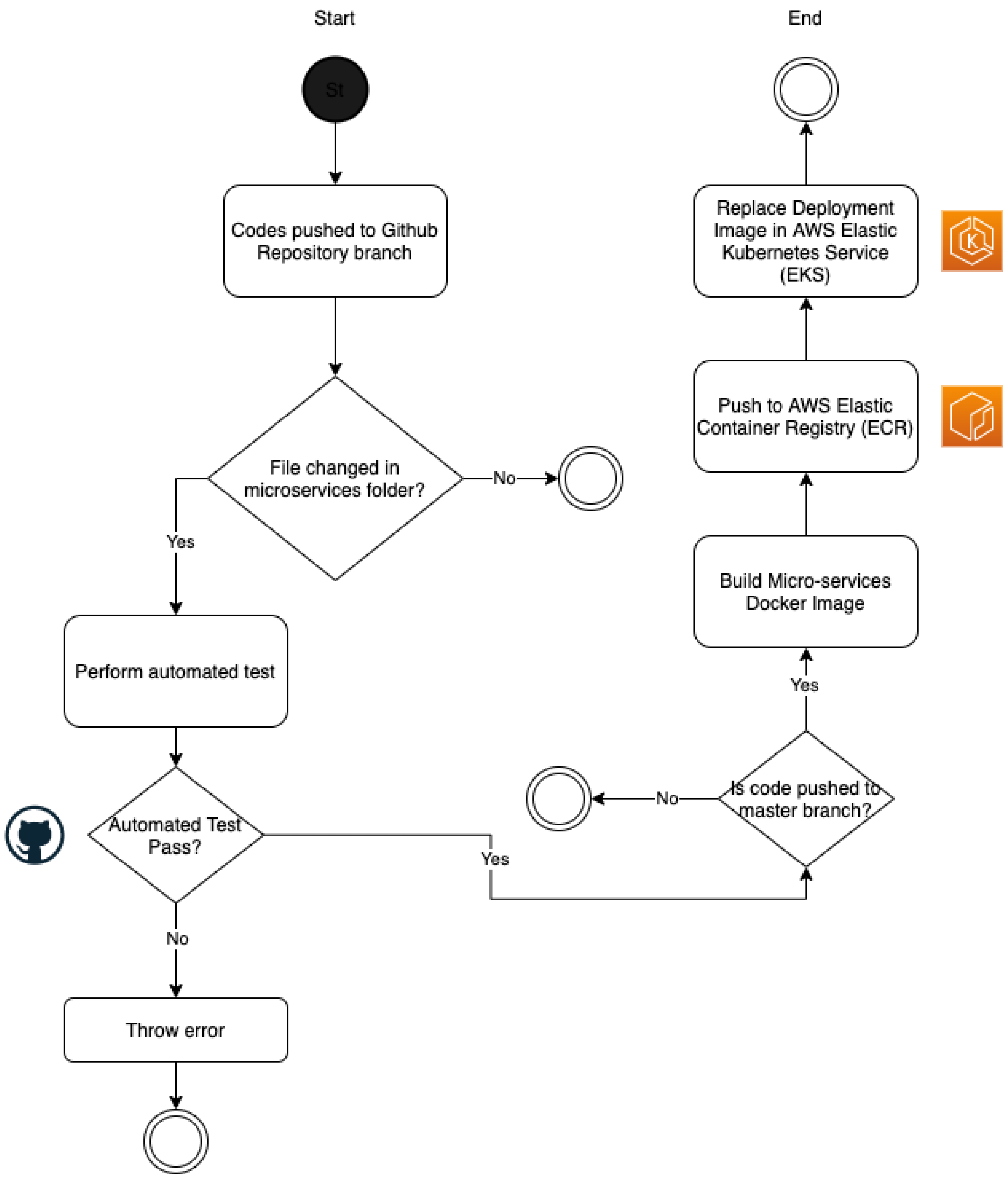
5.5.2.2 Backend CI/CD

With the inclusion of test cases in the backend microservices, the CI/CD pipeline workflow will be slightly different from the frontend.

There are two possible scenarios for the backend CI/CD pipeline:

1. **Code pushed to the respective backend services in a non-master branch**
   In this particular scenario where code is pushed to non-master branches, the CI/CD pipeline would be triggered to only perform the continuous integration (CI) portion of the pipeline. This means that codes for the respective microservices will be built, and test cases will be run to ensure that newly pushed code pass all defined test cases, preventing regression.

2. **Code pushed to the master branch (live deployment)**
   This happens when code is pushed into non-master branches (such as in scenario 1), and their pull requests (PRs) are approved. Then, the pipeline is ready to carry out its continuous deployment (CD) process as follows:
   1. Test cases will be ran
   2. Build microservice docker image
   3. Push image to AWS Elastic Container Registry (ECR)
   4. Set the newly pushed image as the new container for the respective microservice deployment through AWS Elastic Kubernetes Service (EKS)


Start
End
St
Codes pushed to Github Repository branch
File changed in microservices folder?
No
Yes
Perform automated test
Automated Test Pass?
No
Throw error
Yes
Is code pushed to master branch?
No
Yes
Build Micro-services Docker Image
Push to AWS Elastic Container Registry (ECR)
Replace Deployment Image in AWS Elastic Kubernetes Service (EKS)

CI/CD Workflow

The above shows the activity diagram for our Github Actions CI/CD Workflow for a typical backend microservice.

6. Potential Improvement

6.1 Overview

Because of time constraints and various challenges, our team prioritized the high and medium requirements analyzed in section 2. However, given more time, there are various other features that we hope to incorporate into the application.

6.2 Profile Feature

Long-term users might want to change their user settings such as username and password. Although this is of “low” priority based on the functionality requirements, it is still a valid potential improvement to improve the application’s usability.

6.3 Store Question Attempts

Avid users of the application might reach a point where they are consistently getting the same questions. Hence, one of the improvements we can implement is to store the questions attempted by the user. Instead of storing the match result in Redis, we could instead store this information in a database. Therefore, it would allow the matching service to assign a question which the user has not attempted before.

6.4 Admin Control Panel

In our current application, adding new questions to the database could only be done through the backend by manipulating the database directly. This might be inconvenient for future administrators who might not be familiar with how databases work. Therefore, one potential improvement is a creation of an admin control panel, where admins can perform admin-related operations on that platform, with some of them being adding, modifying, or deleting question related data.

<u>7. Reflection and Learning Points</u>

Overall, our team has learned about the various popular software design patterns in the module. Working on this project allowed us to apply our learning by using the enhanced knowledge we have to make design decisions when working on the project.

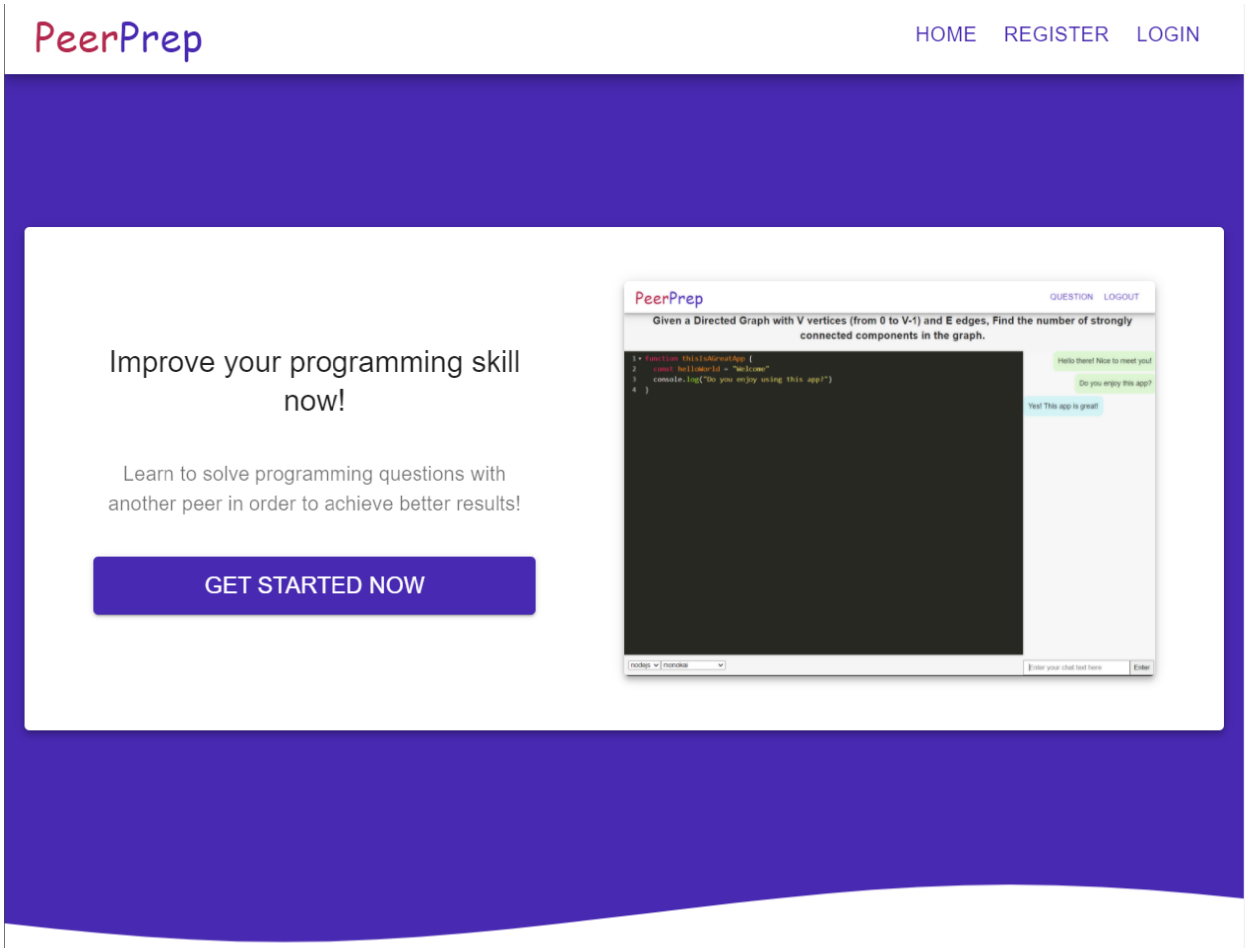
As we had learned about design patterns, we had a common language when discussing the approaches we can take towards designing the codebase. From the project, we have also experienced the iterative nature of software designing as there can be various issues that we had not thought of before that could impact our design decisions. Our team has also taken up the challenge of working using the Microservices architecture as it was a new concept for all of us.

Kubernetes, in particular, has been an interesting technology to work with. While working with Kubernetes initially, it was quite a daunting experience as there were many new concepts and terminologies to learn. However, as the weeks passed, we became more skilled in Kubernetes and learnt how to deploy our project from scratch on AWS Elastic Kubernetes Service (AWS EKS)! This particular skill will definitely be handy for us in the future, be it for future school projects or at the workplace.

Ultimately, we have learned about the benefits of microservices which allows each team member to focus on a specific portion of the backend components in parallel. However, it has also given rise to code complication, which is a general disadvantage of microservices, and in turn increasing code complexity. Nonetheless, our team members have worked very hard on the project, and we hope to continue to better ourselves to become better software engineers, even after this module has ended.

Appendix

Application Screenshot



Improve your programming skill now!

Learn to solve programming questions with another peer in order to achieve better results!

Given a Directed Graph with V vertices (from 0 to V-1) and E edges, Find the number of strongly connected components in the graph.

```
function thisIsAGreatApp {
  const helloWorld = "Welcome"
  console.log("Do you enjoy using this app?")
}
```

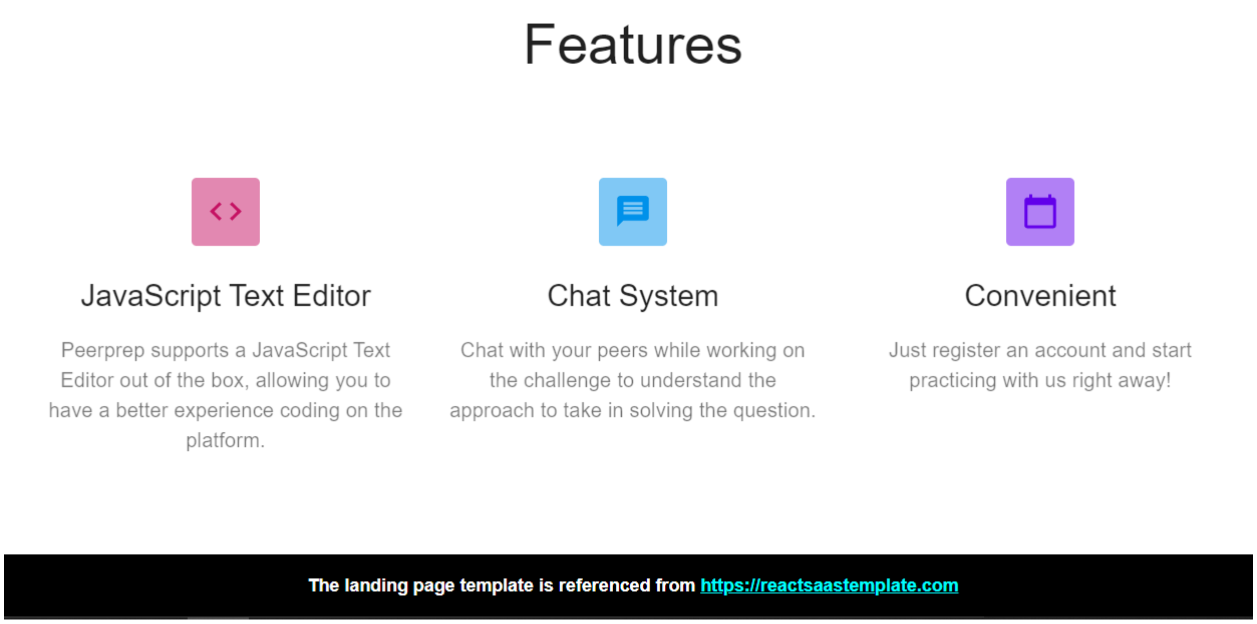
Hello there! Nice to meet you!

Do you enjoy this app?

Yes! This app is great!

nodejs monokai

(Landing Page - Top portion)


Features

JavaScript Text Editor

Peerprep supports a JavaScript Text Editor out of the box, allowing you to have a better experience coding on the platform.

Chat System

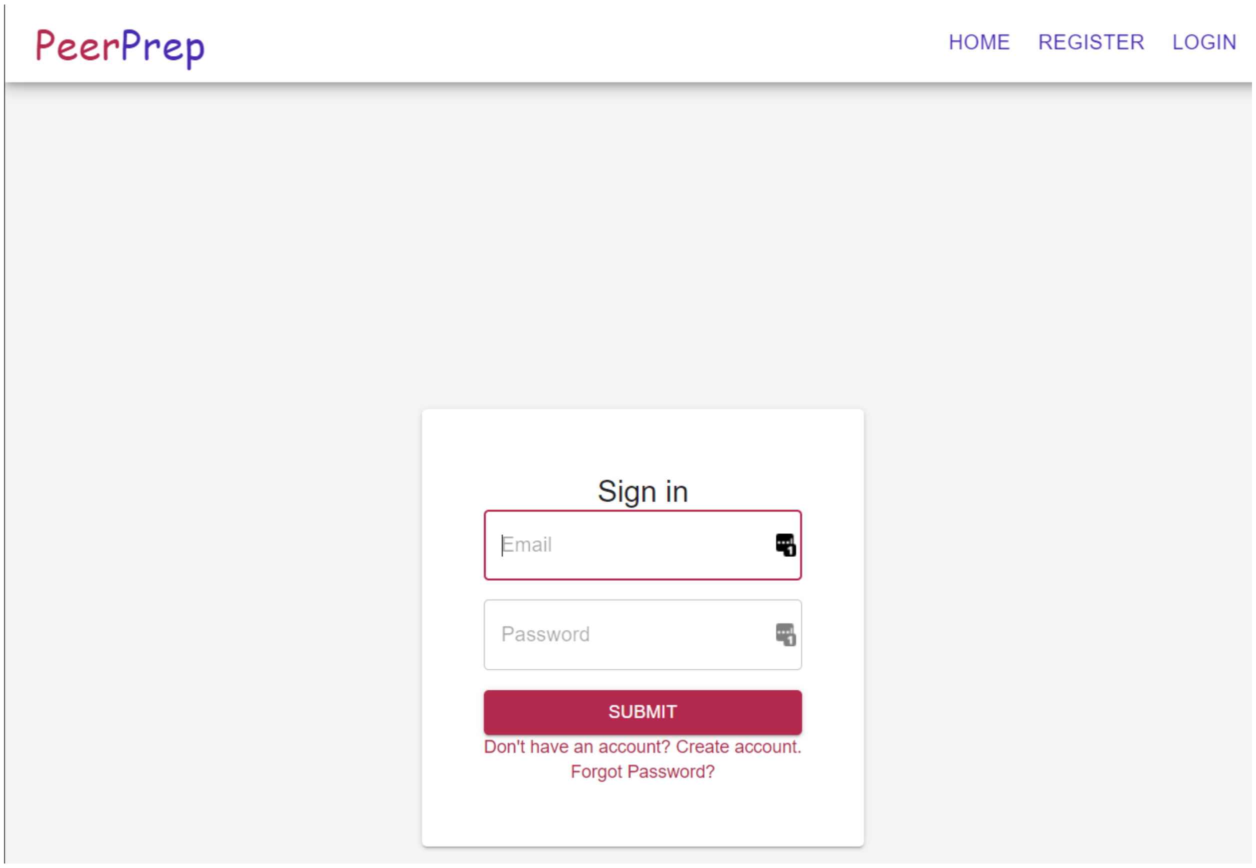
Chat with your peers while working on the challenge to understand the approach to take in solving the question.

Convenient

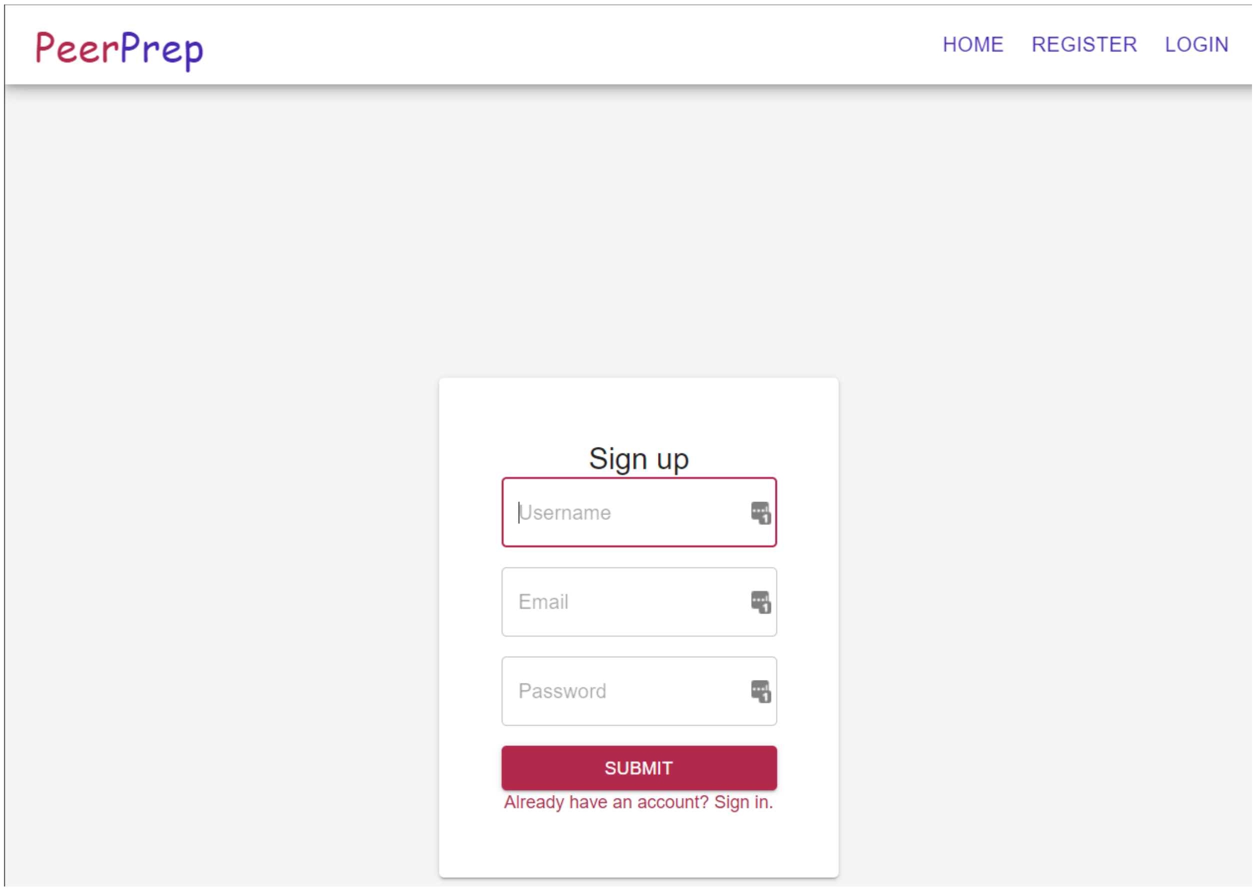
Just register an account and start practicing with us right away!

The landing page template is referenced from https://reactsaastemplate.com

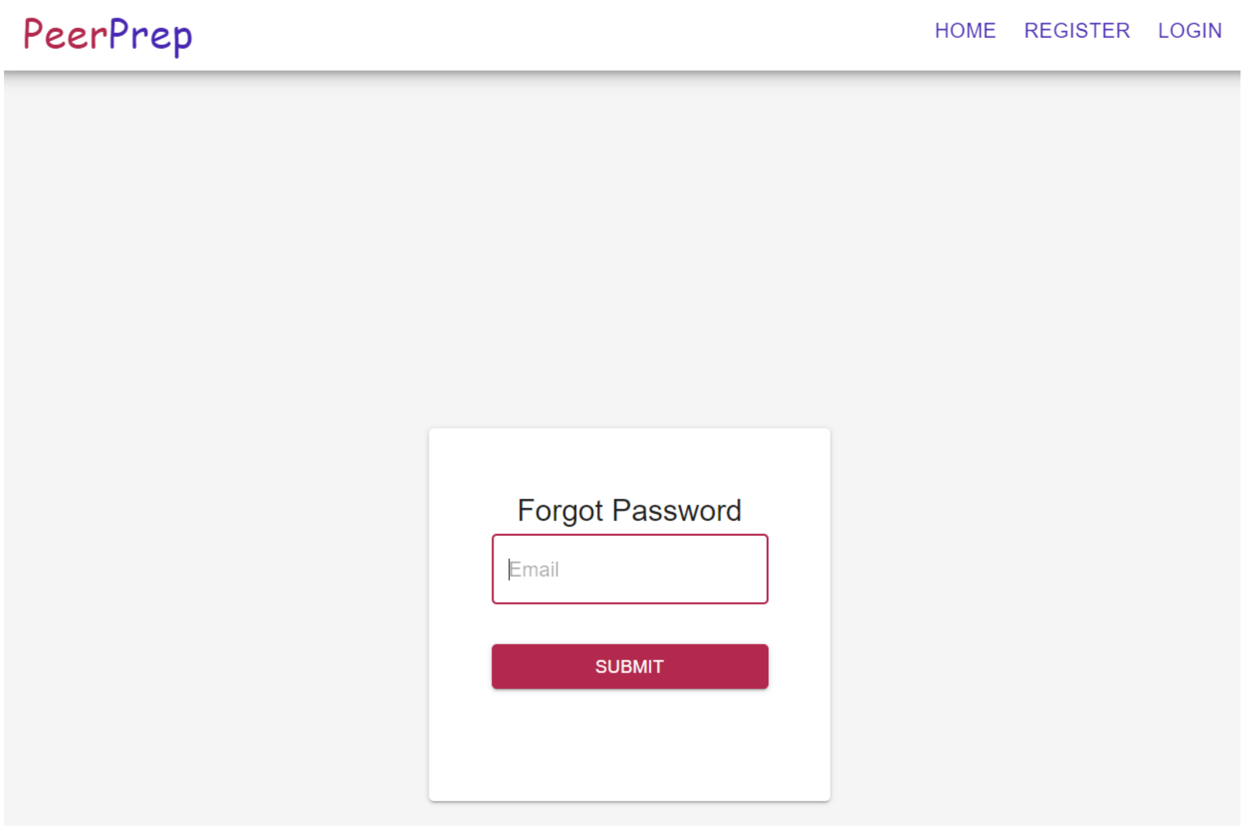
(Landing Page - Bottom portion)



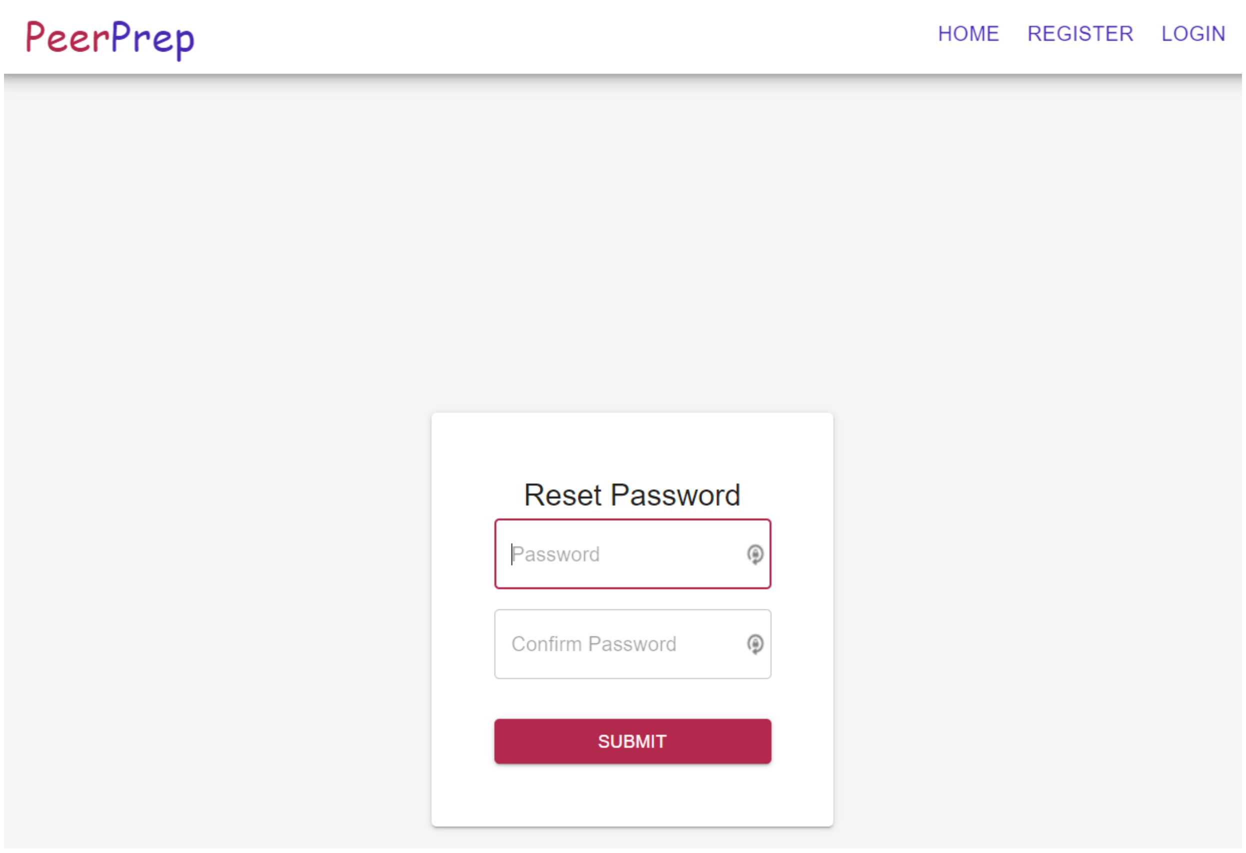
(Sign In Page)


PeerPrep
HOME
REGISTER
LOGIN
Sign up
Username
Email
Password
SUBMIT
Already have an account? Sign in.

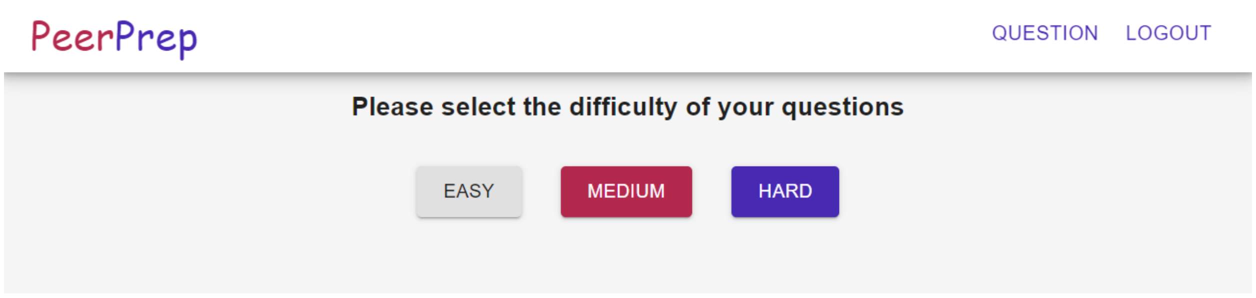
(Sign Up Page)



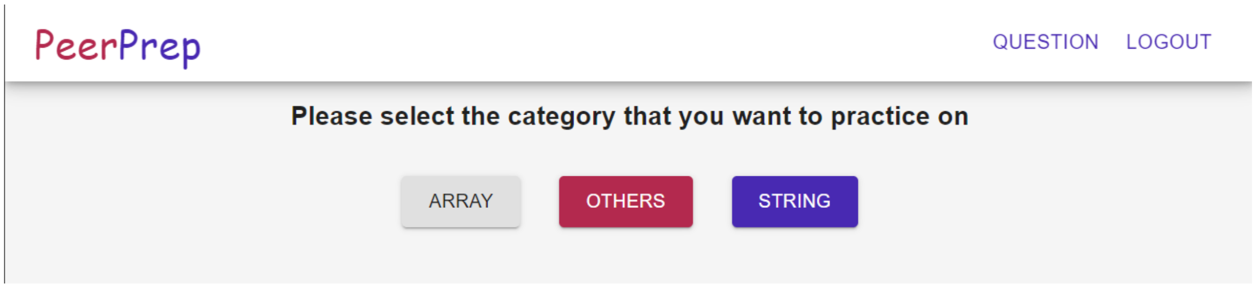
(Forget Password Page)



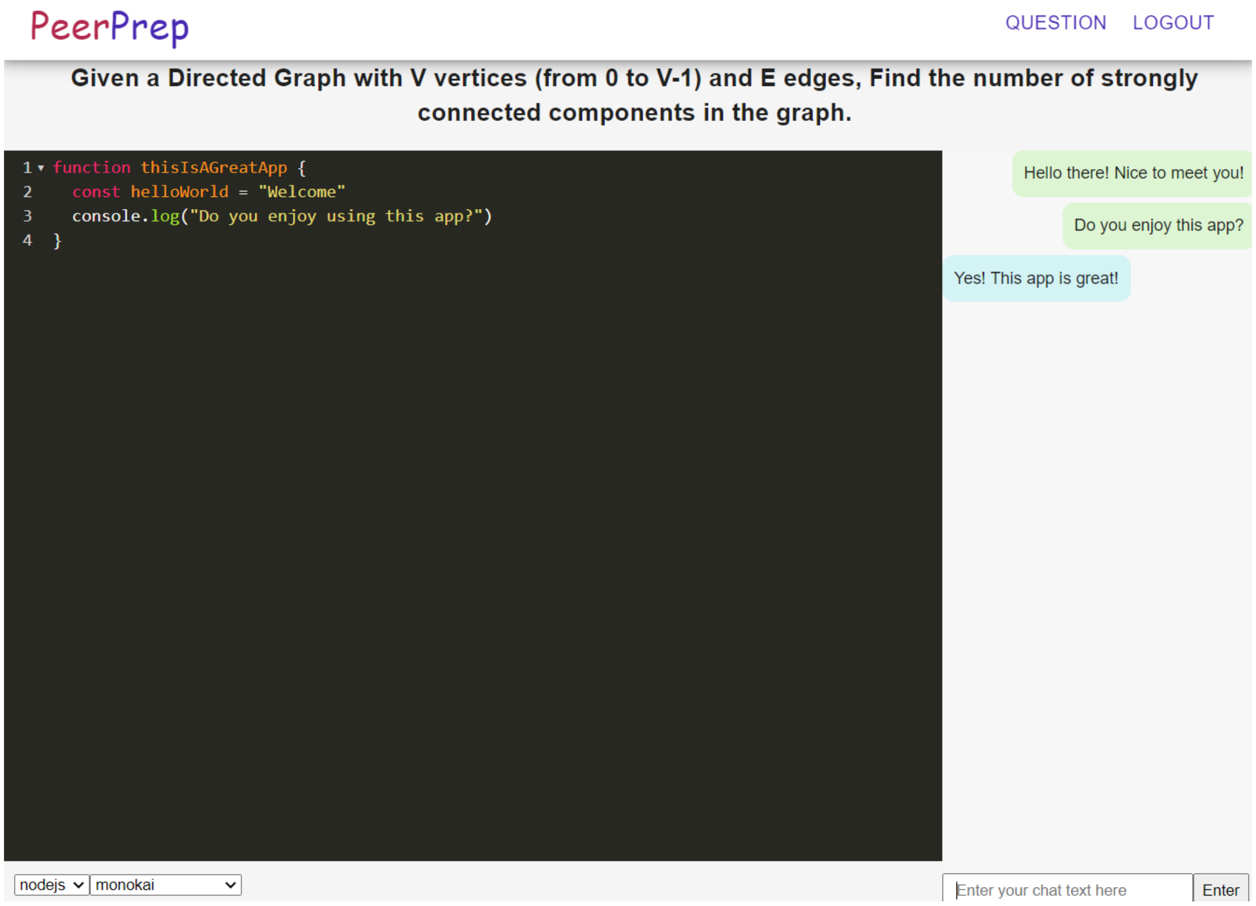
(Reset Password Page)



(Difficulty Selection Page)



(Category Selection Page)



Given a Directed Graph with V vertices (from 0 to V-1) and E edges, Find the number of strongly connected components in the graph.

```
function thisIsAGreatApp {
  const helloWorld = "Welcome"
  console.log("Do you enjoy using this app?")
}
```

Hello there! Nice to meet you!

Do you enjoy this app?

Yes! This app is great!

nodejs

monokai

(Question Room Page)