Final Report

for

Peer Prep

Prepared by
Caitlin Jee Shen-yi
Christopher Yong Wan Kang
Do Xuan Long
Oong Jie Xiang

NUS CS3219 Group 8

Table of contents

1. Introduction

1.1 Background

The increasing demand for more interactive, modern, and efficient programs and software has led to the opening of more software-related roles. Despite the demand for software engineers, potential job candidates are expected to be well versed with various aspects of computing, ranging from operating systems, databases, to front-end design and user experience. Hence, students need to prepare for many technical interviews.

To secure an internship or graduate job, university students have to equip themselves with relevant technical skills and soft skills such as communication skills. Their weaknesses include lack of communication skills, inability to understand the problem, and problem-solving. However, grinding practice questions can be tedious and monotonous.

To engage students with practicing for interviews, PeerPrep aims to build a collaborative 2-player platform for students of the same skill level to solve problems together, as well as to help students address their weaknesses by providing suitable problems that are appropriate for their skill level.

1.2 Purpose

PeerPrep is a web application that aims to match students to practice interview questions in order to reduce the monotony of preparing alone and improve their communication and technical skills. It matches currently online students that indicate the difficulty of questions they wish to attempt, and brings them to an answering interface with a free text entry and chat. The matched students can communicate with each other when answering the questions together.

PeerPrep has these main functions:

1. Match students online that indicate the same difficulty for questions they wish to attempt.
2. Provide randomized questions of the chosen difficulty to both users.
3. Provide a chat function when answering questions.
4. Display collaborative text editor for students to type their answers.

2. Individual contributions

Member	Technical contributions	Non-technical contributions
Caitlin Jee Shen-Yi	Frontend ● Login and register page design and implementation ● Integrate backend authentication (register,	Documentation ● User management sequence diagram ● Design decisions ● FR and NFR ● Improvements

	login, logout, tokens) with frontend ● Difficulty selection page design and implementation ● Integrate backend peer matching with frontend ● Write tests	● Report misc
Christopher Yong Wan Kang	● Collaborative editor design and implementation ● Chat function design and implementation ● Question panel design and implementation ● Deploy frontend ● Editor page style ● Handling sockets	Documentation ● Design decisions ● Design principles ● ER diagram
Do Xuan Long	Backend ● User management: Authentication and authorisation ● Fetching questions for each matched session and next question in the room. ● Handling sockets ● Database design	Documentation ● Activity diagrams ● Sequence diagram (Match making) ● FR and NFR
Oong Jie Xiang	Backend ● Question CRUD operations ● Peer matching ● Handling sockets at the backend for chat and editor features ● Testing	● Project management Documentation ● Component diagram ● Design patterns ● Reflections

3. Functional Requirements

S/N	Functional requirement
User management	
User registration	
F1.1	The system must allow the user to create an account on the app.
F1.2	The system must allow the user to key in the email, username and password.
F1.3	The system will return an error if the input email address or username already exists in the database, or if any of the input fields are left empty.
F1.4	After the user has successfully registered, the system will navigate the user to the login page.
Priority: High	
User login	
F2.1	The system must allow the user to login into the app after successfully registering for an account.
F2.2	The system must allow users to log in using their account's email address and password.
F2.3	If the email address and password pair input by the user are invalid, then the system shall not allow the user to proceed to the home page. The system shall prompt the user to key in the fields again.
F2.4	If the email address and password inputs are authenticated, then the system shall redirect the user to proceed to the home page and assign a valid authentication token.
Priority: High	
User logout	
F3.1	The system must allow the user to log out of the app and redirect back to the login page after successfully logging out.
Priority: High	

S/N	Functional requirement
Matching management	
Difficulty selection	
F4.1	The system must allow the user to select the chosen question difficulty (easy, medium, hard) to practice.
F4.2	Once the user selects the difficulty, the system shall start to find a online match.
Priority: High	
Peer matching	
F5.1	The system will search for another online user who has selected the same difficulty within 30 seconds.
F5.2	The system will select one of these online users with the same difficulty randomly (if present), and then redirect both users to the same answering session.
F5.3	The system must return a time out to the user if no suitable match is found after 30 seconds.
Priority: High	

S/N	Functional requirement
Room management	
Question viewing	
F6.1	After peer matching, the system must allow the user to view the same question as the matched peer, that is a question of the chosen difficulty level.
F6.2	The system must allow users to choose another question of the same difficulty level to appear for both users.
Priority: High	
Near real-time updating text field	
F7.1	The system must allow the user to see any changes on the collaborative text field, the peer must also be able to see the user's updates in near real-time.
F7.2	The system must allow users to bold, italicize, underline and input tabs and paragraphs in the editor.
Priority: High	

Chat function	
F8.1	The system must allow users to send and receive messages with alphanumeric letters to each other when both users are matched and brought into the answering session page.
F8.2	The system shall not save the chat history in the answering session when the user ends the answering session.
Priority: High	
Question completion	
F9.1	The system must allow the user to leave an answering session to choose another question difficulty.
Priority: High	

4. Non-functional Requirements

S/N	Non-functional requirements
Performance	
NF1.1	The system shall authenticate the user with a delay of no more than 5 seconds when they login.
NF1.2	Each page of the system shall load and respond quickly with a loading time of maximum 5 seconds if the user has a stable internet connection.
NF1.3	The delay time between text syncing of the free text entry in the answering session page for users who have been matched shall be no more than 1 second.
Priority: High Reason: The user should not feel a significant delay when they are using the system, especially when using the collaborative text editor and chat, which may hinder their experience while using the system.	
Security	
NF2.1	The user’s password shall be encrypted with a hashing algorithm before being stored in the database.
NF2.2	The system's features will only be available to authorized users who have logged in and who have a valid authentication token issued by the system.
NF2.3	The system shall not allow any unauthorized addition, deletion, or

	modification of user data.
Priority: High Reason: As the user will interact with another peer to practice questions together, they must be authenticated before being able to use the application. Additionally, the user should feel secure that their data is sufficiently encrypted and will not be leaked to or modified by users with malicious intentions.	
Usability	
NF3.1	More than 80% of the users must be able to click from login to answering session within 3 minutes.
NF3.2	The system shall display informative feedback for users.
Priority: Medium Reason: Because when coming to use this app, users should have had some basic knowledge about computers because it supports users who would like to find tech jobs.	

4.1 How the application achieves the chosen non-functional requirements

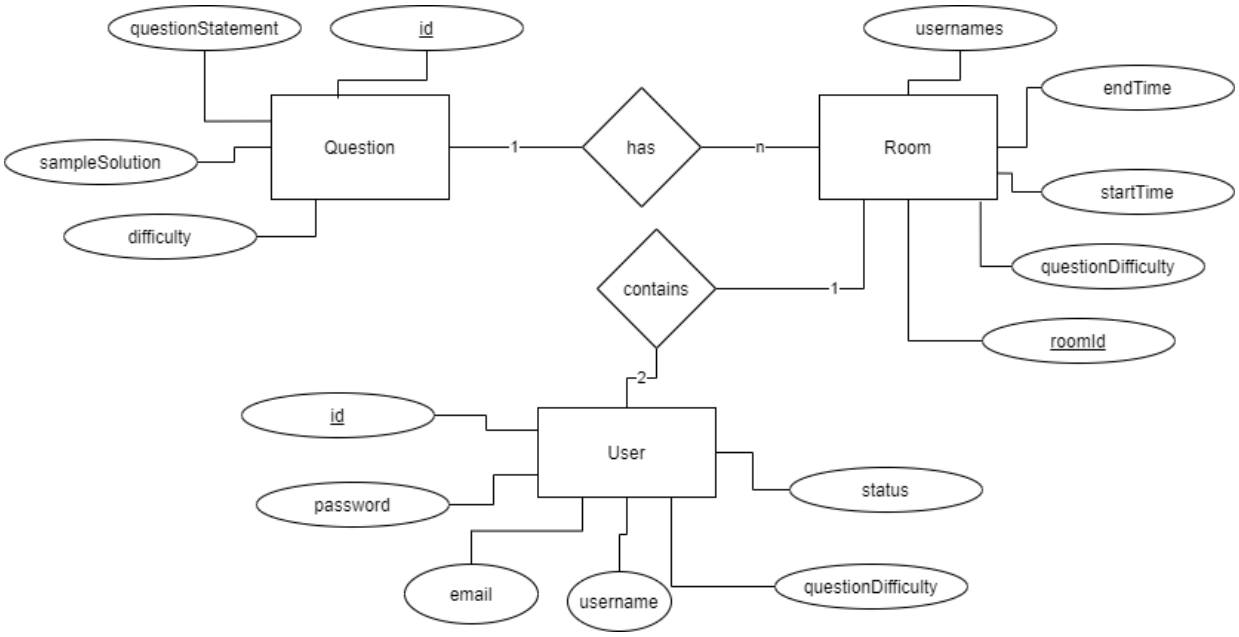
1. Security:
    - Users can only use the features if they have logged in with a valid account and have been assigned a valid authorization token.
    - Users who have not logged in are unable to access the application's features.

2. Performance:
    - When a user clicks the 'Send' button to send a chat message, the message is received on the peer's web application in less than 1 second.
    - When a user types on the collaborative editor section, the letter is displayed on the other peer's end in less than 1 second.
3. Usability:
    - The application's UI is intuitive and provides informative feedback to the user if there are any errors, for example, the Register page returns a "passwords do not match" error if the user's password and confirmation password do not match.

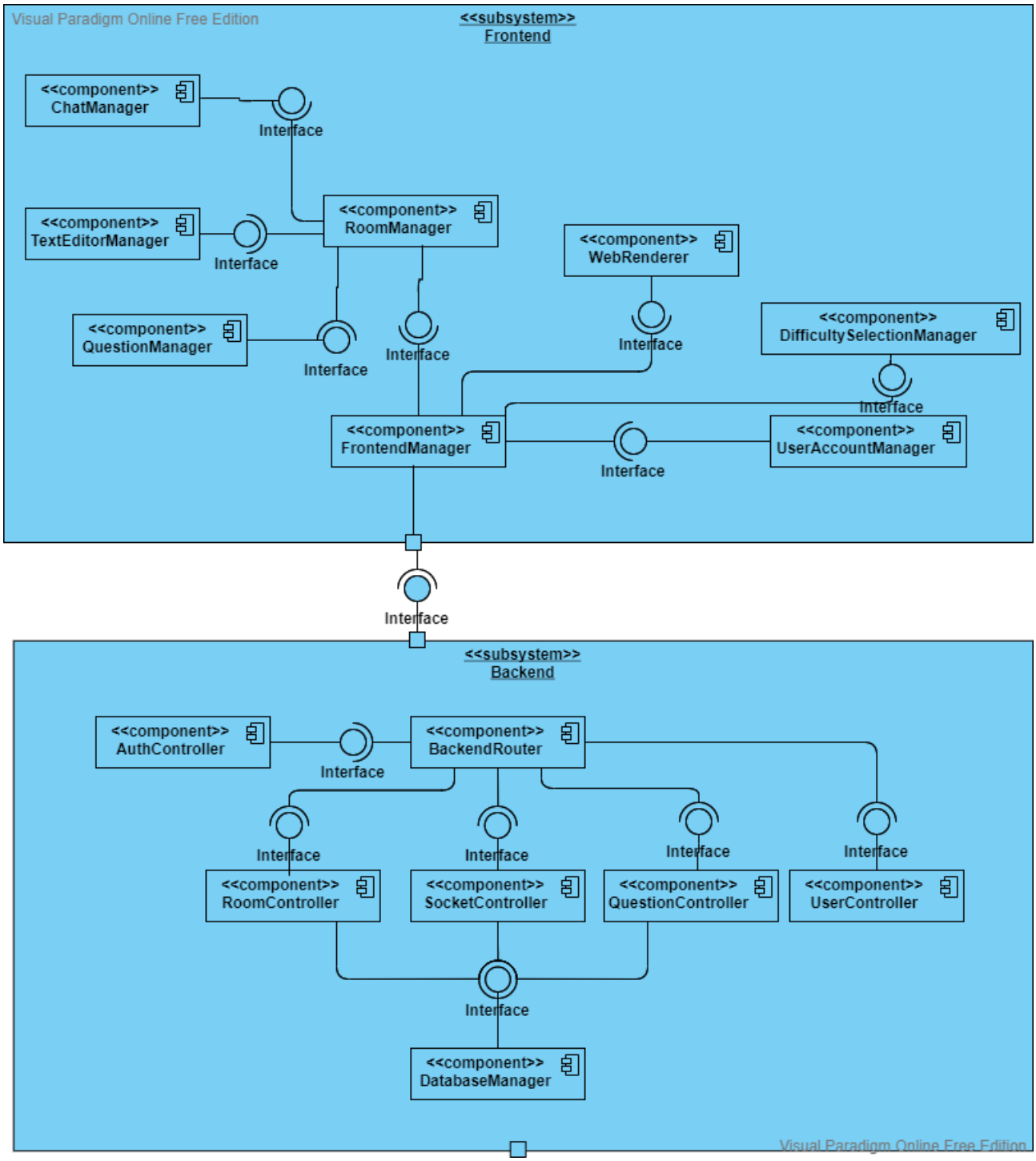
5. Developer documentation

5.1 Design diagrams

5.1.1 ER diagram

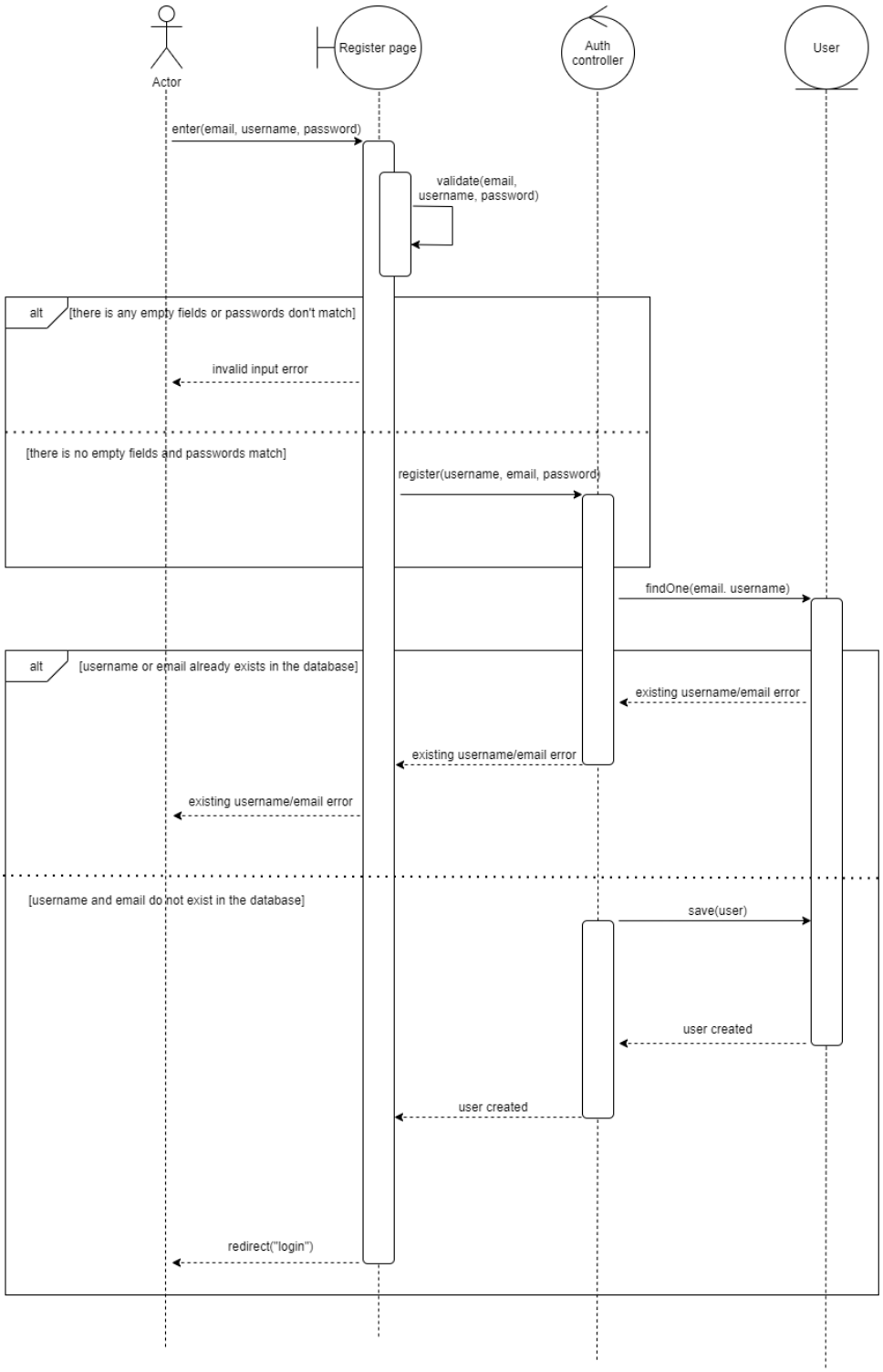


5.1.2 Component diagram

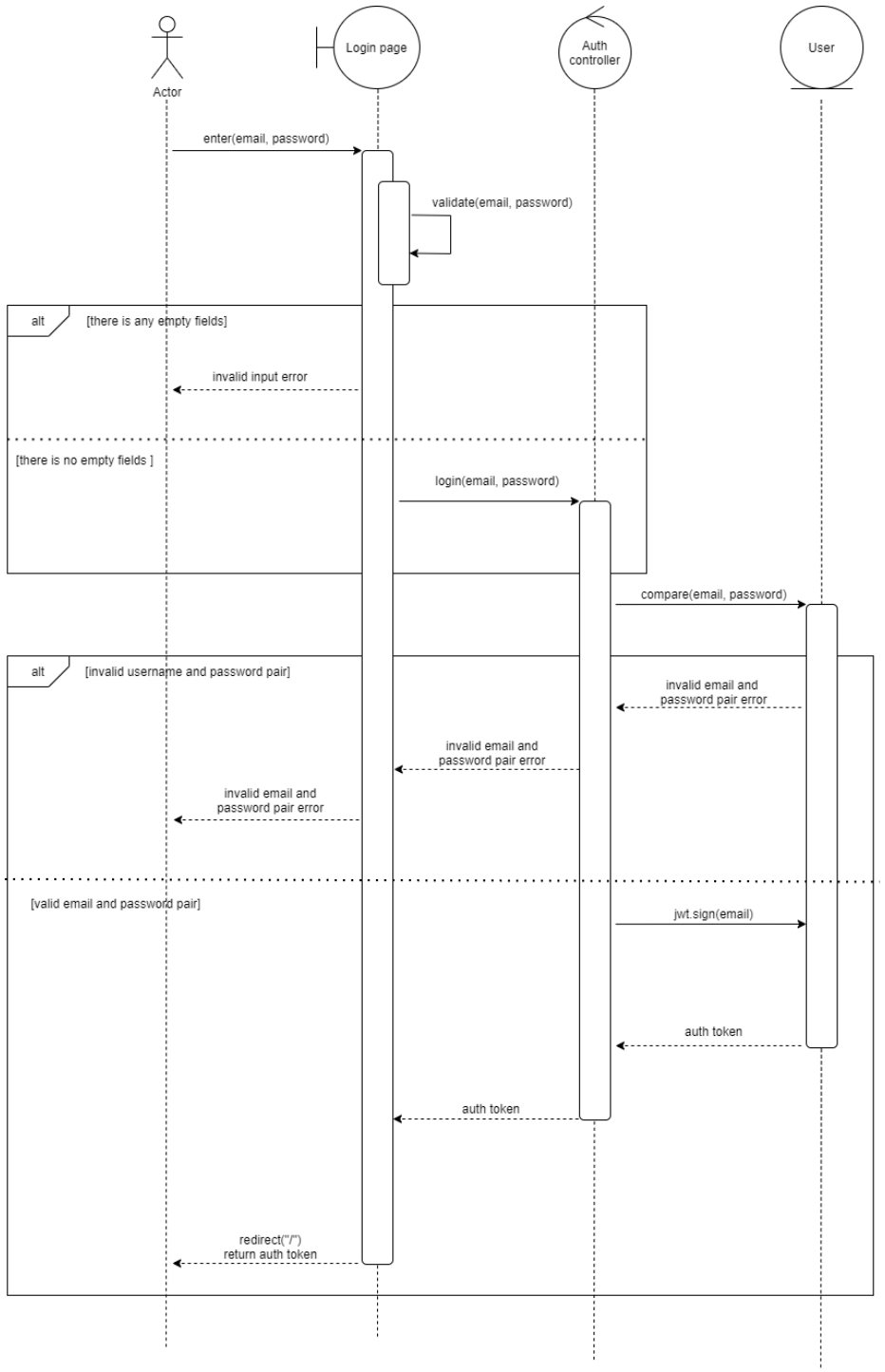


5.1.3 Sequence diagrams

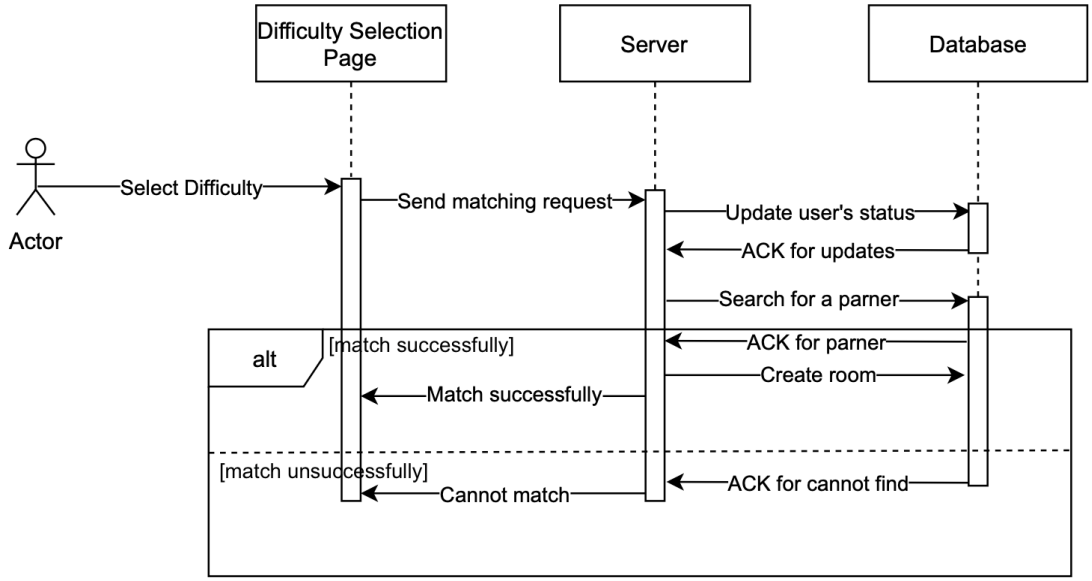
Register



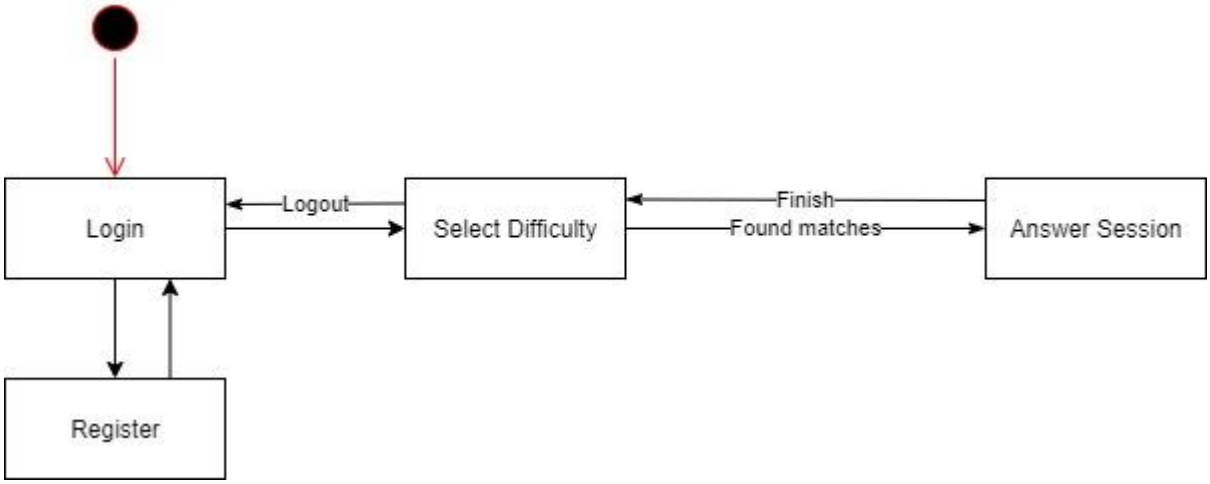
Login


Actor
Login page
Auth controller
User
enter(email, password)
validate(email, password)
alt
[there is any empty fields]
invalid input error
[there is no empty fields]
login(email, password)
compare(email, password)
alt
[invalid username and password pair]
invalid email and password pair error
invalid email and password pair error
invalid email and password pair error
[valid email and password pair]
jwt.sign(email)
auth token
auth token
redirect("/")
return auth token

Match making

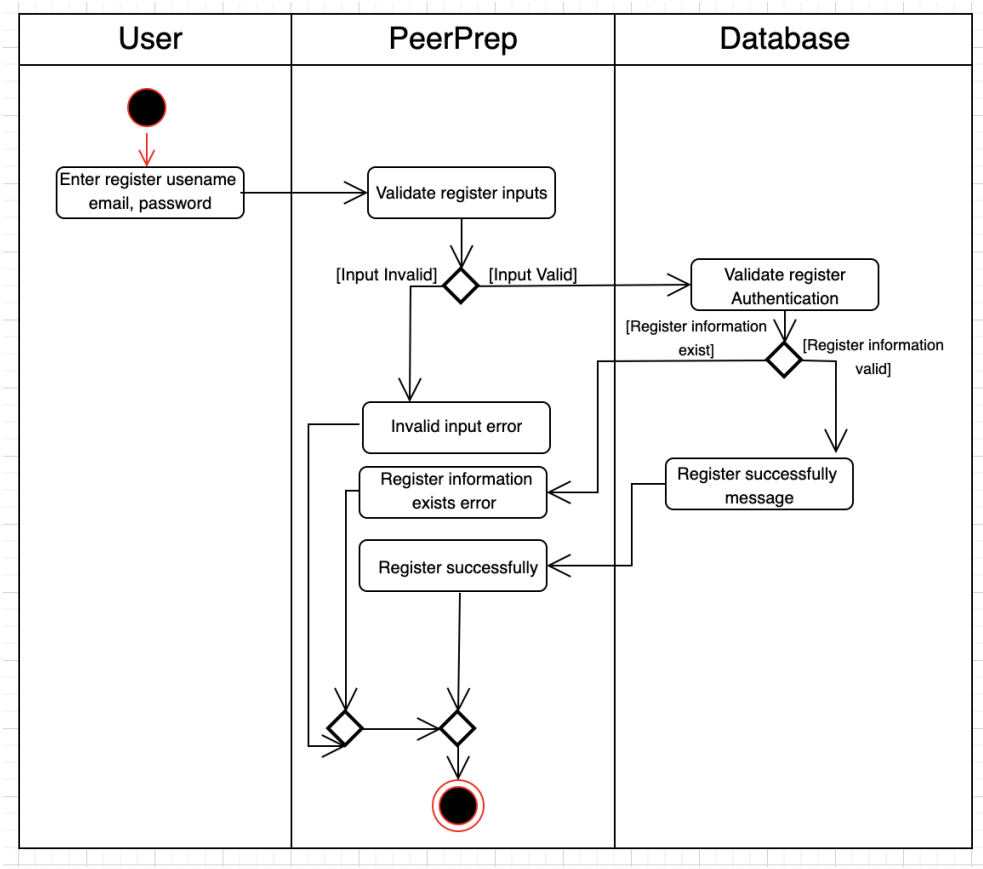


5.1.4 Dialog map

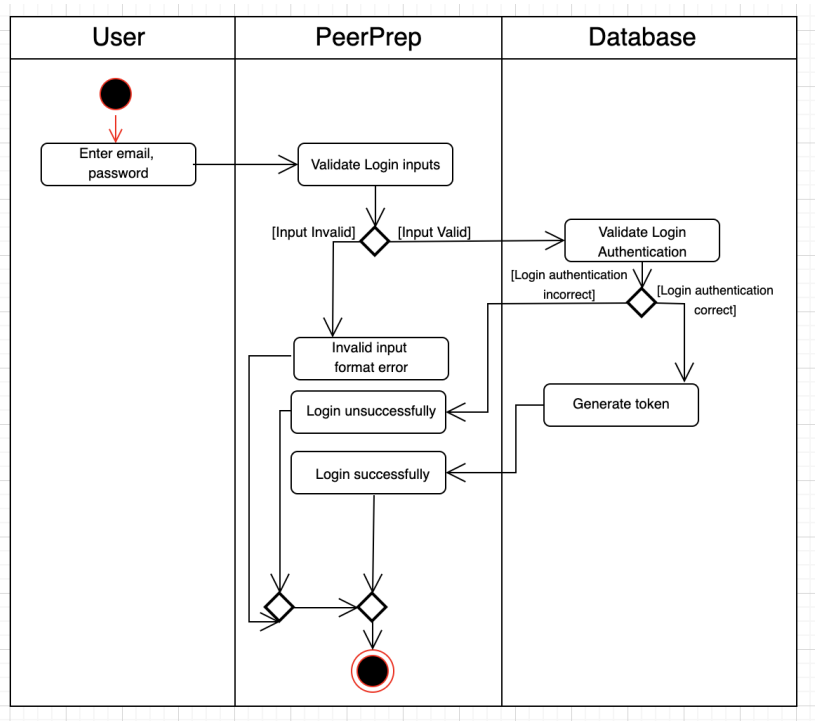


5.1.5 Activity diagrams

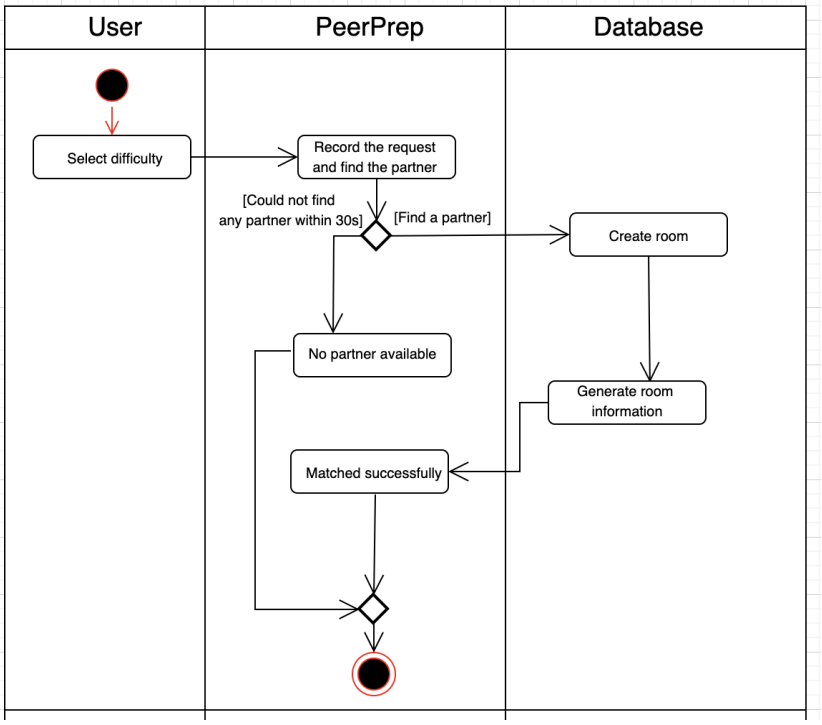
Register



Login



Match making



5.2 Project timeline

Week	Date	Tasks completed
3	28/8/2021	● Decide on project management tool ● Decide on software development process
4	5/9/2021	● Draft out functional requirements and non-functional requirements
5	11/9/2021	● Decide on tech stack ● Decide on software architecture
6	15/9/2021	● Research on chosen tech stack ● Complete requirement report
Recess	22/9/2021	● Frontend: ○ Search for library for real-time text editor and test ○ Design prototype for each page ● Backend: ○ Familiarise with Express/Node and Socket.io
7	2/10/2021	● Frontend: ○ Complete register and login UI ○ Implement real-time text editor ○ Test Socket.io ● Backend: ○ Implement register and login api ○ Implement questions api
8	9/10/2021	● Frontend: ○ Integrate question api with frontend ○ Integrate register and login api with frontend ● Backend: ○ Set up Travis CI workflow
9	16/10/2021	● Frontend: ○ Fix user authentication bugs ○ Complete difficulty selection UI ○ Implement chat function ● Backend: ○ Implement room matching api ● Deployment: ○ Replace Travis CI with Github Action ○ Deploy backend on Heroku

10	19/10/2021	● Frontend: ○ Integrate room matching api with frontend ○ Complete question viewing page ● Backend: ○ Fix room matching api bugs ○ Creating sockets with Socket.io
	23/10/2021	● Frontend: ○ Connect collaborative editor ○ Add tests ● Backend: ○ Connect room matching api ● Documentation: ○ Lay out structure of the final report
11	26/10/2021	● Frontend: ○ Search for alternatives for real-time editor ○ Fix styling issues for UI (question viewing page and difficulty selection page) ● Backend: ○ Add authorization of access token for requests ○ Testing with Automerge and Redis ● Documentation: ○ Create developer diagrams
	30/10/2021	● Frontend: ○ Implement and fix real-time editor ○ Integrate access token authorization with frontend ● Backend: ○ Collaborate with frontend to implement and fix real-time editor ○ Fix get question bug to return question with correct difficulty ● Documentation: ○ Create developer diagrams ○ Draft out final report
12	3/11/2021	● Evaluate feasibility of new framework -- Yjs and brainstorm solution ● Fix remaining bugs ● Check in final report progress
	6/11/2021	● Finalise code and report ● Laying out presentation flow

5.3 Software development process

Development process chosen: Scrum methodology

- Reason:
    - We believe that Scrum methodology for PeerPrep's implementation was the best choice as we had a small team and that with an incremental approach to the implementation, we could better deliver a functioning end product in the small amount of time that we had.
- Process:
    - Weekly sprint meetings and Scrum review
        - To organize deliverables for the next sprint and to review completed work
        - Each member illustrates three aspects: finished tasks, tasks in progress, and blockers
    - Biweekly sprint meetings (if needed)
        - To update on progress and provide help if facing any difficulties
    - Project backlog
        - List of prioritized project requirements to be completed over each sprint, managed on Trello

5.4 Design decisions

1. Choice of tech stack:
    a. Frontend: React
        i. We have chosen React as the library for our frontend as we believe that it was the quickest to learn and that it has more resources online due to its large community. Additionally, we also decided to use MaterialUI to help with the frontend design as it is simple to use and the results are clean and neat. We also used Socket.io for the chat function and editor.
        ii. Alternative 1: AngularJS
            1. While AngularJS has more years of backing, it also has a steeper learning curve as compared to React. As our team has had limited frontend development experience, we have decided not to choose AngularJS.
        iii. Alternative 2: Vue.js
            1. We considered Vue.js due to its growing popularity but decided that its community was still small as compared to React, and thus there were fewer resources online for Vue.js.
    b. Backend: Node.js/ Express.js
        i. We have chosen Node.js and Express.js for the backend, after choosing React as the frontend because we believe that it is the most compatible as both use Javascript. Other than that, there are many packages for Node.js from its large community that we could use during our development. We also used Redis for the authentication tokens.
        ii. Alternative 1: Django
            1. Django provides a framework for web development. Although both Django and Node/Express.js use Model-View-Controller (or Model-View-View-Controller for Django) architecture, Node.js provides us

with more refined control. To develop deeper insights in backend web development, instead of using Django, which handles many operations behind the scenes, we find Node.js more suitable to the group's learning outcome.

c. Database: MongoDB
    i. We have chosen MongoDB as our database because it is a NoSQL database. We expected that our database schema would have changed over the course of our implementation, and thus chose not to use a relational database, instead of going with NoSQL as it is more flexible and we could update it as our requirements changed. As MongoDB is the most popular NoSQL with the largest community, we decided to choose it as our database.
    ii. Alternative 1: PostgreSQL
        1. As explained above, we did not expect our database schema to be static and thus chose not to use PostgreSQL. Additionally, we believed that it might be harder to use as it is not as flexible as MongoDB.
    iii. Alternative 2: Firebase
        1. We found that Firebase will charge beyond a certain limit. To avoid that, we do not consider it.

We have chosen the above combination of technology used for our full-stack development as we have also considered that the MERN stack is already widely used and is still growing in popularity, adding to the already large community using this combination. Thus, we could also familiarise ourselves more with this stack for future use in the industry.

2. Choice of platforms for CI/CD:
    a. CI platform: Github actions
        i. GitHub Action was first introduced in 2018, later than other CI platforms. Since the group uses GitHub, it is more convenient to configure CI at GitHub instead of a third-party CI. Additionally, with growing popularity, reusable 'Action', and clear documentation, the group estimates that GitHub Action will be used more frequently in the near future. While we initially chose to explore Travis CI, the credit limit in Travis CI (elaborated in the next section), combined with this observation, drove us to explore GitHub Action instead.
        ii. Alternative 1: Travis CI
            1. While we had initially decided on Travis CI from our experience with it from the OTOT tasks, we decided to switch from Travis CI due to the limited credits. As the CS3219's organization has many repositories, the credits were exhausted over time as many teams were involved. Additionally, the repository must be made public, but our repository was configured to be private. So, after searching for a CI platform with unlimited credits for repositories, we found that Github actions to be the most optimal.
    b. CD platform: Heroku Free Tier
        i. It is simple to deploy an application to Heroku one just one server (termed 'dyno'). Additionally, as other CD platforms such as AWS and GCP will charge for their use,

we decided to use Heroku's Free Tier as we felt it was the safest choice for our constant deployment, without being limited by any charges.

ii. Alternative 1: AWS
    1. We have decided not to use AWS for continuous deployment as we are concerned about the charges, potentially going over the free trial charge limit and being charged while testing and debugging the application.

iii. Alternative 2: GCP
    1. Although the school has given us credits to use for GCP, similarly we are worried about the charges and that we may be limited by the credits given, potentially running out of the credits while testing and debugging the application.

3. Choice of a monolith or microservice architecture:
    a. Current design: Monolith
        i. We believe that PeerPrep's complexity and functionality are moderately small and that a monolith architecture would be sufficient for it, as opposed to a microservice. Additionally, all of our team has had experience implementing monolith applications, so it would be intuitive for us. However, we also considered that a monolith architecture would lead to higher coupling.
    b. Alternative: Microservice
        i. Microservices work best for large-scale applications. It requires more effort on container orchestration and hosting applications. It would be overfitting for a small-scale application like PeerPrep.

In the end, we decided on using a monolith architecture for the application, after considering the above pros and cons. We decided on the MVC pattern as we had experience using the pattern previously.

4. Decision to return questions that have not been attempted before by both users:
    a. Current design: Choose a question that both users have not attempted before when both users have entered a room
        i. We have decided on this approach as we believe that it increases usability, and it will be a worse user experience if users repeatedly encounter the same questions to attempt. Furthermore, we believe that our question database will be populated with enough questions such that it will take a while before users will run out of questions to attempt. Only if both users were to have attempted all questions already, then only the system will return a random question of the chosen difficulty to the users,
    b. Alternative 1: Choose a random question when both users have entered a room, regardless of if they have completed it before
        i. While this design would be easier to implement as we do not have to keep track of attempted questions for each user, we have decided to not go with this approach as we believe that it will lead to a worse experience for the user and less effective practice if they encounter the same questions while using the application.

5. Behavior of room after one user has left the room:

    a. One or all of the users disconnect.
        i. Current design: The user is able to connect back by reloading the page
            1. This is crucial to have in the case the user may not disconnect voluntarily, for example, the internet connection is broken.
        ii. Alternative: Both users are redirected to the difficulty selection page
            1. As opposed to one of the users voluntarily finishing the answering session, we considered disconnection issues and decided that it was lead to a worse user experience if the other user was redirected from the room abruptly if the session wasn't finished
    b. One of the users clicks the Finish button.
        i. Current design: Both of the users will be redirected back to the difficulty selection page
            1. This gives the user an option to end the session at their will. The other user will also be redirected back because PeerPrep aims to offer a pair programming service instead of a single-player online code editor.
        ii. Alternative: The remaining user stays in the room
            1. As opposed to if a user accidentally disconnected, we decided that, if the remaining user stays in the room when the other user has chosen to leave the answering session, this would lead to a worse user experience as the user who has left would most probably not return to the room, unlike if they disconnected accidentally

6. Functionality to show next question during answering session:
    a. Current design: Added functionality to show another question during answering session by clicking the "Next" button
        i. We have decided to add this functionality to let the user choose whether they want to have another question to practice on or not. We believe that this increases usability, as users may want to attempt a question different from the one they have gotten, or they would like to practice additional questions.
    b. Alternative: The users can only attempt one question per answering session
        i. We decided that this approach would lead to less usability, as users may want to practice more with the same peer, or they would like to try out other questions to have more practice.

7. Implementation of the collaborative editor:
    a. Initial design:
        i. Initially, we used Draft.js as our selected editor choice and socket.io to handle the syncing of editor state.
    b. Challenges
        i. The editor works fine under an isolated state, without connecting to the socket. After it connects with the socket, it also works fine if there is no simultaneous typing. However, when there is simultaneous typing, the cursor will move irregularly, like two steps ahead and return to the beginning position after some time. (As there is no typing timeout, every character typed is sent to the server immediately, so we do not

have to handle the merging problem at that stage.) We explored many options listed below:

1. Rewrite the editor component in functional component
2. Using Redux
3. Add typing timeout
    a. We have to handle merging problem by this
4. Attempt to use Automerge (CRDT)
    a. Unable to proceed because Draft.js doesn't provide API to capture or categorize internal state change
5. Handle only changes sent back from the server, instead of changing the editor directly. This means that the changes you made on your side still have to go through the server to take effect.
6. Save the cursor position as component state and feed it to Draft.js manually.

After going through all the attempts above, we still could not solve the cursor bug. There are several reasons:

1. Draft.js is not made for collaboration purposes. Looking at the Github issue, the developers stated that they have no intent to do so either.
2. The internal state representation of Draft.js state is very complex and hard to interpret.

c. Solution and current design:
    i. After researching, we have decided to switch to Quill.js which has built-in CRDT with slight modification with our original socket.io js.
        1. The editor works perfectly in localhost. The cursor bug does not appear anymore.
        2. However, after the application is deployed in heroku, there is some structure misform under simultaneous typing. We suspect that it may be because of latency issues after being deployed.
d. Alternative design: Using Yjs, another CRDT implementation that is more suitable in shared editing applications.
    i. After attempting until Challenge 3 above, we resorted to CRDTs as implementation of editing conflicts is infeasible (decades of research are still conducted on this matter). Automerge was our first consideration.
    ii. However, while Automerge is simple, it is suitable for shared application states such as Kanban boards. To apply it in collaborative editing, the exact operation performed on the text must be known. As Draft.js was used at that time, Automerge was deemed infeasible.
    iii. Next, although Quill's CRDT implementation was not as complete as Yjs, it is still able to handle concurrent changes when there is little latency in the server. Implementing Quill as the default implementation for this project allowed us to explore the third major implementation: Yjs.

iv. Yjs is built precisely to address shared editing applications, and therefore surpasses Automerge's capabilities in this domain. However, there were a few constraints:
    1. Implementation resources needed: Yjs extends many other packages for implementation. For example, it uses Y-Level DB (an IndexedDB) to persist data, and WebSockets or Socket.io to create client-server architecture. This creates the necessity to understand several concepts and ensuring that all dependencies work. During the project, Yjs' implementation of Level DB through 'y-memory' package is faulty. While this might be machine dependent, the stability and need to learn all packages become a concern.
    2. Development resources needed: Despite the team's efforts in researching on collaborative text editing issue, we only discovered Yjs in Week 11. This provided insufficient time to explore the vast concepts required to use Yjs.
v. Due to implementation and development resource constraints, the team decided to use Quill as the more stable version of our PeerPrep collaborative text editor.

5.5 Use of design principles and patterns

Design Principles

1. **Single Responsibility Principle**
   When we are writing frontend code, we decided to put all the API calls into one module. This follows the Single Responsibility Principle (SRP). This brings multiple advantages. When we need to change the base url, it can be done at only one place. An example is switching between localhost and cloud (Heroku). In addition, it keeps the code cleaner, without being filled by repetitive request code, following the Do not Repeat Yourself (DRY) principle.

   Another demonstration of SRP principle is that all the components in one editor page are handled by different modules. For example, the three main components in editor pages are handled by different modules, text-editor.js, question.js and chat.js.

2. **Choice in React Framework**
   In terms of choosing between functional components and class components, we decided to use class components for the parent component that is responsible to render the entire editor page. This is because class components allow easier and cleaner constructor, which is important for us to set up the socket connection and events, ensuring there is only one connection per client.

Design Patterns

1. **Model-View-Controller Architecture: Web-SPA (Single Page Application)**
   The React frontend uses Single Page Application framework. Naturally, Node.js server becomes a HTTP server that communicates in JSON format. To promote parallel development and modularity, both teams organised the code into folders. For instance, backend teams' code is separated into

‘models’, ‘controllers’, ‘middleware’ and ‘db’. In this case, a change in a ‘models’ file will cause minimal changes in ‘controllers’ folder

2. **Singleton Pattern**
   A queue is implemented to match peers when they request for a particular difficulty. As users can request for a match at any time, there must only be one queue for consistency. This is realised in matchMaking.js and roomController.js files. To ensure that only one queue and one match maker (denoted as roomManager, an object in roomController.js) are created, we instantiated one match maker on server start-up and exposed it to all room-related APIs.

3. **Observer Pattern**
   Although not explicitly implemented, the usage of Socket.io illustrates the importance of low coupling in the publish-subscribe paradigm. This is evident when the server has to authorise the user who connects to a specific room ID. Once the user is authorised by referring to the database, the system will group the user under the room ID generated. For subsequent communication, the backend server will transmit information to all receivers. This helps in scalability: although we are limiting two users per room, allowing more users per room requires little modification as the backend server can simply group more users under the same room ID, and then broadcast communication to all relevant receivers (in the same group).

6. Improvements and enhancements to be made

1. Improvement of UI
   a. Various improvements to the UI could be made in future iterations, for example making it more intuitive and pleasing, as the current UI’s design is rather simple. We focused on functionality rather than design so we could improve more on this.
2. Ability for users to add their own questions
   a. We understand that, although we have populated the question database, there is still the possibility of users running out of questions or getting repeated questions. Thus, one improvement could be to have a page to allow users to add questions of a chosen difficulty. This way, users can have more questions to practice with less possibility of getting repeated questions.
3. Less latency in updating collaborative editor
   a. Collaborative editing faces challenges of merge conflicts. Traditionally, Operational Transformation (OT) concept is quick and requires a central server. In recent years however, Conflict-free Replicated Data Types (CRDT) have become more popular. In the CRDT concept itself, there are Automerge and Yjs packages. After experimenting with Automerge and Yjs, we find that Yjs might be more suitable for collaborative apps. Hence, a natural step further is to apply Yjs in the collaborative editor.
4. Separate services and change to microservice architecture
   a. The current monolithic architecture makes it hard to isolate components. If the application were to have more functionality or more users in the future, it would be more optimal to refactor it into a microservice architecture. Thus, we could also more easily scale the application if needed.

7. Reflections and Learning Points

This project has provided us the opportunity to practise CI/CD, project management, and exploring different tech stacks and current frameworks for collaborative editing and real-time communication. Abstracting the lessons, however, there are three key reflections.

1. **Constant alignment on requirements is needed to improve productivity**

   In this project, we first aligned on the functional and non-functional requirements, determined the architecture and chose the technology stack. Inevitably, there are some challenges during the implementation. For instance, we thought of a slightly different architecture that could enhance our components' maintainability and modularity. Additionally, the sudden depletion of credits on Travis CI caused us to switch to another CI platform. While working through the project, we also had more ideas for the features we intend to deliver

   However, just like in a production setting, we should balance cost and impact. Instead of rashly changing directions, we evaluated the tasks at hand, and prioritised more impactful tasks. This allows us to stay focused on this project, while noting down the technical (software patterns) and non-technical (project management) considerations. Considering that this project requires team effort, we also aligned constantly within the team. The product is increased productivity.

2. **Software projects should only be implemented when extensive research is performed**

   Issues such as introducing new functionalities or using the most suitable technology or framework arose in this project. Notably, the collaborative editor shows us the importance of surveying for frameworks extensively. We first encountered Automerge (CRDT), then Quill, and finally Yjs with Quill, which took a few weeks. However, careful surveying by weighing the pros and cons of each framework and prototyping it will save us more time and improve the robustness of the system. This incident shows us that complete planning that takes slightly more time is counter-intuitively more time-saving than early implementation.

3. **Software patterns and principles improve team performance**

   To encourage parallel development and testing, each member has to work on a particular function. However, this would not be achieved had we not applied software patterns and principles. By applying core principles -- modularity, encapsulation and Do not Repeat Yourself (DRY), we are able to trace bugs more effectively. This yields us more time to deliver features.

   It is also worth mentioning that software development styles also play a role in a performing team. For instance, the CI/CI pipeline taught in this course convinced us that automated development processes saves work and simplifies communication. In an overview, a project is well delivered when the environment is conducive for members to focus mostly on delivering features and requirements.

8. Appendix

Appendix A: Glossary

Term	Description
System	This includes both the front-end for the user interface and the back-end for the logic, which is hidden from the user of PeerPrep application.
Database	An organized collection of structured information, or data, consisting of the user's data and a list of questions.
Difficulty	The difficulty level of the question that the user may select, i.e Easy, Medium, Hard
Matching	The process of matching the user with another online user who has selected the same level of question difficulty to work on a question in the same answering session.
User	The person that uses the PeerPrep application.
Peer	Another online user that has been matched with the current user.
Answering session	The duration when two online users are in the same room and collaborating to answer the chosen question on the same collaborative editor.
Collaborative editor	The real-time updating text field that updates for both users when one user makes a change on the editor.