CS3219 - Software Engineering Principles and Patterns

Team 37 Project Report

Team Members	Matric Number	Email
Dora Heng Zhi Qin	A0205773A	e0425696@u.nus.edu
Ang Chew Chern Nicole	A0205250X	e0425173@u.nus.edu
Shen Yang	A0205725J	e0425648@u.nus.edu
Toh Sihui	A0205472L	e0425395@u.nus.edu

Table of Contents

1 Background

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

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

A student who is keen to prepare for his technical interviews visits the site. He creates an account and then logs in. After logging in, the student selects the question difficulty level he wants to attempt today (easy, medium, or hard). The student then waits until he is matched with another online student who has selected the same difficulty level as him. If he is not successfully matched after 30 seconds, he times out. If he is successfully matched, the student is provided with the question of the selected difficulty, assigned the role of interviewee/interviewer randomly, a chat, and a free text field. The text field should be read only for the interviewer. This free text field should be updated in near-real time, allowing the student and his matched peer to collaborate on the provided question. After the interviewee student finishes working on the question and is ready to end the session, clicking on a “Finish” button brings the students to the session ended screen where after 10 seconds, they start working on another problem, but with reversed roles. This continues until both users have attempted each role, then they will be brought back to the Home page. From this page, the student logs out.

2 Purpose of Project

The purpose is to create a web application that helps students better prepare themselves for technical interviews. This can be achieved in many ways. For example, (i) by revising core algorithmic concepts with other students in a unique and effective way OR (ii) by using a peer learning system such as the one proposed above so students can learn from each other and break the monotony of revising alone. For this project, we shall use method (ii). Not only does the application serve as a peer learning system, the system also simulates the process of a technical interview so that students can practice.

4 Functional Requirements

4.1 User service

S/N	Functional Requirement	Priority	Schedule
FR1.1	The system should allow users to create	High	Week 4

	an account with username and password.		
FR1.2	The system should ensure that every account created has a unique username.	High	Week 4
FR1.3	The system should allow users to log into their accounts by entering their username and password.	High	Week 5
FR1.4	The system should allow users to log out of their account.	High	Week 6
FR1.5	The system should allow users to delete their account.	High	Week 6
FR1.6	The system should allow users to change their password.	Medium	Week 6
FR1.7	The system should allow users to reset password by providing email	Medium	Week 7
FR1.8	The system should allow users to create an account with email	High	Week 7
FR1.9	The system should reject invalid passwords based on password length and strength restrictions	High	Week 8

4.2 Matching service

S/N	Functional Requirement	Priority	Schedule

FR2.1	The system should allow users to select the difficulty level of the questions they wish to attempt.	High	Week 4
FR2.2	The system should be able to match two waiting users with similar difficulty levels and put them in the same room.	High	Week 5
FR2.3	If there is a valid match, the system should match the users within 30s.	High	Week 6
FR2.4	The system should inform the users that no match is available if a match cannot be found within 30 seconds.	High	Week 6
FR2.5	The system should remove the matched users if one of them leaves the session	High	Week 9
FR2.6	The system should remove the matched users once their session has ended	High	Week 9
FR2.7	The system should remove the user from the match database if a match is not found within 30 seconds	High	Week 7

4.3 Collaboration service

S/N	Functional Requirement	Priority	Schedule
FR3.1	Matched users should be able to see what the other user has typed in the collaboration space	High	Week 9

FR3.2	Matched users should be added to the same room space	High	Week 8
FR3.3	Matched users have limited time to collaborate based on difficulty level of questions	High	Week 8
FR3.4	After the time is up for the first interviewee, the interviewer and interviewee switch roles and a new question is given	High	Week 9
FR3.5	Matched user should be able to leave the room at any time	Medium	Week 9
FR3.6	When both users have switched roles, users can try again with the same match if both parties are willing	Low	Future extension
FR3.7	Link to the question solution should be shown by the end of the session	Low	Future extension
FR3.8	Interviewee can finish the session early if they have completed the question	Medium	Week 11
FR3.9	Interviewer should move on to the next session as well when interviewee ends the session early	Medium	Week 11

FR3.10	Collaboration session should end when one of the users leaves the session	Medium	Week 10
FR3.11	Matched users should be randomly assigned to a role when they first enter the room	Medium	Week 9
FR3.12	The system should give users 10 seconds to choose if they want to leave before the next session starts	Medium	Week 9

4.4 Question service

S/N	Functional Requirement	Priority	Schedule
FR4.1	The system should have several questions sorted by difficulty level	High	Week 9
FR4.2	The system should have several questions sorted by topic	Low	Future extension
FR4.3	The system should return a question of the correct difficulty level	High	Week 9
FR4.4	The system should not return a question previously attempted in the same room.	High	Week 9

4.5 Communication Service

S/N	Functional Requirement	Priority	Schedule

FR5.1	Users in the same room should be able to see each others chat messages	High	Week 9
FR5.2	Users should be notified when a new chat message arrives	Medium	Week 11
FR5.3	Users should be able to send chat messages to each other	High	Week 9

4.6 Frontend

S/N	Functional Requirement	Priority	Schedule
FR6.1	The system should provide a collaboration space for users	High	Week 7
FR6.2	The collaboration space should show the users in the room, user's current role, time remaining and question	High	Week 7
FR6.3	The system should provide a screen to indicate that the session has ended	High	Week 8

5 Non-Functional Requirements

S/N	Non-Functional Requirement	Quality Requirement	Priority
NFR1	A user can create an account successfully more than 90% of the time	Usability	High

NFR2	The system should be able to run locally with the native tech stack on Windows, Mac and Linux	Installability	High
NFR3	The system should be able to run as a Docker container on Windows, Mac and Linux	Installability	Medium
NFR4	The system shall backup the data typed in the collaboration space in the database to ensure information typed on both users ends will not be lost	Integrity	High
NFR5	The system shall sync the information typed in the shared collaboration space within 5 seconds	Performance	Medium
NFR6	The system should load the room within 15 seconds of users being matched	Performance	High
NFR7	The system should only allow users to access the main pages e.g. Home, MatchedRoom etc after they have logged in with a valid username and password	Security	High

6 Developer Documentation

6.1 Tech Stack

Microservice		Tech Stack	Rationale for choice
User service	Database	MongoDB	MongoDB is a document-based noSQL database. Since it is a NoSQL database management system, it provides faster reads and writes as compared to RDBMS like SQL. It is also more scalable
	Backend	NodeJS	JavaScript used for both frontend and backend development Scalable

Matching service	Database	Sequelize, SQLite	Sequelize allows writing of queries in plain JavaScript and provides a clear definition of the database schema. SQLite is lightweight and provides fast reads and writes. It is also portable and can be run across different platforms. Furthermore, no installation is required to run SQLite.
	Backend	Socket IO, NodeJS	Socket.IO allows for low-latency, bidirectional and event-based communication between a client and a server. This allows for real time communication between users, a main functionality of PeerPrep. Broadcasting can also be made to a subset of clients, appropriate for the interactions between users in their own collaboration room. It is built on top of the WebSocket protocol and provides additional guarantees like fallback to HTTP long-polling or automatic reconnection
Question service	Database	MongoDB	Similar to User Service
	Backend	NodeJS	
Collaboration service	Database	Sequelize, SQLite	Similar to Matching Service
	Backend	Socket IO, NodeJS	
Communication service	Backend	Socket IO, NodeJS	Similar to Matching Service
Frontend	React		Many libraries are available with ready made components for quicker development. In our case, we used Material UI.

6.2 Design

6.2.1 Architecture and High Level Design

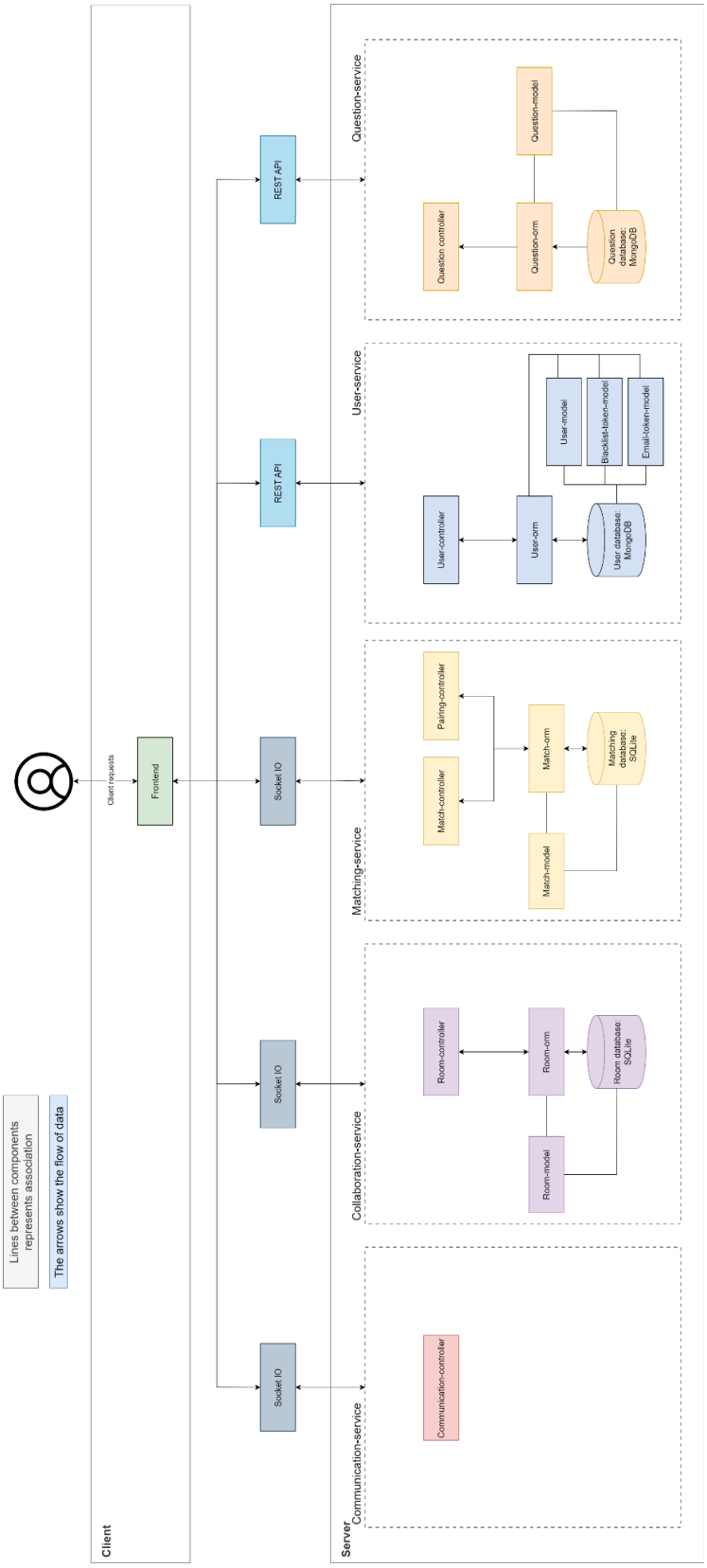


Fig 6.2.1.1 Overall architecture of PeerPrep

Our PeerPrep application adopts the **Microservice architecture**. This was chosen for **better modularity** and **agility** which allows for **ease of development**. Since the services are **loosely coupled** each person can develop a service independently, allowing for quicker development. There is also **better fault isolation**. An error in one service does not cause the whole application to fail. Lastly, there is more **flexibility** in choosing the tech stack for each service, as seen in our Tech Stack section. Rather than a shared tech stack, each service can make a choice based on what is most suitable or needed for implementing the service. The analysis of each architecture pattern in this diagram also aided us in choosing the microservices architecture. In particular, we prioritised development since this application had to be completed within a short timeframe, and testability to apply CI.

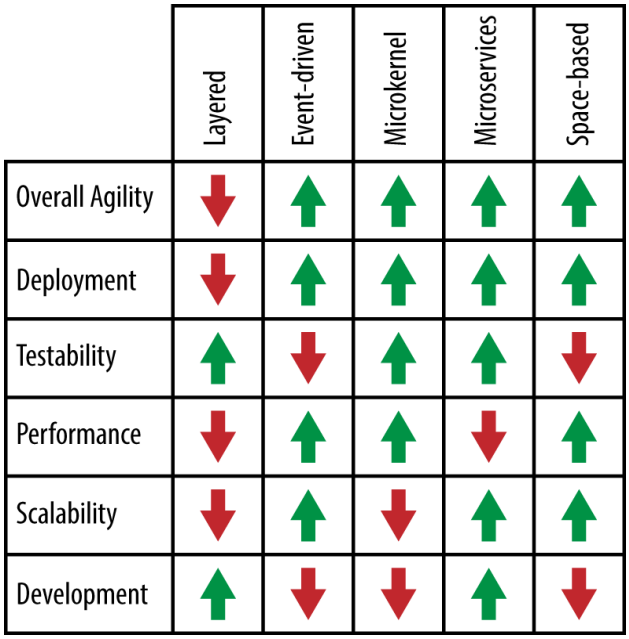


Fig 6.2.1.2: Image source

We have chosen to separate our services into: user service, matching service, collaboration service, question service and communication service. The process of analysing and identifying each of these microservices is explained in detail in the appendix.

In each service, we use the **Model-View-Adapter (MVA) architecture**. The MVA architecture applies the mediator pattern, where the controller acts as the mediator. The model and view do not communicate directly and only interact via the controller. The controller is responsible for managing the interactions between the model and view, and routes requests and events received from the view to the model.

The benefits of such an architecture is:

- **Better decoupling:** Model is only responsible for database related logic. View is only responsible for the UI logic and generating the UI for the user. Controller is the intermediary that connects View and Model. This allows for better separation of concerns and decoupling between components.
    - In Fig 6.2.1.1, in each service, components are mainly separated into controller, ORM and the database. For example, if we did not have user-

controller acting as mediator, frontend will be dependent on the User model, black list token model and email token model directly via user-orm. So any changes made in the user-orm will directly impact the frontend.

- **Easily extendable:** A new view/controller/model/functionality in a view-controller-model trio can be added without affecting other existing components.
    - We can introduce multiple views for a single model for example. As seen in the architecture diagram Fig 6.2.4.1 in collaboration-service, we have 2 frontend Views, MatchedRoom and SessionEndedPage, using functionalities from a single room-orm.
    - We can also introduce multiple controllers for a single view/model pair. For example, in the architecture diagram Fig 6.2.3.1 in matching service, we separate the logic of pairing and Match entry creation/deletion into a pairing-controller and matching-controller respectively.
    - The ORM methods and model schemas can be changed easily without requiring changes in the frontend. For example, new fields can be added to the room database of the collaboration-service without the frontend changing.

Each service has its own folder and contains the files that implement the Model and Controller components. Within each service is mainly:

- An index.js file that sets up the service by connecting to the database and initialising the server socket for the service (if applicable).
- A controller folder that contains the controller components.
- A model folder that contains the database schema and orm logic to interact with the database.

View components are grouped into the frontend component and pages folders.

6.2.2 User Service

The user service is responsible for the creation, retrieval, update and deletion of an user account in PeerPrep.

The user service interacts with the frontend through REST APIs

Architecture

This section shows the structure and logic of the user service of PeerPrep. It shows a generalised view of of user service microservice

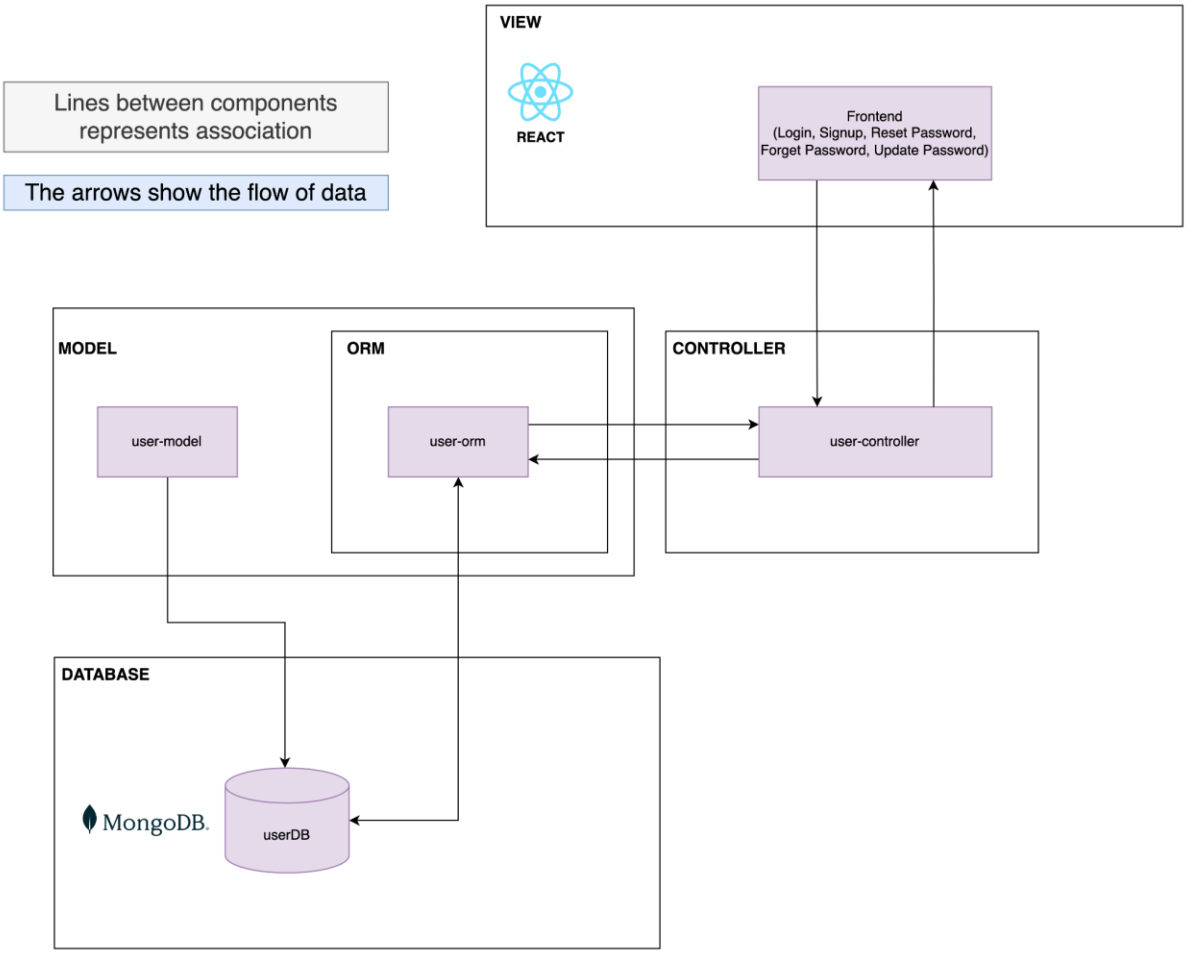


Fig 6.2.2.1: Architecture of user service

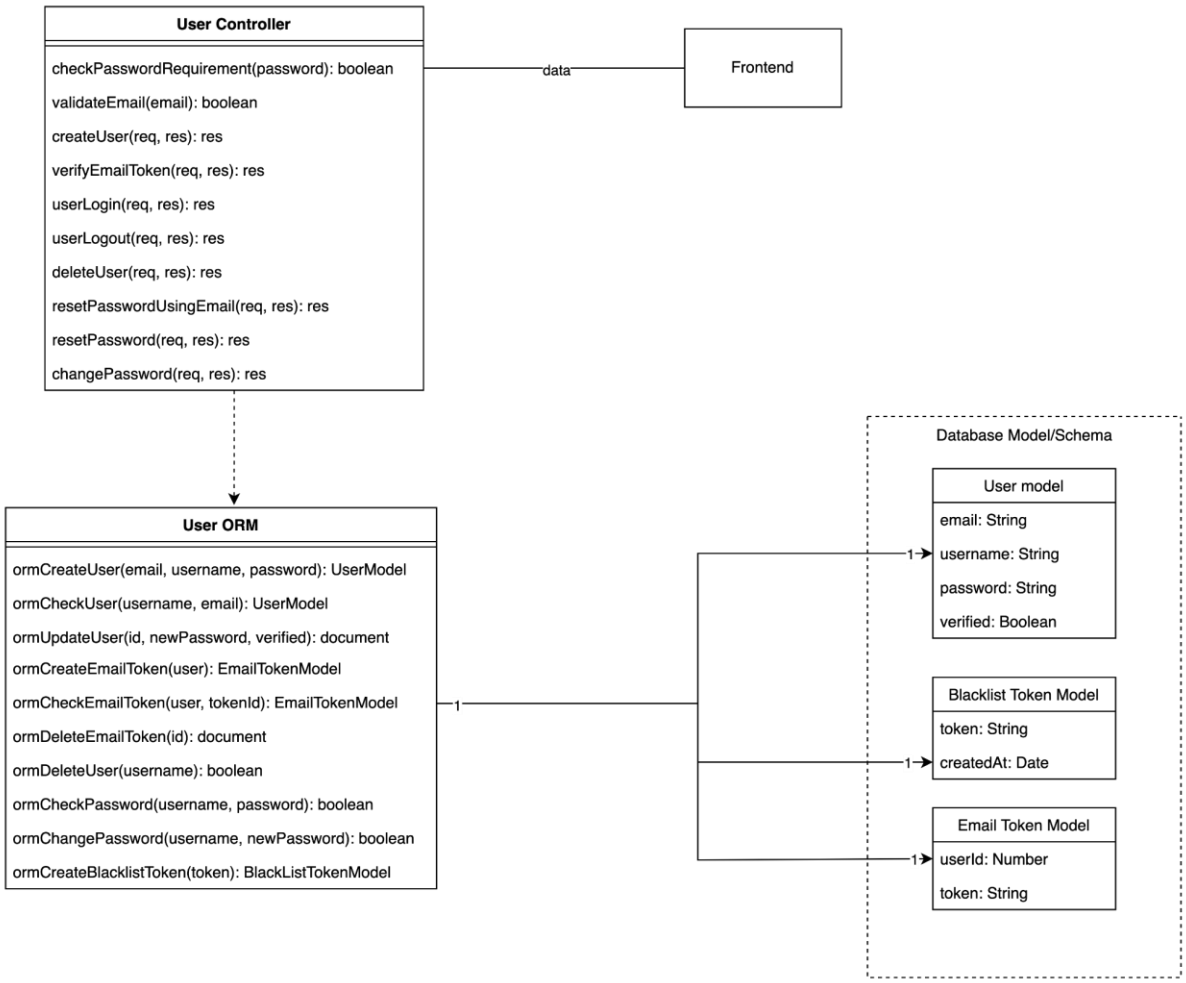


Fig 6.2.2.2: Class diagram of user service

- The `User Controller` interacts with both the `Frontend` and `User ORM` and acts as a Mediator to send data to and fro the 2 classes
- The `User ORM` interacts with the database directly and deals with the storage, retrieval, deletion and update of information of the respective database models

UserModel Database Schema

Field	Data type	Field explanation
email	string	Email of a user
username	string	Username of a user
password	string	Password of a user
verified	boolean	Verification of an account. Its default value is false.

EmailTokenModel Database Schema

Field	Data type	Field explanation

userId	objectId	Unique ID of a user
token	string	Token generated when an email is sent

BlacklistTokenModel Database Schema

Field	Data type	Field explanation
token	string	Token generated when a user logs out
createdAt	date	Time token is created, and set for the token to be deleted within 3 hours

Features aka APIs

User Service consists of 8 APIs in total that interact with the frontend. In total, there are 6 features for User Service

Create Account Feature

This section will explain the implementation of the Create Account feature. Once the user clicks on the 'Sign Up' button, the API for signing up for an account will be called and data (containing the user's email, username and password) will be sent to `user-controller#createUser` method in User Service to be parsed and validated before being stored in the database. Once the account is successfully created, an email will be generated and send through `email#sendEmail` to the user to verify their email

Below are the status codes and messages that a user will see when signing up for an account:

Status Code	Message	Explanation
201	Account Successfully Created!	
400	Missing fields!	Email, Username and/or Password are missing
400	Password does not meet requirements!	Password input does not meet requirements and therefore failed validation
400	Invalid email format!	Email format is invalid and therefore failed validation
409	Username already exist in the database!	There is already a user with the same username created
409	Email already exist in the database!	There is already a user created with that same email
500	Database failure when creating	This is most likely due to connectivity

	new account!	issue with the database or network

For better visualisation, below shows the general activity and sequence diagram to create and account:

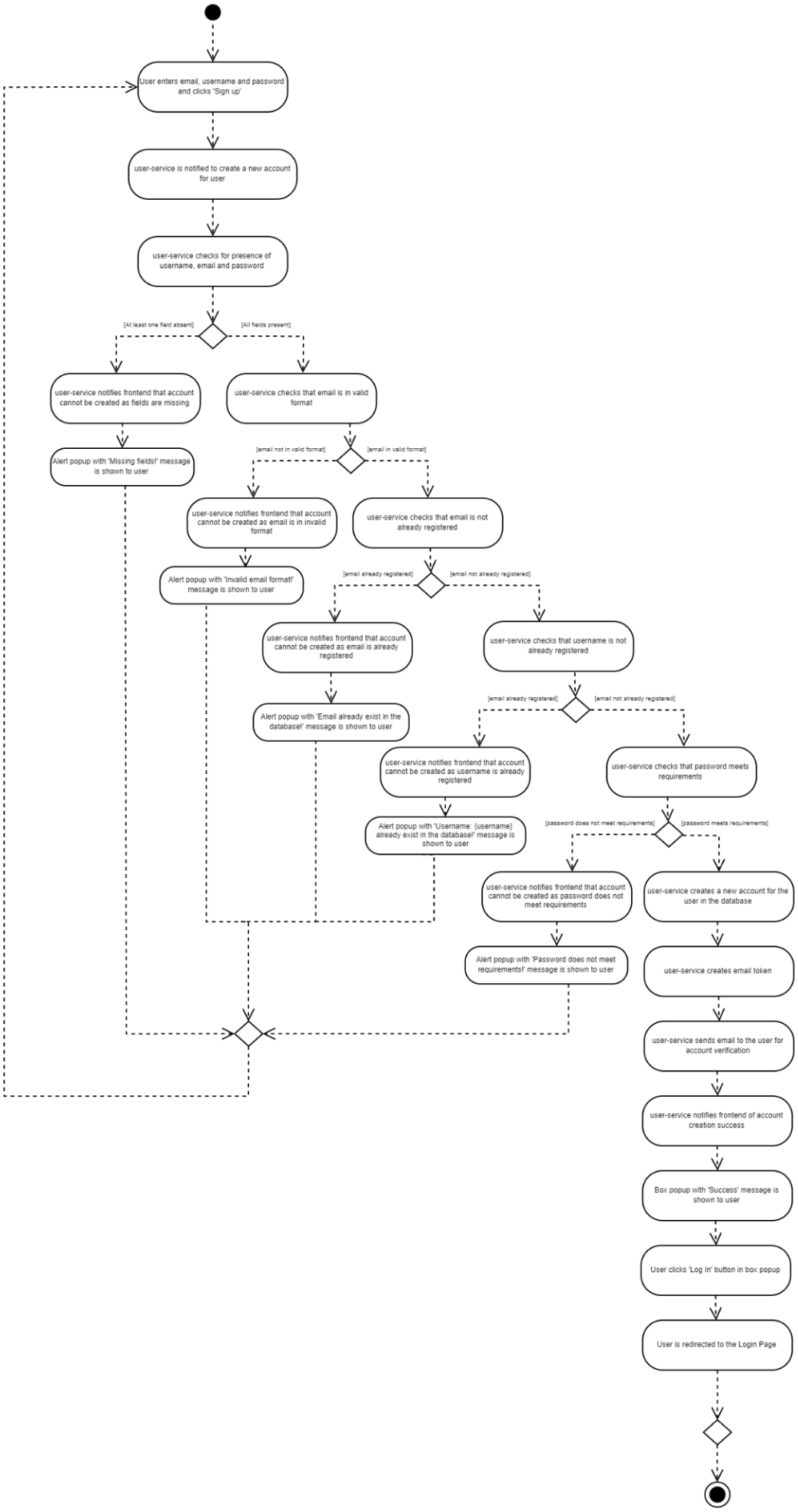

User enters email, username and password and clicks 'Sign up'
user-service is notified to create a new account for user
user-service checks for presence of username, email and password
[At least one field absent]
[All fields present]
user-service notifies frontend that account cannot be created as fields are missing
Alert popup with 'Missing fields!' message is shown to user
user-service checks that email is in valid format
[email not in valid format]
[email in valid format]
user-service notifies frontend that account cannot be created as email is in invalid format
Alert popup with 'Invalid email format!' message is shown to user
user-service checks that email is not already registered
[email already registered]
[email not already registered]
user-service notifies frontend that account cannot be created as email is already registered
Alert popup with 'Email already exist in the database!' message is shown to user
user-service checks that username is not already registered
[email already registered]
[email not already registered]
user-service notifies frontend that account cannot be created as username is already registered
Alert popup with 'Username: {username} already exist in the database!' message is shown to user
user-service checks that password meets requirements
[password does not meet requirements]
[password meets requirements]
user-service notifies frontend that account cannot be created as password does not meet requirements
Alert popup with 'Password does not meet requirements!' message is shown to user
user-service creates a new account for the user in the database
user-service creates email token
user-service sends email to the user for account verification
user-service notifies frontend of account creation success
Box popup with 'Success' message is shown to user
User clicks 'Log In' button in box popup
User is redirected to the Login Page

Fig 6.2.2.3: Account creation activity diagram

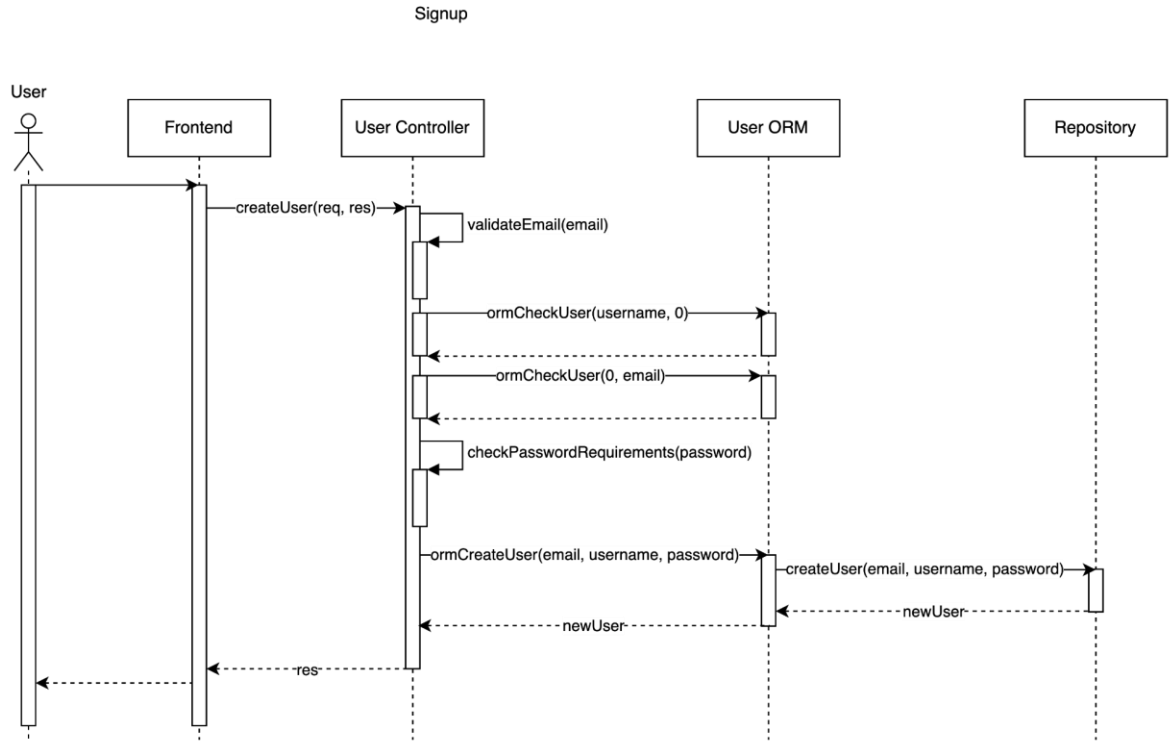


Fig 6.2.2.4: Account creation simplified sequence diagram

When the user receives the email, there will be a link for the user to click for verification and confirmation.Once the user clicks on the link in the email, an API will be called to verify the email token in the link, which will be parsed and validated in `user-controller#verifyEmailToken` method. The database will also be updated, with the `verified` field in the `UserModel` database set to `true`. After confirming and verifying, the user can now login to PeerPrep

Below are the status codes and messages that a user will see when the link is clicked:

Status Code	Message	Explanation
200	Email successfully verified!	
400	Unable to update!	There is a problem when trying to update user in the UserModel database
404	Verification failed! Token cannot be found!	This is most likely due to the user already been verified
404	User is not found!	User cannot be found in the UserModel database
500	Error verifying user!	This is most likely due to connectivity issue with the database or network

For better visualisation, below shows the general activity diagram and sequence diagram when a user clicks on the link in the email:

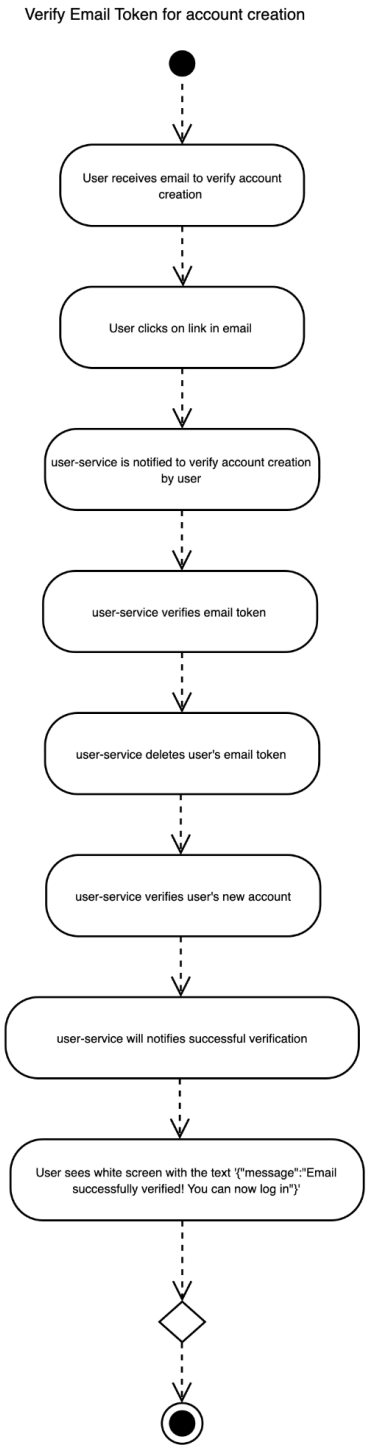


Fig 6.2.2.3: Account creation link verification simplified activity diagram

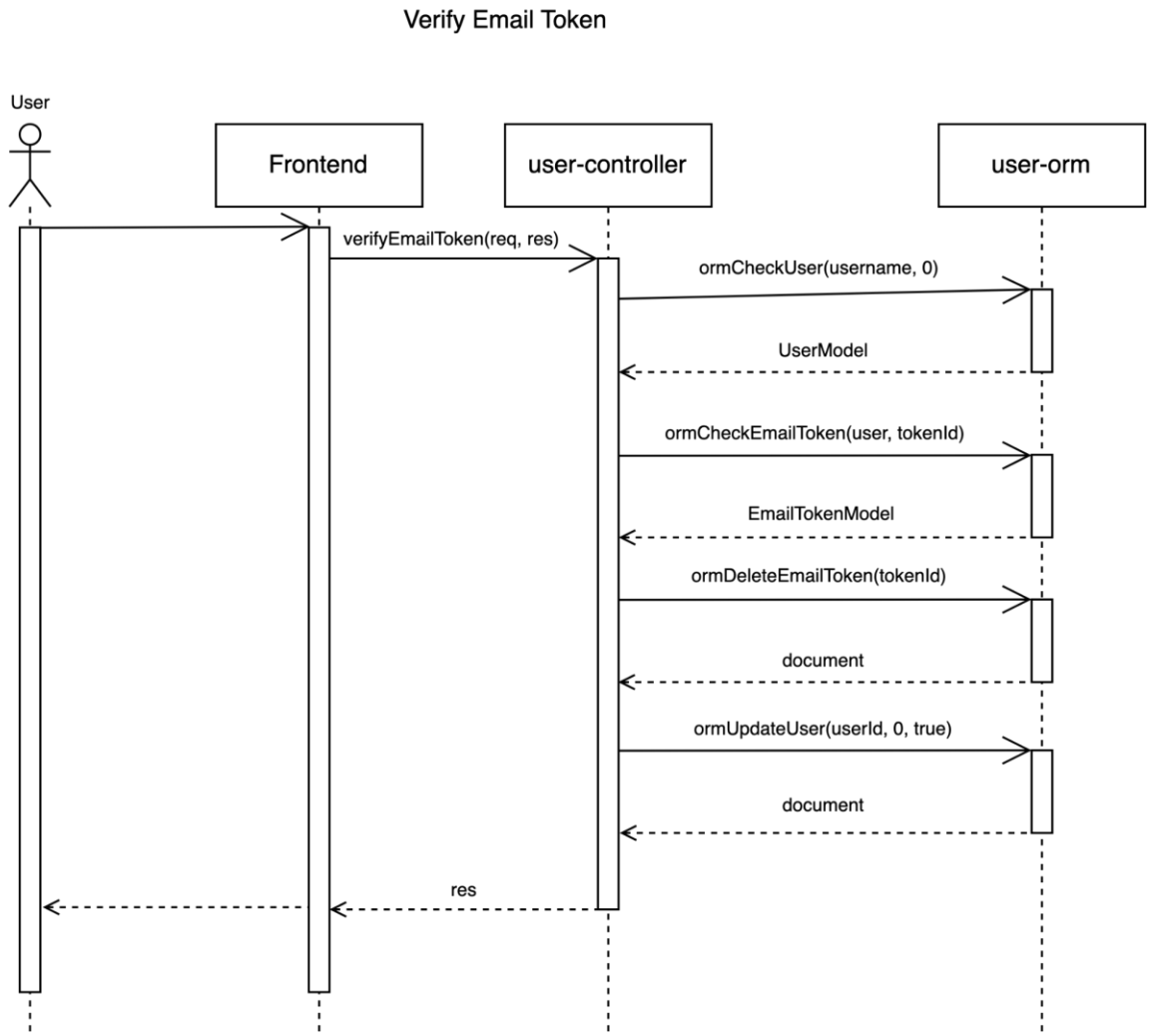


Fig 6.2.2.6: Account creation link verification simplified sequence diagram

Login Feature

This section will explain the implementation of the Login feature. When the user clicks on the 'Log in' button, the API for logging into an account will be called and data (containing the username and password) will be sent to `user-controller#userLogin` method in User Service to be parsed and validated before gaining access into PeerPrep. Once login data is validated successfully, the user will be brought to the main page of PeerPrep. This validation fulfils NFR7.

Below are the status codes and messages that a user will see upon login:

Status Code	Message	Explanation
200		Direct to homepage of PeerPrep
400	Missing fields!	Email, Username and/or Password are missing
401	Incorrect password!	Password input is incorrect

404	Username not found in database!	The user is not registered in the database yet
500	Database failure when logging in!	This is most likely due to connectivity issue with the database or network

For better visualisation, below shows the general activity diagram and sequence diagram when a user login:

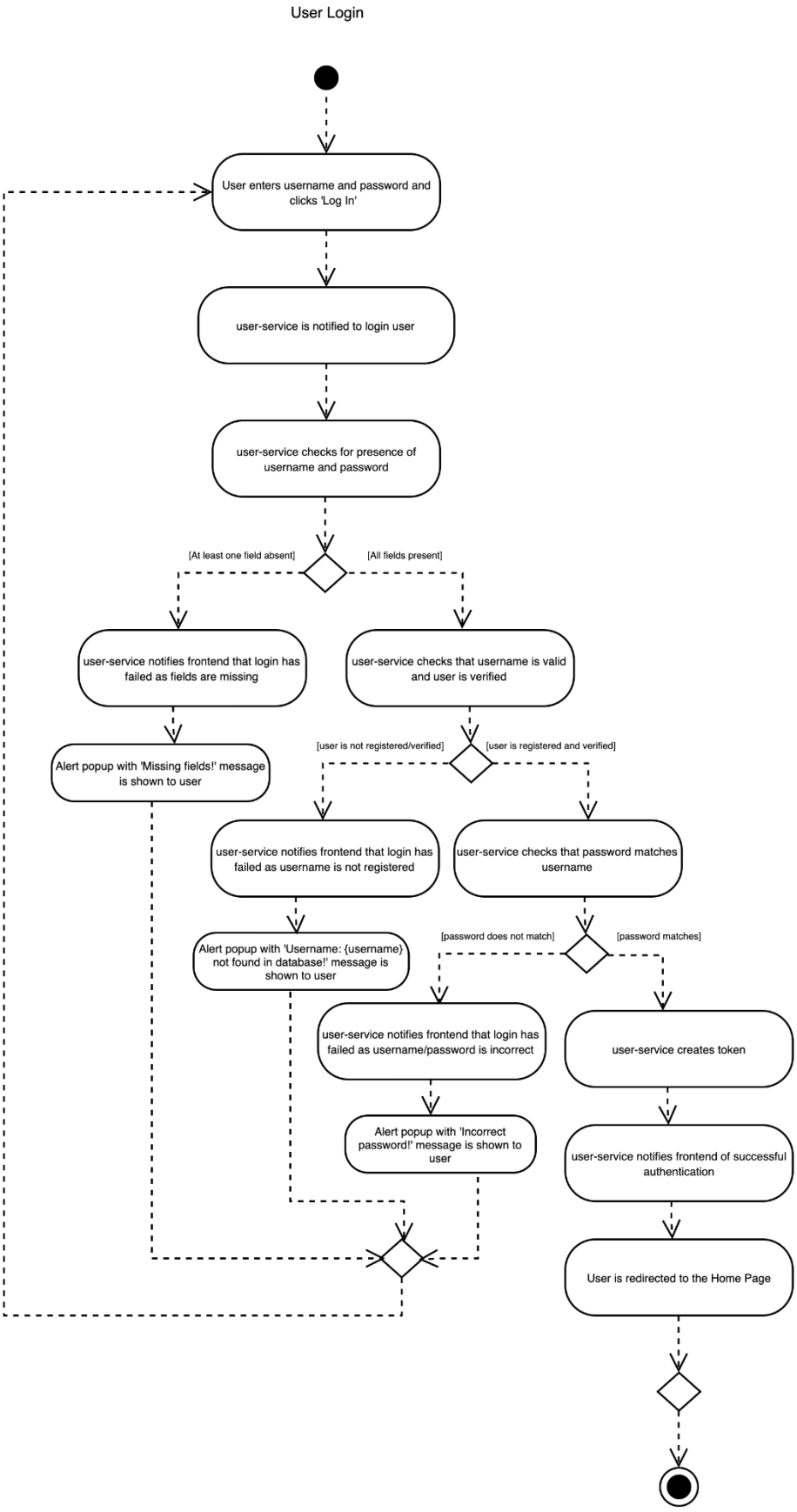


Fig 6.2.2.7: Login activity diagram

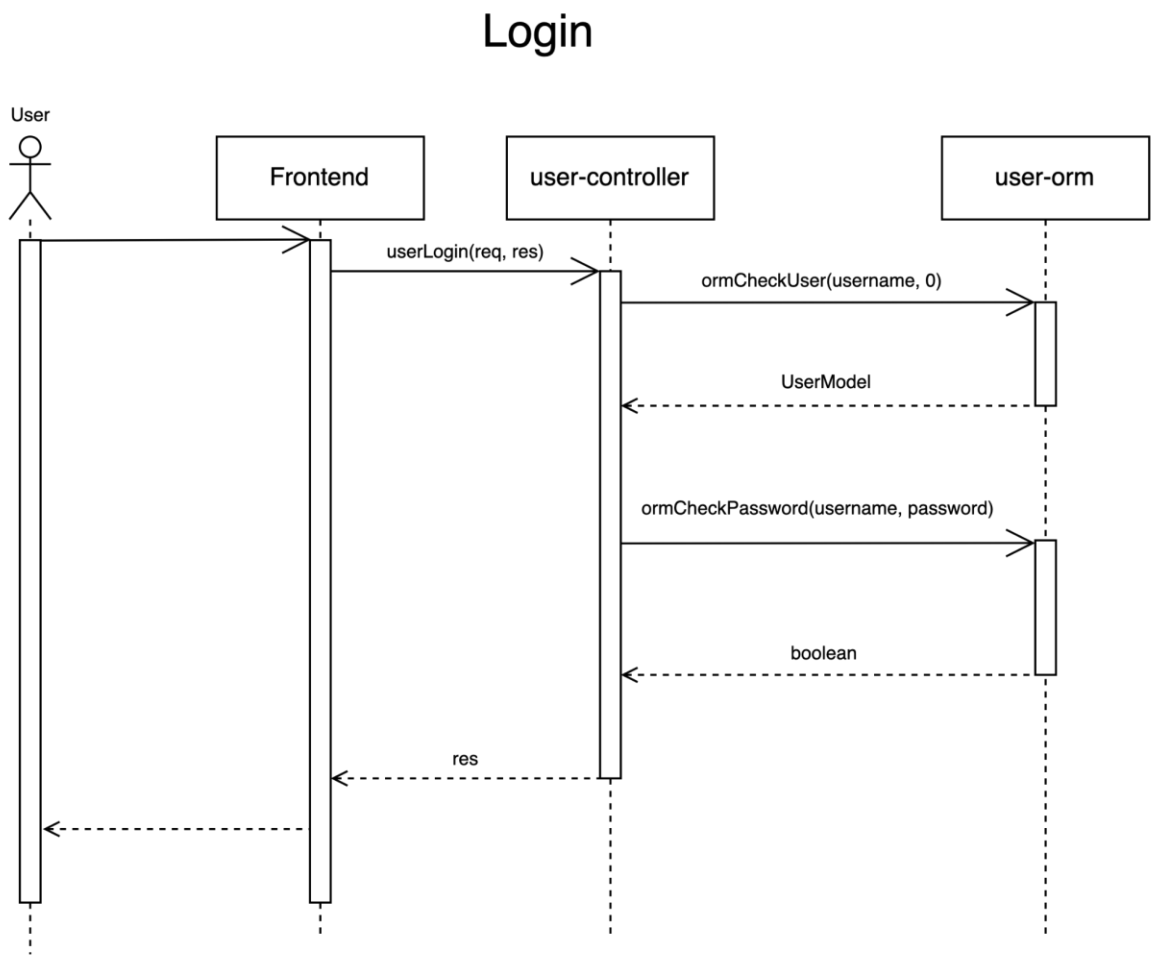


Fig 6.2.2.8: Login simplified class diagram

Update Password Feature

This section will explain the implementation of the Update Password feature. For a user to change password, he/she needs to be logged in, in other words, authorised. Otherwise, the user will be directed to the login page. If the user is authorised, the user will be able to update their password and the data provided will need to be parsed and validated before the update is successful. This will be done through the `user-controller#changePassword` method. Once the password is updated successfully in the database, the user will be brought back to the login page to login again with the new password

Below are the status codes and messages that a user will see when updating password:

Status Code	Message	Explanation
200	Password successfully updated!	Direct to login page
400	Missing fields!	Username, new password and/or old password are missing
400	New password failed to meet requirements	New password input did not meet the requirements

401	Incorrect password!	Old password input is incorrect
401	Unauthorized!	The user is not authorised and will be brought back to the login page
404	Username not found in database!	The user is not registered in the database yet
500	Database failure when changing password!	This is most likely due to connectivity issue with the database or network

For better visualisation, below shows the general activity diagram and sequence diagram when a user updates password:

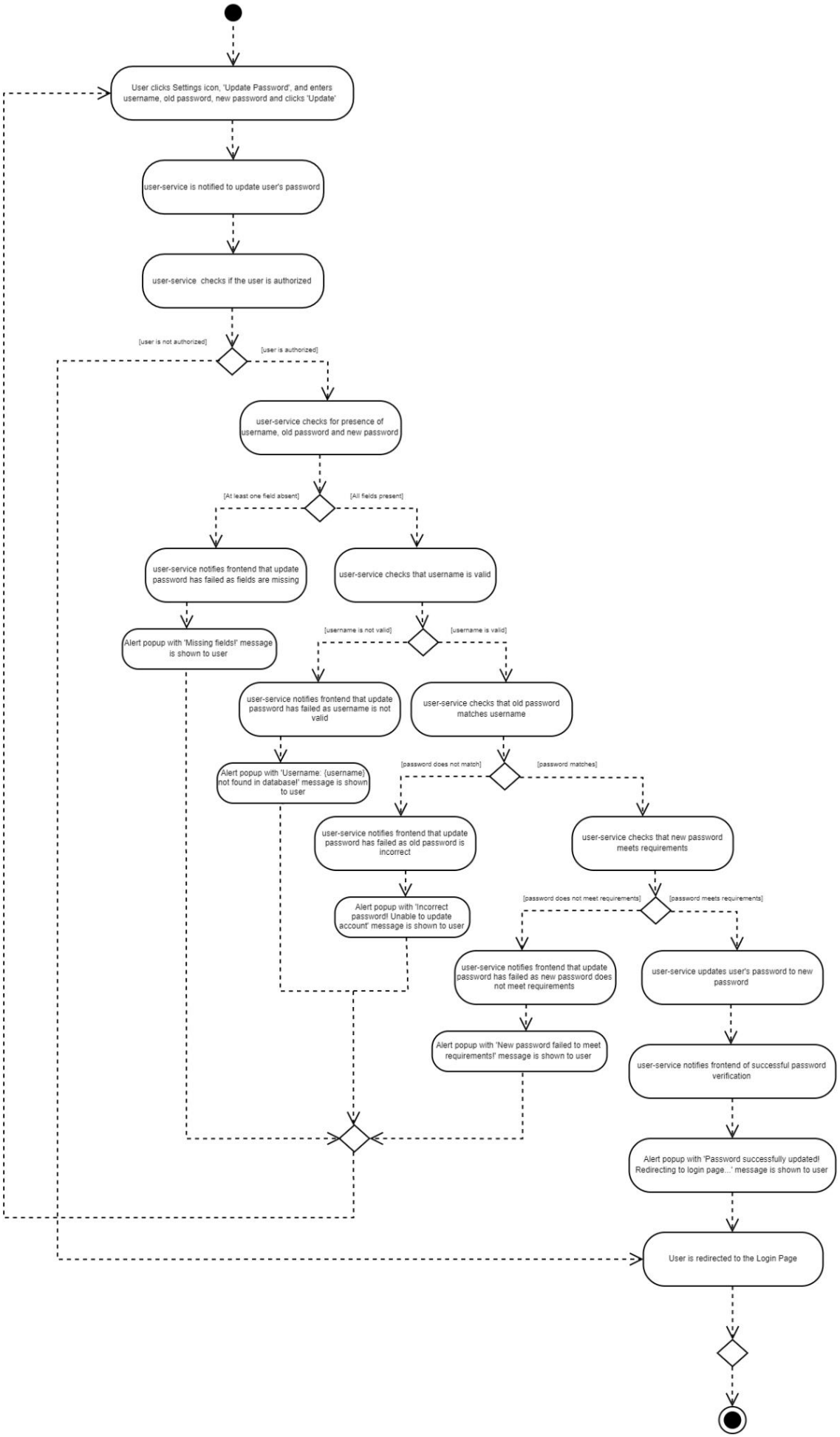

User clicks Settings icon, 'Update Password', and enters username, old password, new password and clicks 'Update'
user-service is notified to update user's password
user-service checks if the user is authorized
[user is not authorized]
[user is authorized]
user-service checks for presence of username, old password and new password
[At least one field absent]
[All fields present]
user-service notifies frontend that update password has failed as fields are missing
user-service checks that username is valid
Alert popup with 'Missing fields!' message is shown to user
[username is not valid]
[username is valid]
user-service notifies frontend that update password has failed as username is not valid
user-service checks that old password matches username
Alert popup with 'Username: {username} not found in database!' message is shown to user
[password does not match]
[password matches]
user-service notifies frontend that update password has failed as old password is incorrect
user-service checks that new password meets requirements
Alert popup with 'Incorrect password! Unable to update account' message is shown to user
[password does not meet requirements]
[password meets requirements]
user-service notifies frontend that update password has failed as new password does not meet requirements
user-service updates user's password to new password
Alert popup with 'New password failed to meet requirements!' message is shown to user
user-service notifies frontend of successful password verification
Alert popup with 'Password successfully updated! Redirecting to login page...' message is shown to user
User is redirected to the Login Page

Fig 6.2.2.9: Password update activity diagram

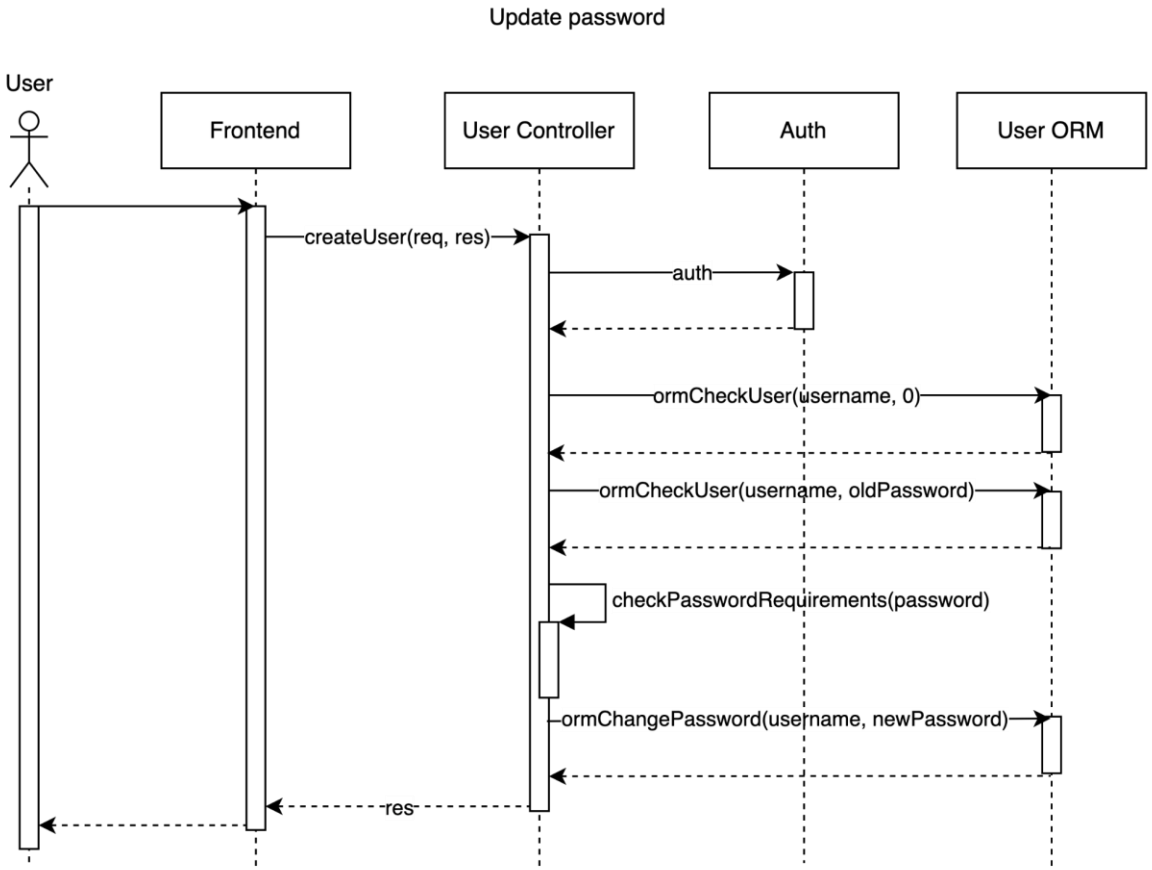


Fig 6.2.2.10: Password update simplified sequence diagram

Delete Account Feature

This section will explain the implementation of the Delete Account feature. For a user to delete their account, he/she needs to be logged in, in other words, authorised. Otherwise, the user will be directed to the login page. If the user is authorised, the user will be able to delete their account and the data provided will need to be parsed and validated before the deletion is successful. This will be done through the `user-controller#deleteUser` method. Once the account is deleted successfully in the database, the user will be brought back to the login page.

Below are the status codes and messages that a user will see when updating password:

Status Code	Message	Explanation
200	Successfully deleted	Direct to login page
400	Missing fields!	Username and/or password are missing
401	Incorrect password!	Password input is incorrect
401	Unauthorized!	The user is not authorised and will be brought back to the login page

404	Username not found in database!	The user is not registered in the database yet
500	Database failure when deleting account!	This is most likely due to connectivity issue with the database or network

For better visualisation, below shows the general activity diagram and sequence diagram when a user delete their account:

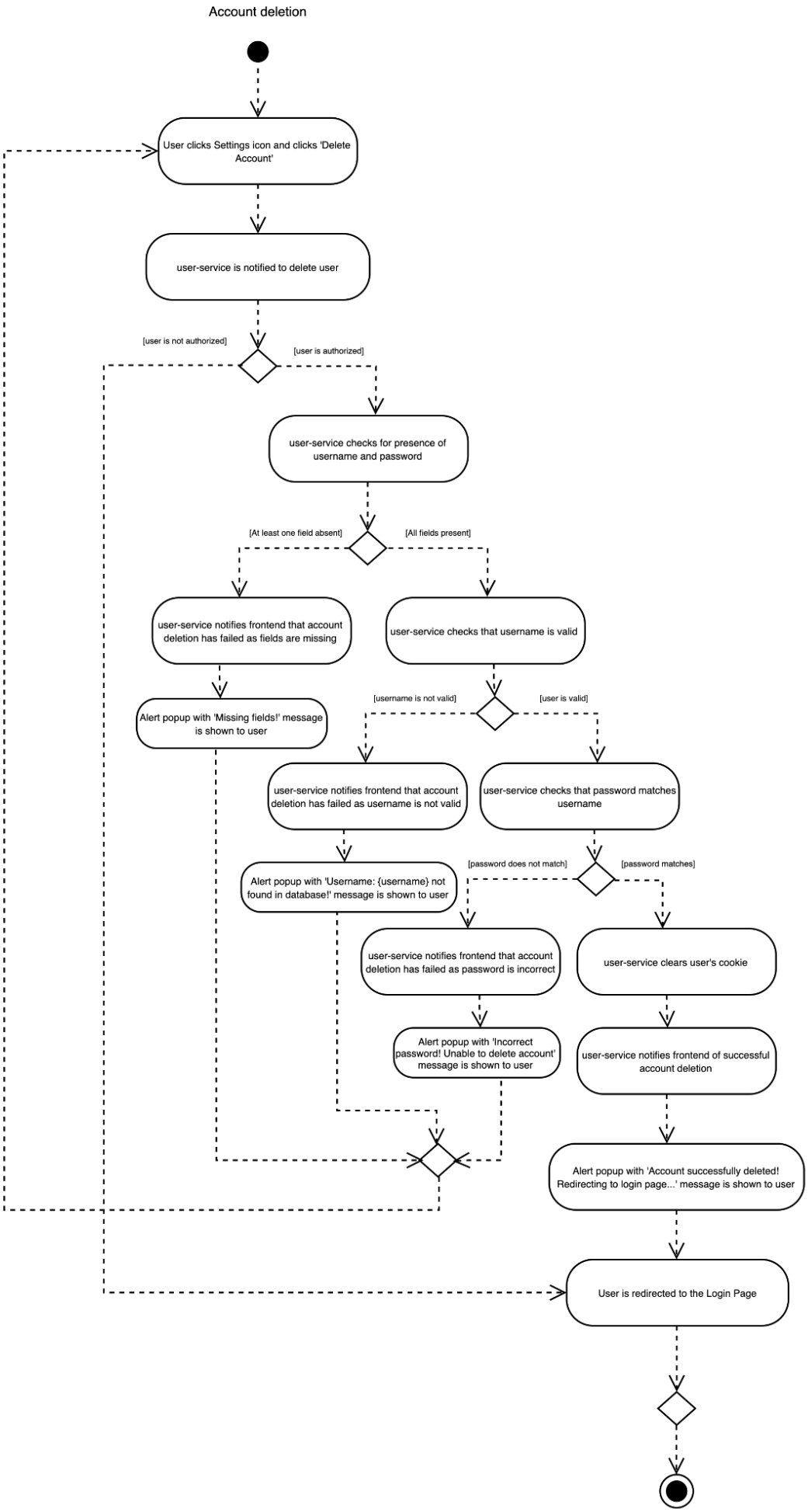

Account deletion
User clicks Settings icon and clicks 'Delete Account'
user-service is notified to delete user
[user is not authorized]
[user is authorized]
user-service checks for presence of username and password
[At least one field absent]
[All fields present]
user-service notifies frontend that account deletion has failed as fields are missing
user-service checks that username is valid
Alert popup with 'Missing fields!' message is shown to user
[username is not valid]
[user is valid]
user-service notifies frontend that account deletion has failed as username is not valid
user-service checks that password matches username
Alert popup with 'Username: {username} not found in database!' message is shown to user
[password does not match]
[password matches]
user-service notifies frontend that account deletion has failed as password is incorrect
user-service clears user's cookie
Alert popup with 'Incorrect password! Unable to delete account' message is shown to user
user-service notifies frontend of successful account deletion
Alert popup with 'Account successfully deleted! Redirecting to login page...' message is shown to user
User is redirected to the Login Page

Fig 6.2.2.11: Account deletion activity diagram

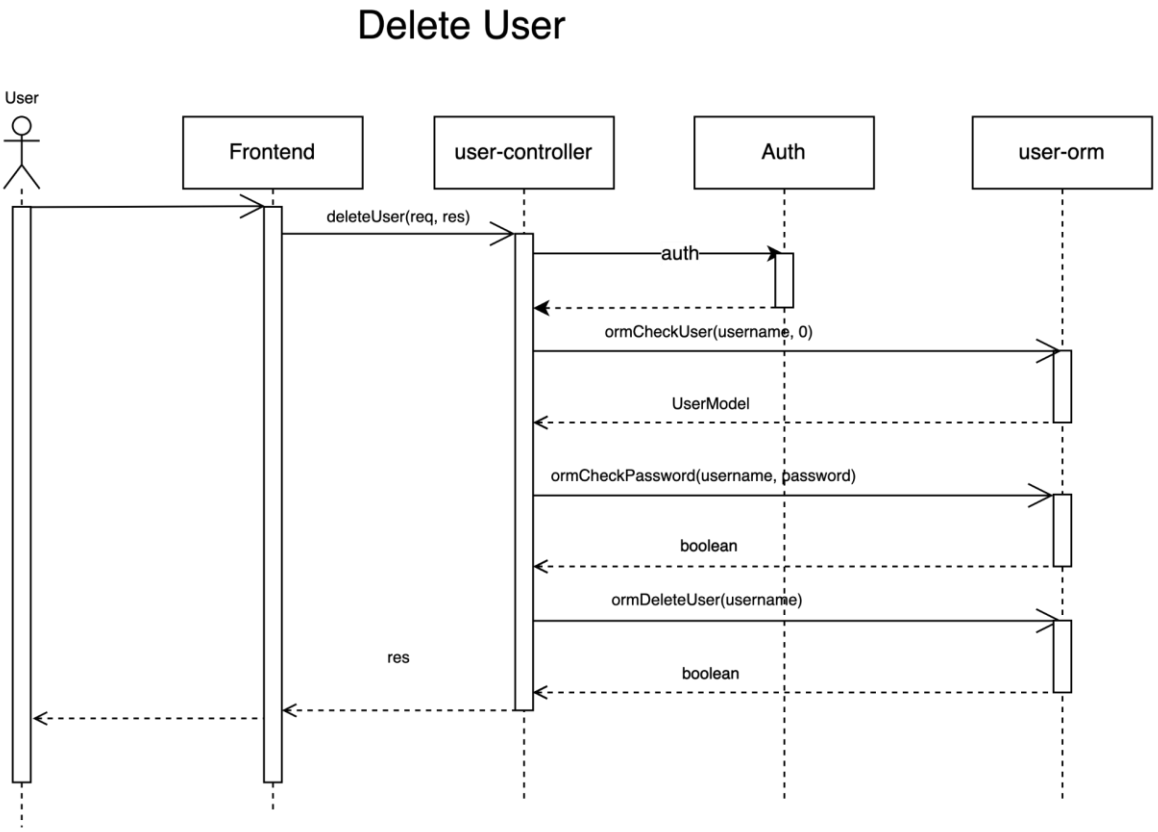


Fig 6.2.2.12: Account deletion simplified sequence diagram

Logout Feature

This section will explain the implementation of the Logout feature. When the user wants to logout, they have to be logged in, in other words, authorised. If they are not authorised, they will be brought back to the login page. If the user is authorised and logs out, the user will be directed to the login page. This is done through the `user-controller#userLogout` method. Correspondingly, the cookies and token saved are also deleted and blacklisted.

Below are the status codes and messages when a user logouts:

Status Code	Message	Explanation
200	Successfully log out!	Direct to login page
401	Unauthorized!	The user is not authorised and will be brought back to the login page

For better visualisation, below shows the general activity diagram and sequence diagram when a user logouts:

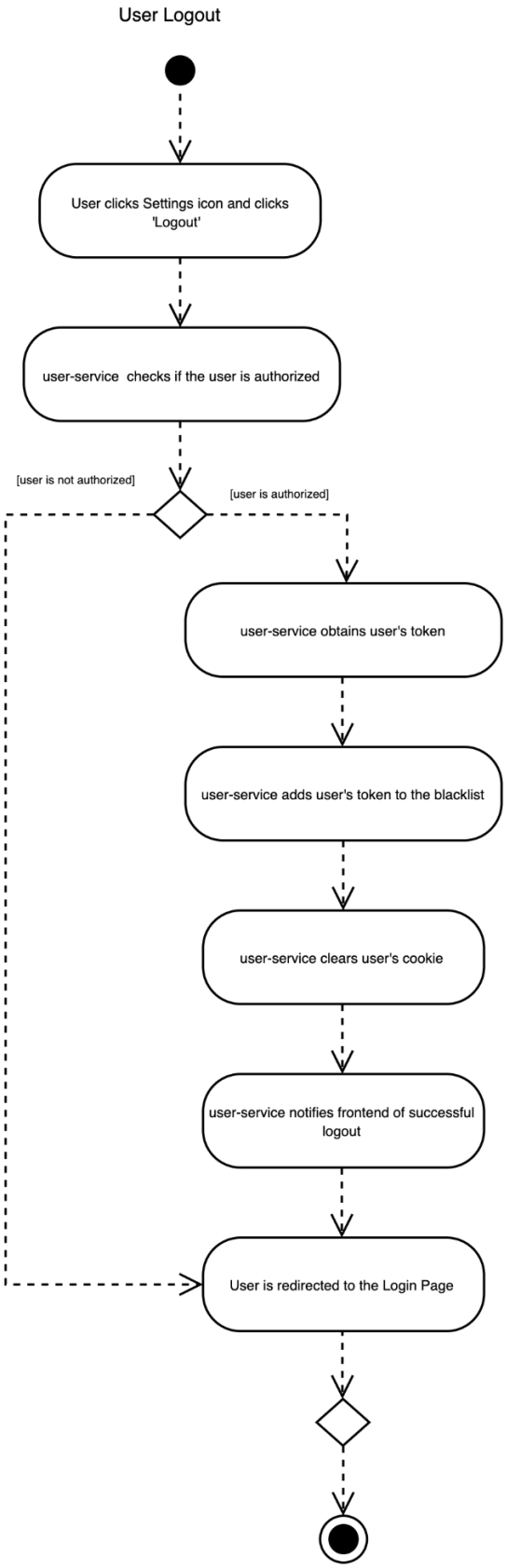


Fig 6.2.2.13: Logout activity diagram

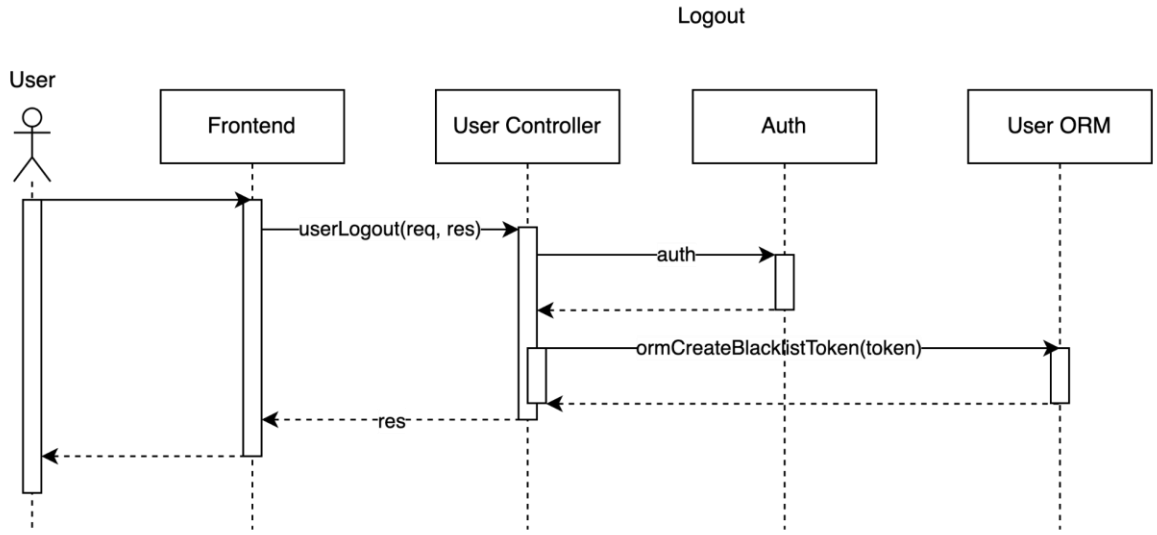


Fig 6.2.2.14: Logout simplified sequence diagram

Reset Password Feature

This section will explain the implementation of the Reset Password feature. This feature is used when a user forgets their password. In that case, the user is able to reset their password through email, with email as the data to be parsed and validated through the `user-controller#resetPasswordUsingEmail` method. In return, an email will be sent using the `email#sentEmail` method to the user with a link to reset their password.

Below are the status codes and messages that a user will see when resetting password using email:

Status Code	Message	Explanation
200	Email to reset password is sent successfully	
400	Missing fields!	Email is missing
400	Invalid email format!	Email format is invalid and therefore failed validation
400	Unable to sent email!	This is most likely a problem with creating an email token
404	Email does not exist in the database!	The user is not registered in the database yet
500	Error	This is most likely due to connectivity issue with the database or network

For better visualisation, below shows the general activity diagram and sequence diagram when a user resets password using email:

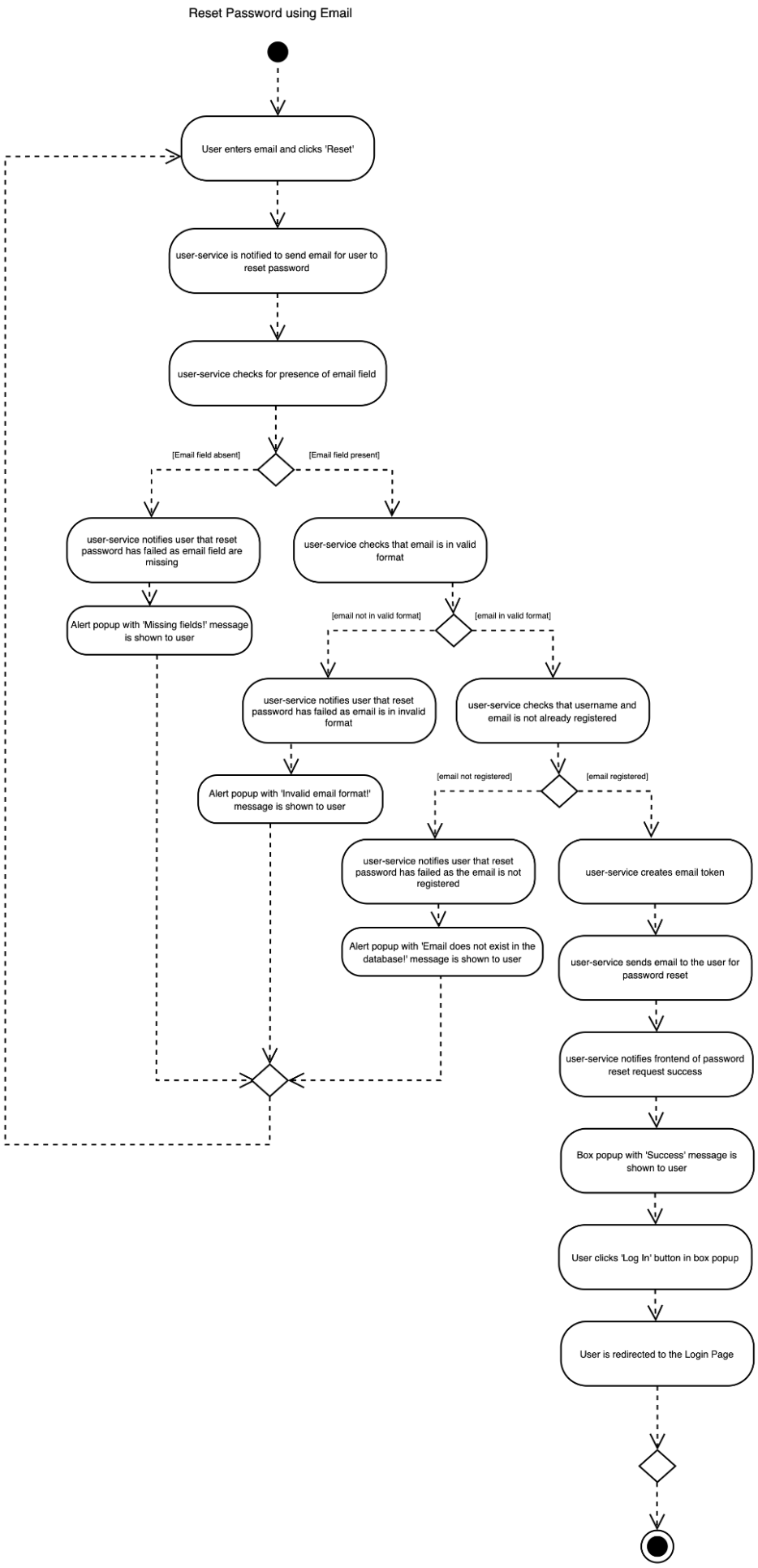

Reset Password using Email
User enters email and clicks 'Reset'
user-service is notified to send email for user to reset password
user-service checks for presence of email field
[Email field absent]
[Email field present]
user-service notifies user that reset password has failed as email field are missing
Alert popup with 'Missing fields!' message is shown to user
user-service checks that email is in valid format
[email not in valid format]
[email in valid format]
user-service notifies user that reset password has failed as email is in invalid format
Alert popup with 'Invalid email format!' message is shown to user
user-service checks that username and email is not already registered
[email not registered]
[email registered]
user-service notifies user that reset password has failed as the email is not registered
Alert popup with 'Email does not exist in the database!' message is shown to user
user-service creates email token
user-service sends email to the user for password reset
user-service notifies frontend of password reset request success
Box popup with 'Success' message is shown to user
User clicks 'Log In' button in box popup
User is redirected to the Login Page

Fig 6.2.2.15: Password reset using email activity diagram

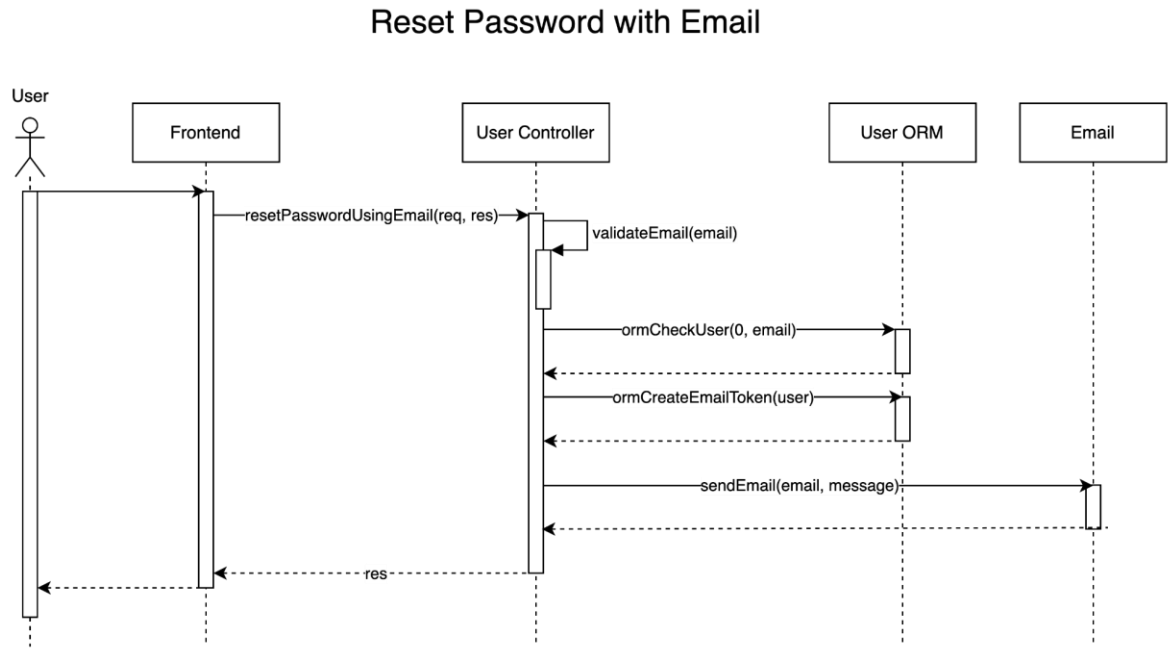


Fig 6.2.2.16: Password reset using email simplified class diagram

Once the user clicks on the link in the email, he/she will be directed to a page to reset their password. When the user clicks on the 'Reset' button, the API for resetting password for an account will be called and data (containing the user's new password) as well as email token in the link will be sent to `user-controller#resetPassword` method in User Service to be parsed and validated before updating the user's old password in the database with the new password. The email token will also be deleted from the database. Once the update is successful, the user can once again log into PeerPrep with the new password.

Below are the status codes and messages that a user will see when resetting password:

Status Code	Message	Explanation
200	Email to reset password is sent successfully	
400	Missing fields!	Password is missing
400	Unable to reset password!	This is most likely due to update failure in the database
400	Password does not meet requirements	New password input did not meet the requirements
401	Token cannot be found!	This is most likely due to user having reset their password before with the same link
404	User cannot be found in the database!	The user is not registered in the database yet

500	Database failure when resetting password!	This is most likely due to connectivity issue with the database or network

For better visualisation, below shows the general activity diagram and sequence diagram when a user resets password:

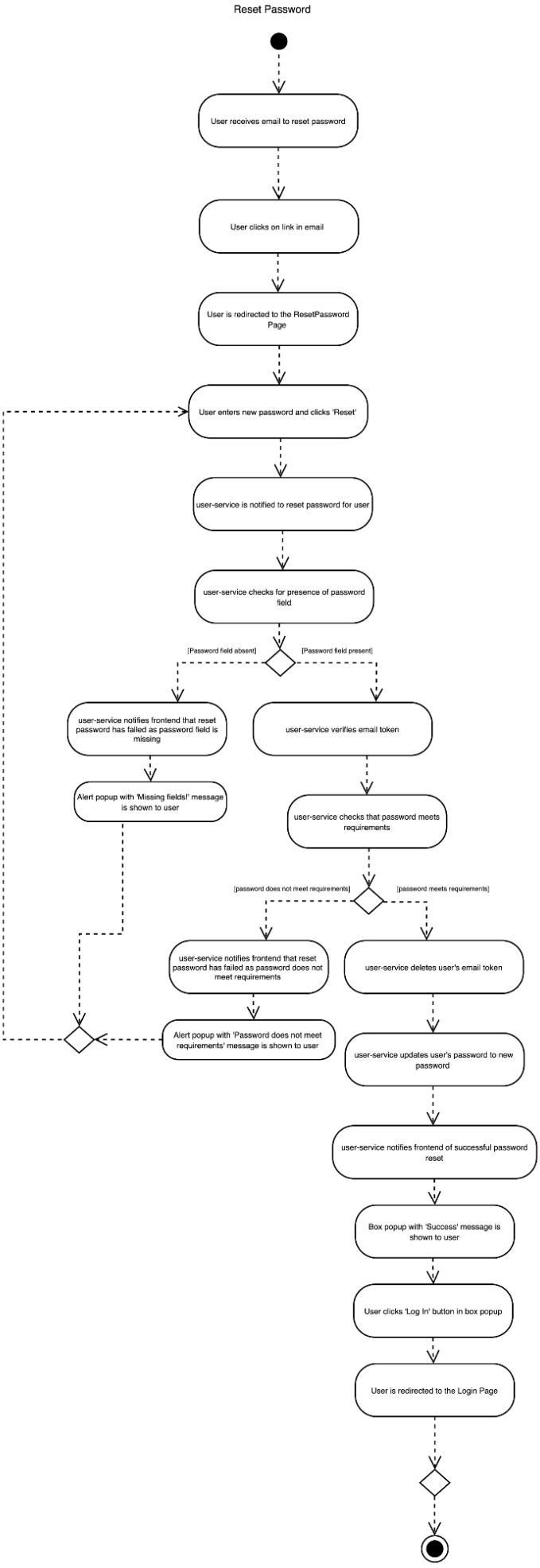

Reset Password
User receives email to reset password
User clicks on link in email
User is redirected to the ResetPassword Page
User enters new password and clicks 'Reset'
user-service is notified to reset password for user
user-service checks for presence of password field
[Password field absent]
[Password field present]
user-service notifies frontend that reset password has failed as password field is missing
Alert popup with 'Missing fields!' message is shown to user
user-service verifies email token
user-service checks that password meets requirements
[password does not meet requirements]
[password meets requirements]
user-service notifies frontend that reset password has failed as password does not meet requirements
Alert popup with 'Password does not meet requirements' message is shown to user
user-service deletes user's email token
user-service updates user's password to new password
user-service notifies frontend of successful password reset
Box popup with 'Success' message is shown to user
User clicks 'Log In' button in box popup
User is redirected to the Login Page

Fig 6.2.2.17: Password reset activity diagram

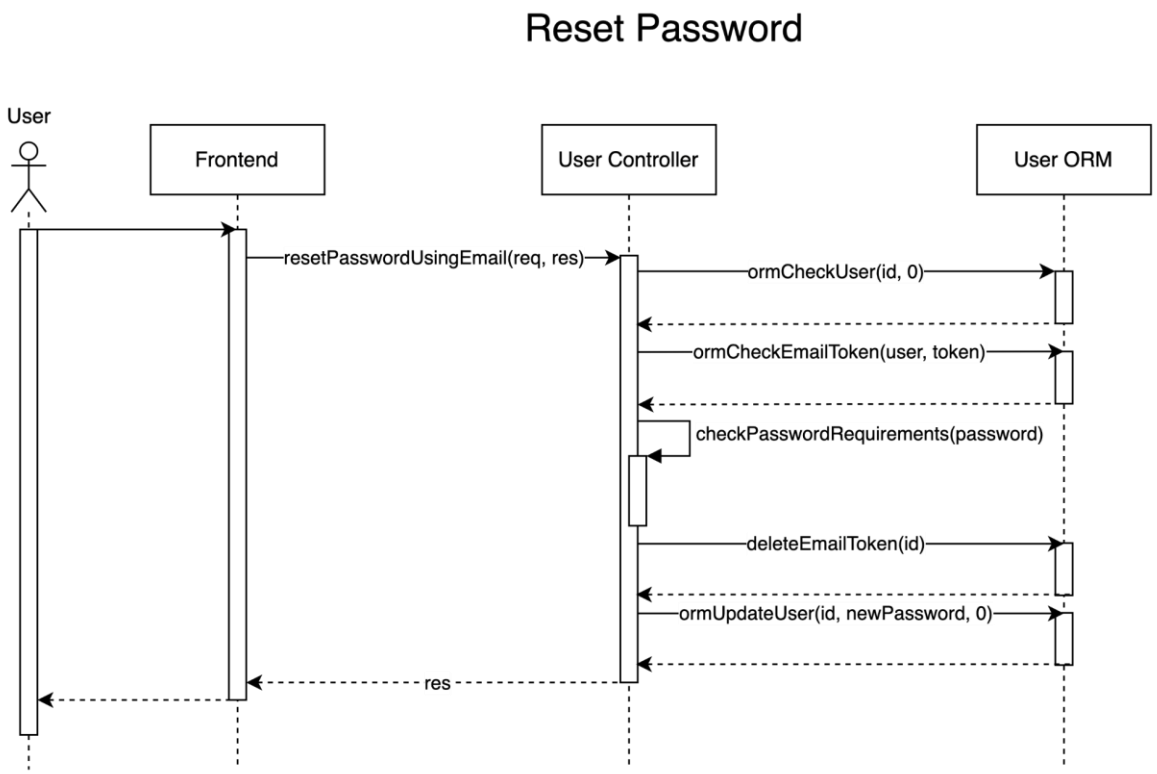


Fig 6.2.2.18: Password Reset simplified class diagram

6.2.3 Matching Service

The matching service is responsible for finding 2 users who have chosen to attempt a question of the same difficulty level and pairing them.

The matching service communicates with the frontend of the application via Socket.IO. Socket.IO allows for persistent connection between the matching service and frontend so they can notify each other of changes until the connection is closed.

Architecture

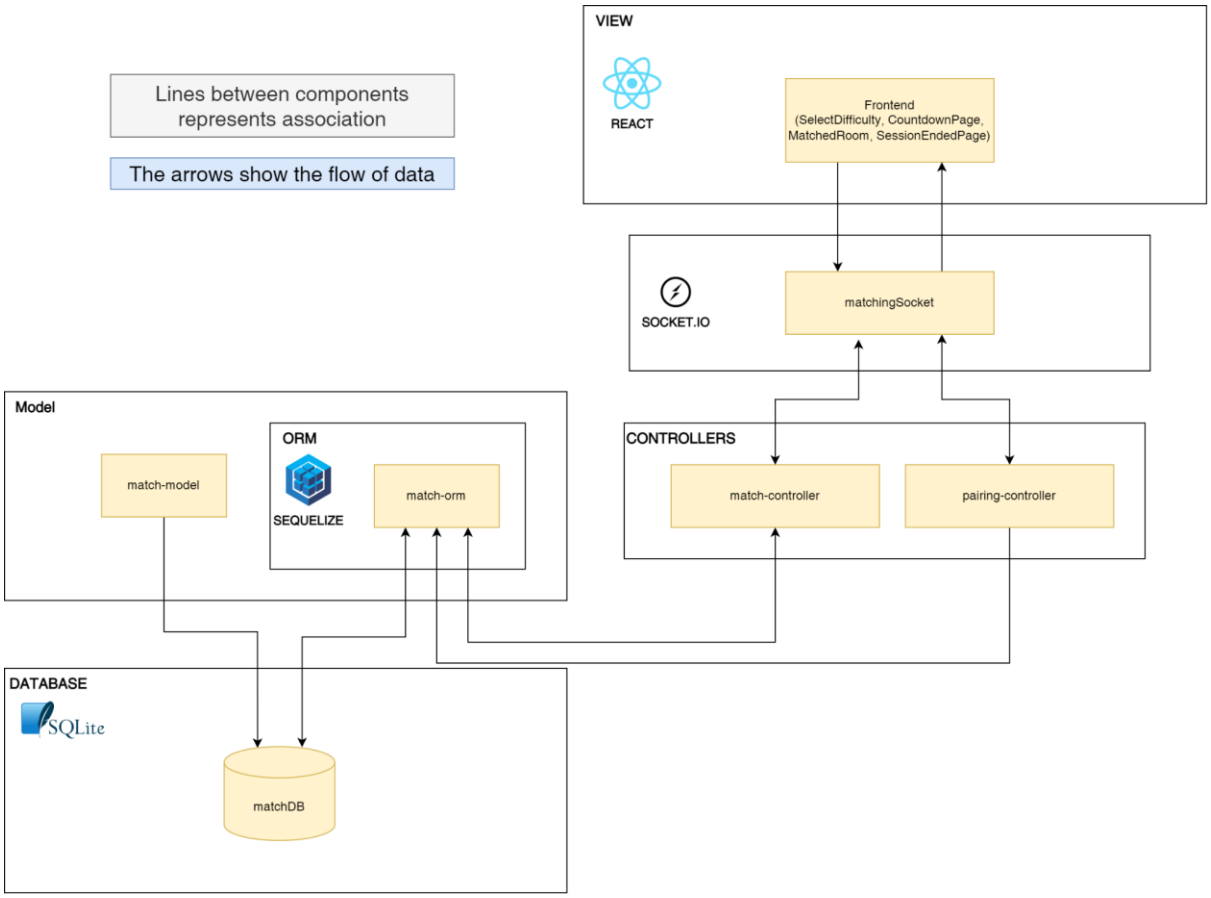


Fig 6.2.3.1: Architecture of matching service

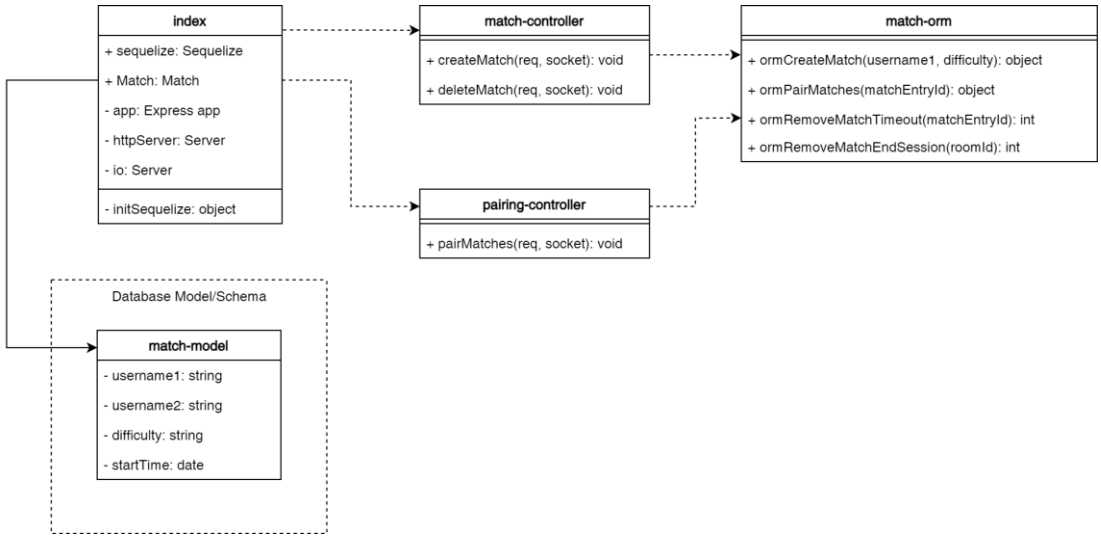


Fig 6.2.3.2: Class diagram of matching service

Match Database Schema

Field	Data type	Field explanation
username1	string	Username of the first user in the pair

username2	string	Username of the second user in the pair
difficulty	string	Difficulty level of the question the users chose to attempt
startTime	date	Time when match entry is created

Features aka APIs

Matching service listens for 3 events emitted by the frontend. In total, there are 3 features for matching service.

Match entry creation

Match entry creation occurs after a user selects the difficulty level of the question they want to attempt in the SelectDifficulty component found on the Home page and clicks the 'Next' button. When this occurs, the frontend will emit a "match" event via the socket connected to the matching service to notify the matching service that the user is requesting for a match. The matching service, which listens for the "match" event in the index.js file, will then attempt to add a new entry in the database for the user.

When notified of the "match" event, index.js calls the `match-controller#createMatch(req, socket)` method. The body of the req passed to the method specifies the username and difficulty level selected by the user. The match-controller then passes the username and difficulty level to the match-orm via the `match-orm#ormCreateMatch(username1, difficulty)` method. This method inserts a new match into the match database with the current user's username as username1, an empty string as username2, the selected difficulty level as the difficulty, and the current date and time as startTime.

Below is the match entry created when a user with the username 'userA' selects the difficulty level 'Hard' and clicks the 'Next' button on the Home page:

id	username1	username2	difficulty	startTime
1	userA		Hard	<time when entry is created>

If the match entry is created succesfully, the match-controller will emit a "matchCreationSuccess" event. Else, a "matchCreationFailure" event will be emitted. The frontend listens for both events. When it receives a "matchCreationSuccess" event, the user is redirected to the Countdown page. However, if it receives a "matchCreationFailure" event, the user will be shown an error dialog and asked to try matching again.

The following sequence diagram (Fig 6.2.3.3) shows the match creation for a user:

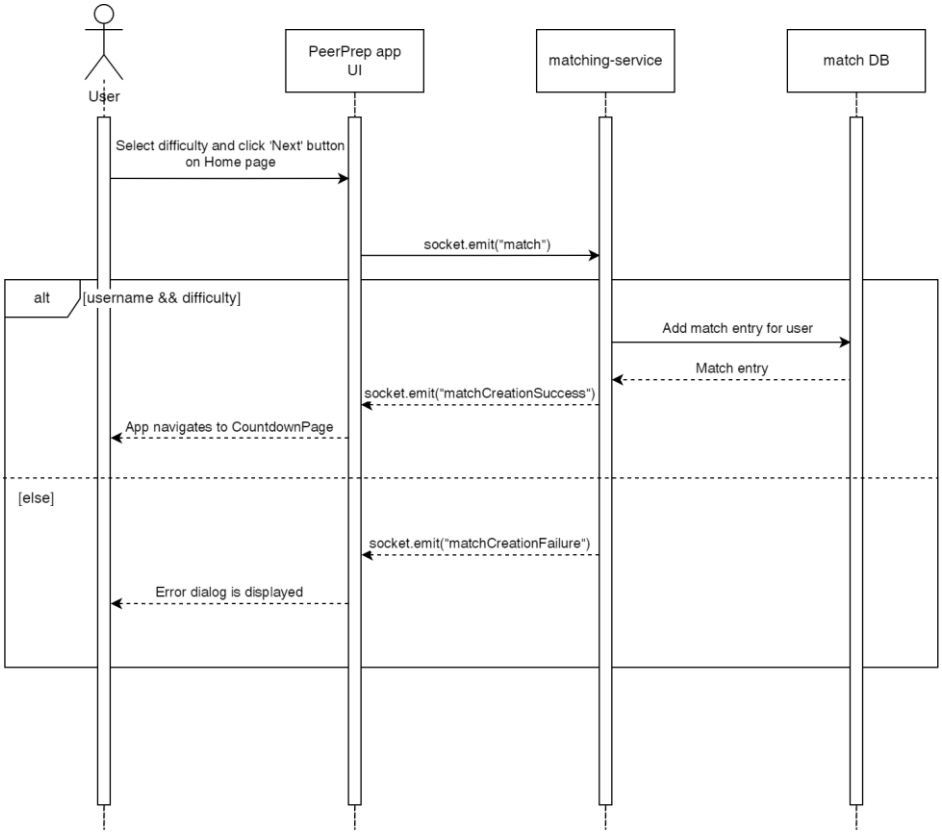


Fig 6.2.3.3: Match creation sequence diagram

The following activity diagram (Fig 6.2.3.4) summarises what happens during match creation:

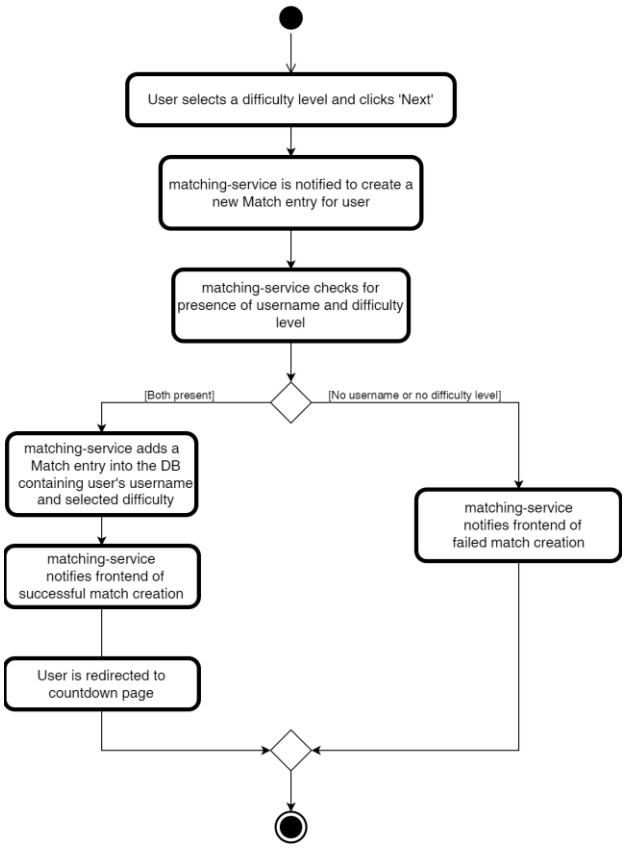


Fig 6.2.3.4: Match creation activity diagram

Pairing of matches

Pairing of matches occurs after the user is brought to the Countdown page. The frontend emits a “pairing” event via the socket connected to the matching service to notify the matching service to pair the current user with a partner. The matching service, which listens for the “pairing” event in the index.js file, will then attempt to find a partner for the user.

When notified of the “pairing” event, index.js calls the `pairing-controller#pairMatches(req, socket)` method. The body of the req passed to the method specifies the match entry ID of the current user and the time left for a partner to be found. The pairing-controller checks if the time left for a partner to be found has reached 0. If it has, the pairing-controller emits a “pairingFailed” event. Otherwise, the pairing-controller passes the match entry ID to the match-orm via the `match-orm#ormPairMatches(matchEntryId)` method. This method first retrieves the difficulty level the current user selected using the match entry ID. It then checks the database for another match entry where the difficulty matches the current user’s selected difficulty level. If match entry meeting the criteria is found, the current user’s match entry is combined with the other match entry.

Below is the updated match entry when users with the usernames ‘userA’ and ‘userB’ who selected the difficulty level ‘Hard’ are paired:

id	username1	username2	difficulty	startTime
1	userA	userB	Hard	<time when entry is created>

If 2 users are paired succesfully, the pairing-controller will emit a “pairingSuccess” event. When the frontend receives a “pairingSuccess” event, the user is redirected to the Matched Room page.

The following sequence diagram (Fig 6.2.3.5) shows the match creation for a user:

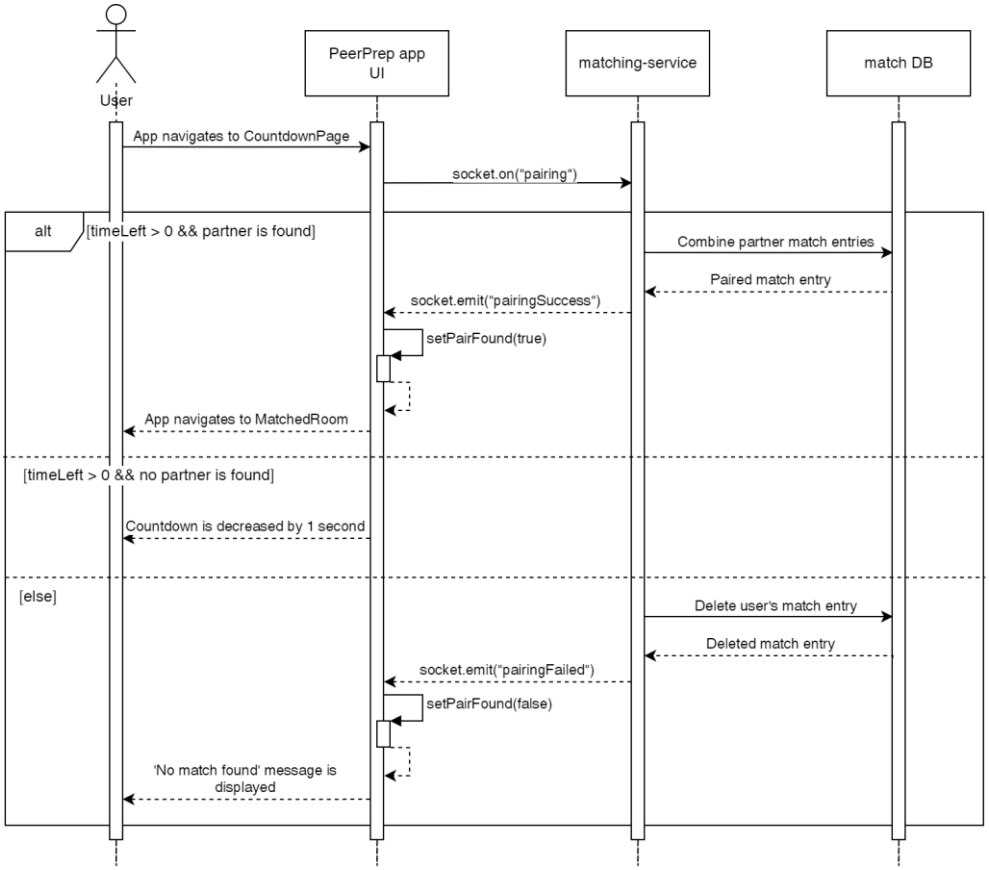


Fig 6.2.3.5: Match pairing sequence diagram

The following activity diagram (Fig 6.2.3.6) summarises what happens during match creation:

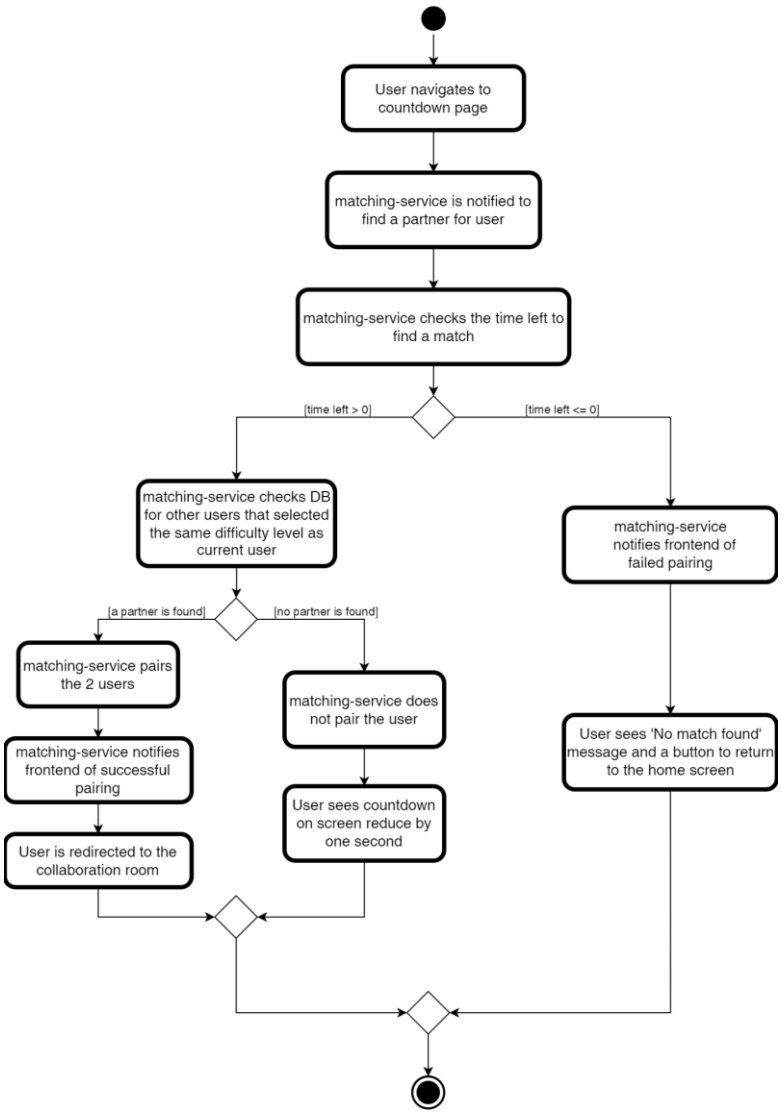


Fig 6.2.3.6: Match pairing activity diagram

Deleting of matches

Deleting of matches occurs when a user leaves the session or the session ends. The frontend emits a “endSession” event via the socket connected to the matching service to notify the matching service to delete the pair’s match entry. The matching service, which listens for the “endSession” event in the index.js file, will then attempt to delete the pair’s match entry.

When notified of the “endSession” event, index.js calls the `match-controller#deleteMatch(req, socket)` method. The body of the req passed to the method specifies the match entry ID of the pair in the session. The pairing-controller then passes the match entry ID to the match-orm via the `match-orm#ormRemoveMathEndSession(roomId)` method. This method removes the match entry with the id that matches the roomId from the database.

The following sequence diagram (Fig 6.2.3.7) shows the match creation for a user:

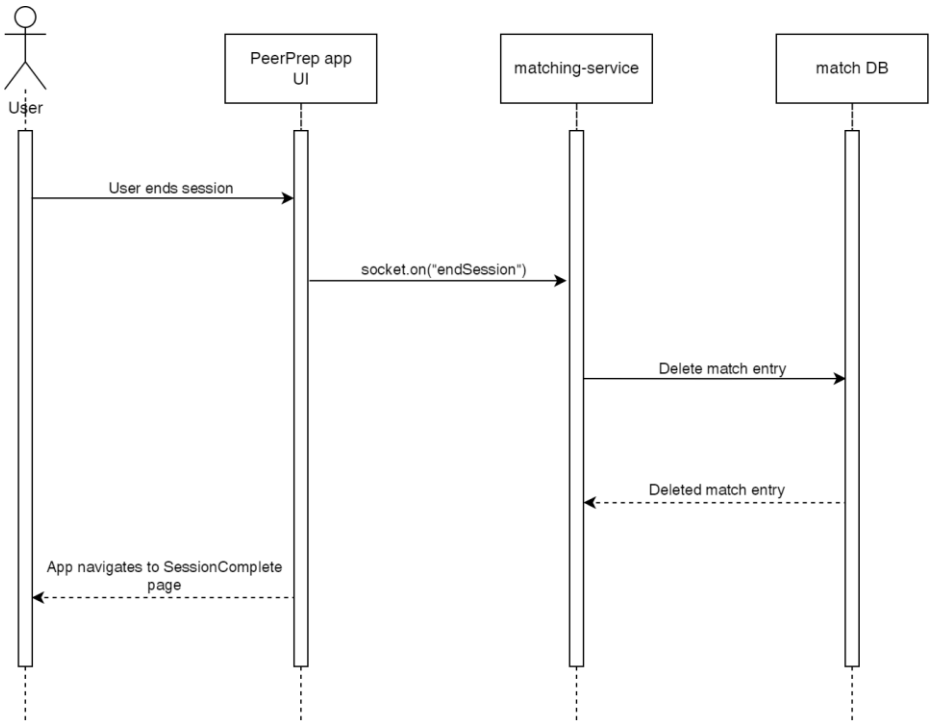


Fig 6.2.3.7: Match deletion sequence diagram

The following activity diagram (Fig 6.2.3.8) summarises what happens during match creation:

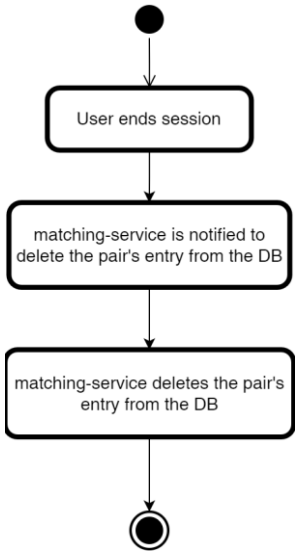


Fig 6.2.3.8: Match deletion activity diagram

6.2.4 Collaboration Service

The collaboration service is responsible for placing the 2 matched users into the same Room so that they can share a collaboration space. In the room, the service ensures that they both view the same question, have the same amount of time to work on the problem, and can collaborate in real time.

The collaboration service, like the matching service, communicates with the frontend of the application via Socket.IO.

Architecture

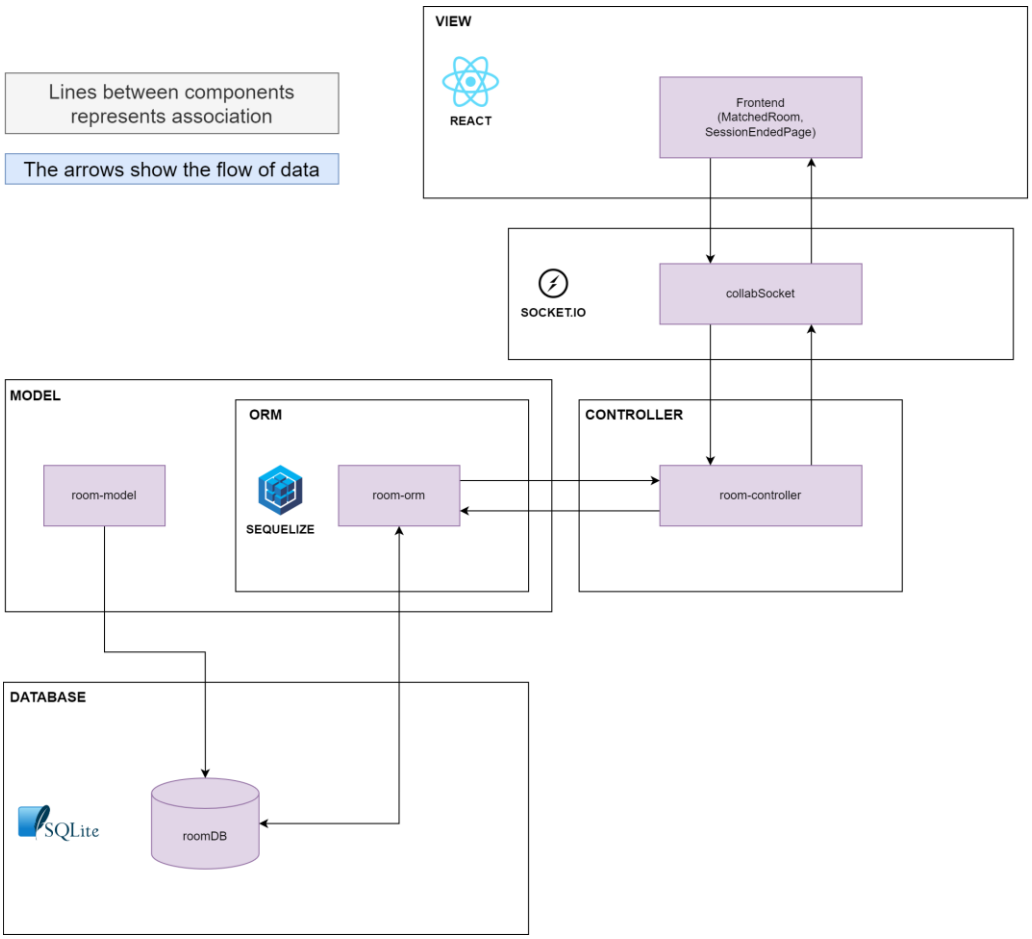


Fig 6.2.4.1: Architecture of collaboration service

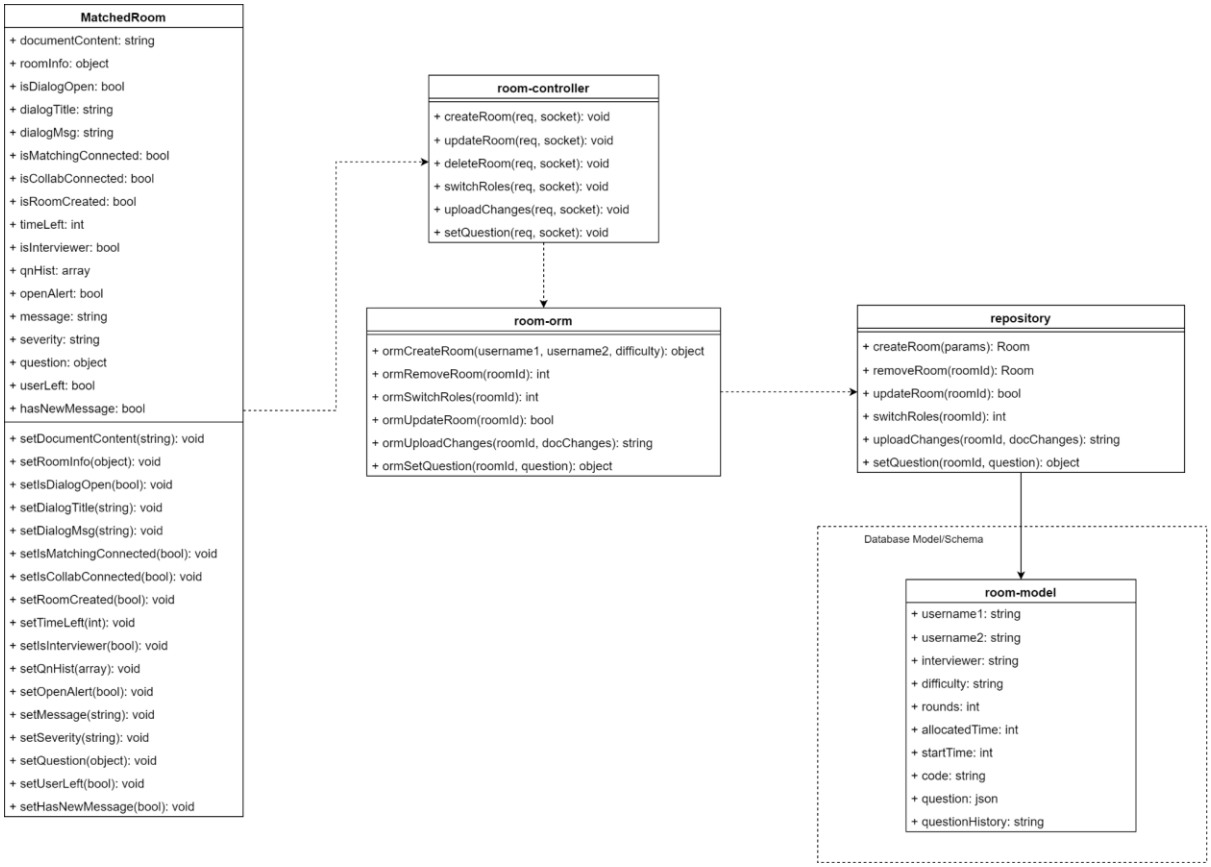


Fig 6.2.4.2: Class diagram of collaboration service

Room Database Schema

Field	Data type	Field explanation
username1	string	User1 in the room
username2	string	User2 in the room
interviewer	string	User that is the interviewer
difficulty	string	Difficulty level selected by both users
rounds	string	Number of rounds that have passed. Used to keep track of if both users have already switched roles. Its default value is 0.
allocatedTime	integer	Time given to solve the question based on the difficulty level. Its default value is 0.
startTime	date	Time that room is created. Its default value is the current date.

code	string	Keeps track of code typed into the shared space
question	json	Question displayed to both users. Its default value is null.
questionHistory	string	Questions that have already been shown before. Its default value is an empty string.

Features aka APIs

Room entry creation

A room is created when 2 users with the same difficulty level selection are matched within 30 seconds and redirected to the MatchedRoom. On initial render of the MatchedRoom, the frontend will emit the event “createRoom” via the socket connected to the collaboration service. The collaboration service listening to the “createRoom” event in the index.js file is informed to create a new room entry for the users and will attempt to create a new entry in the roomDB database.

When notified of the “createRoom” event, index.js calls the `room-controller#createRoom(req, socket)` method. The body of the req passed to the method specifies both users’ usernames and difficulty level. The match-controller then calls the `room-orm#ormCreateRoom(username1, username2, difficulty)` method. This method determines how much time should be allocated based on the difficulty level, and assigns the role of interviewer to a user. It then creates a new Room entry in the roomDB with the current user’s username as username1, the other user’s username as username2, the selected difficulty level, allocated time and the interviewer’s username. More details regarding the schema can be seen in the Room Database Schema section above.

If the Room entry is created successfully, the room-controller will emit an event “roomCreationSuccess”. If entry creation is unsuccessful, the room-controller will instead emit the event “roomCreationFailure”. The frontend listening for the “roomCreationSuccess” event upon being notified, will display the collaboration screen and triggers the question retrieval from the question service. Subsequently, the frontend displays the shared question in the MatchedRoom. However if the frontend receives the “roomCreationFailure” event, the user is redirected back to the Home page.

The sequence diagram (Fig 6.2.4.3) below illustrates what has been explained above and shows the interactions between components to create a Room entry:

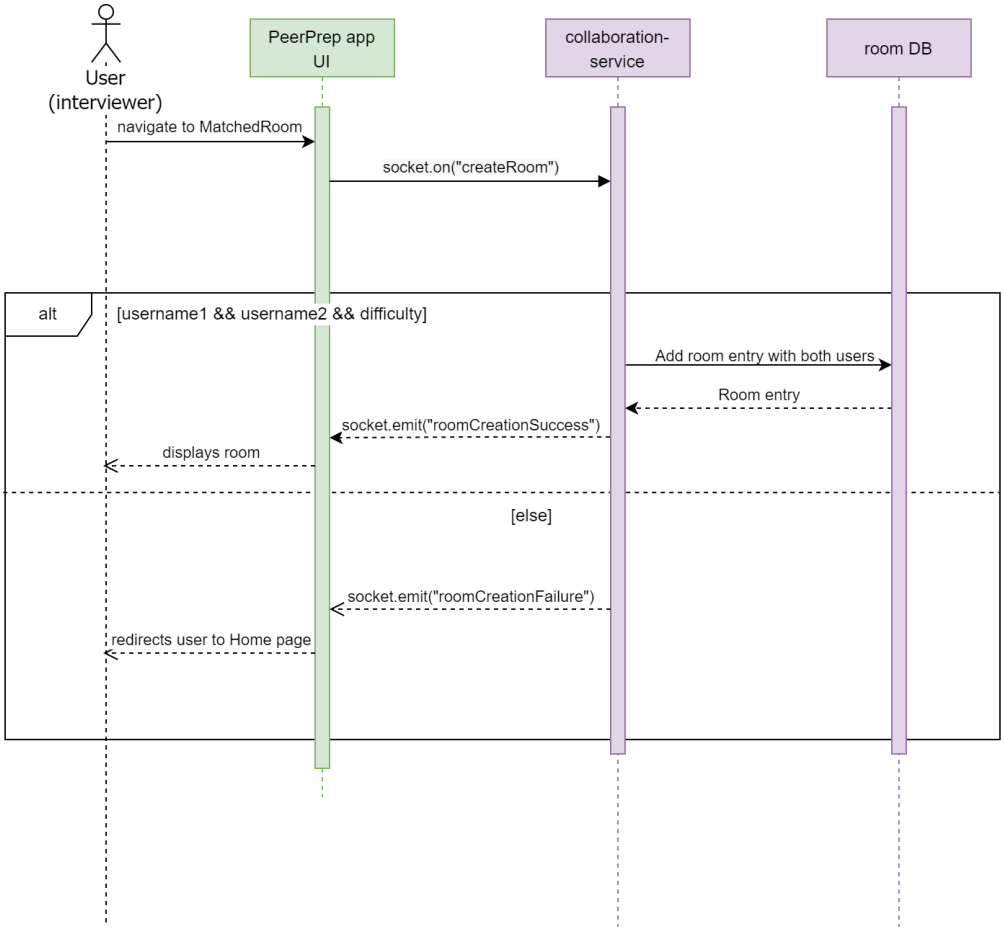


Fig 6.2.4.3: Sequence diagram of Room entry creation

The activity diagram below (Fig 6.2.4.4) shows the process flow of creating a Room entry:

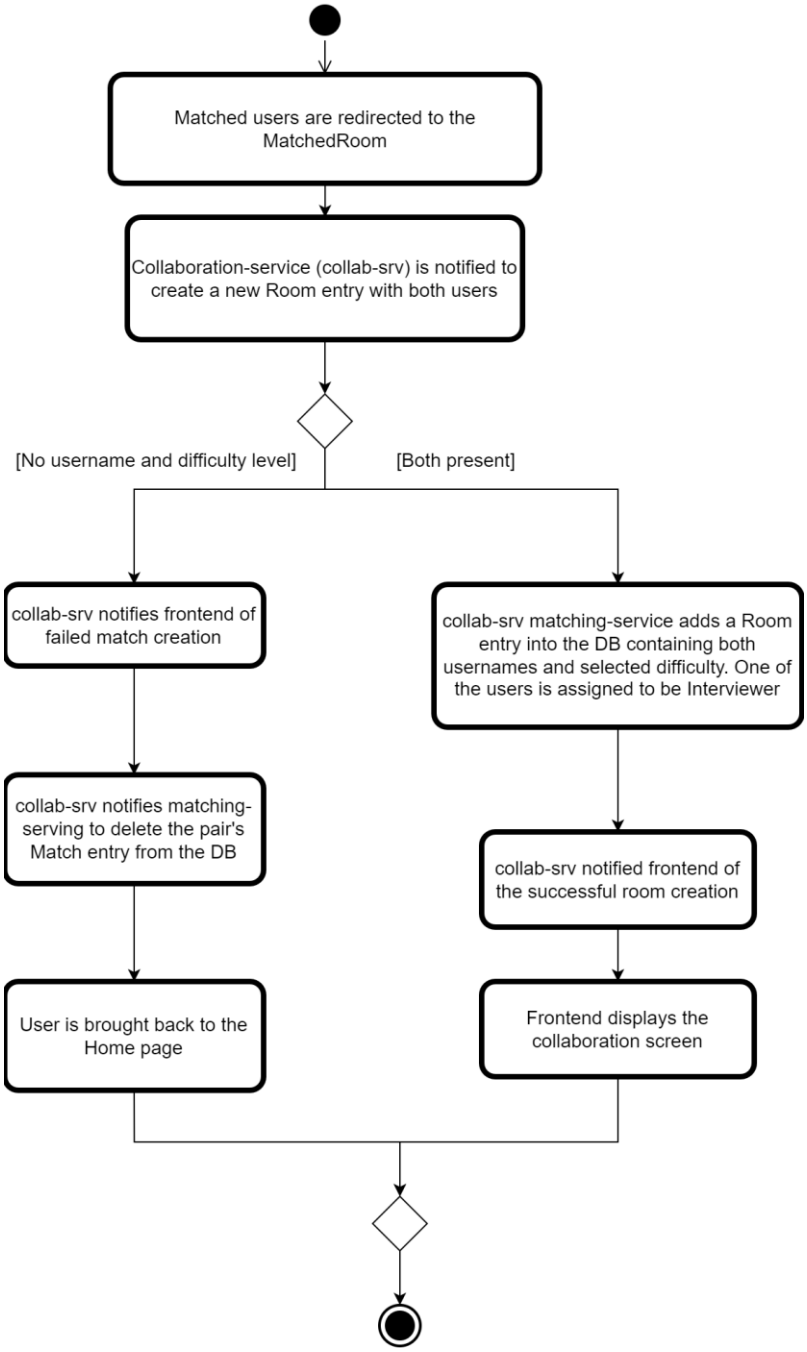


Fig 6.2.4.4: Activity diagram of Room entry creation

Displaying the shared question

Since the room is a shared space, we want to ensure that both users see the same question.

After the room is created successfully, the frontend tries to retrieve the question from the question service. If the question is retrieved successfully, the frontend will emit the event “setQuestion” via the socket connected to the collaboration service. The collaboration service listening to the “setQuestion” event in index.js is informed, and calls the `room-controller#setQuestion(req, socket)` method. The body of the req passed to the method specifies the roomId and question retrieved from question service. The room-controller then calls the `room-orm#ormSetQuestion(roomId, question)` method. This

method updates the question and question history in the Room entry associated with the roomId.

On successfully updating the Room entry, the room-controller will emit an event “questionSet” with the question data. The frontend on receiving the event “questionSet” will display the question in MatchedRoom.

The sequence diagram (Fig 6.2.4.5) below illustrates what has been explained above and shows the interactions between components to display the shared question:

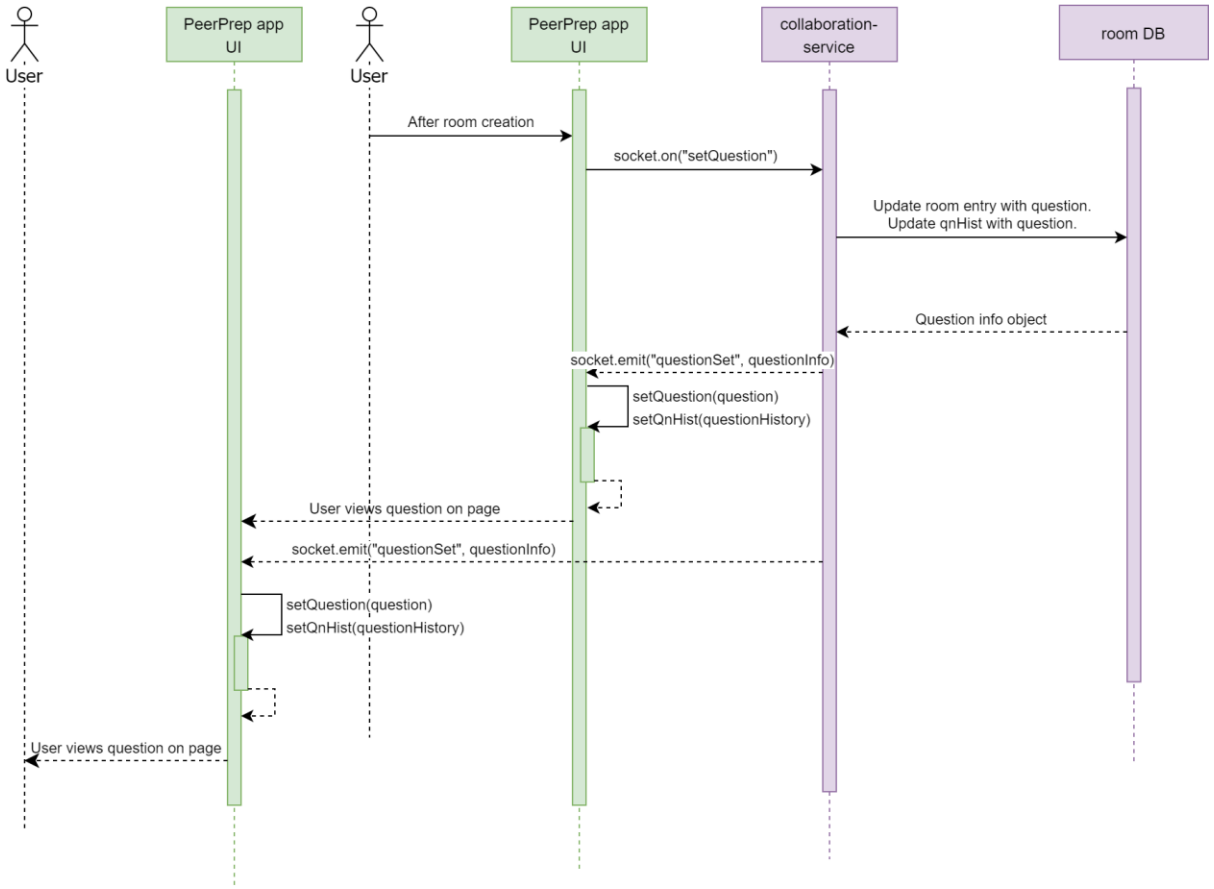


Fig 6.2.4.5: Sequence diagram of displaying shared question

Typing in the collaboration space

In our PeerPrep application, we simulate the interview process by having only one user as the interviewee. In this case, only the interviewee can type into the shared collaboration space.

When the interviewee makes a change in the shared space, frontend will emit the event “uploadChanges” via the socket connected to the collaboration service. The collaboration service listening to the “uploadChanges” event in index.js is informed, and calls the `room-controller#uploadChanges(req, socket)` method. The body of the req passed to the method specifies the roomId and document changed made. The room-controller then calls the `room-orm#ormUploadChanges(roomId, docChanges)` method. This method then updates the ‘code’ field in the Room entry associated with the roomId.

On successfully updating the Room entry, the room-controller will emit an event “documentUpdated” with the most updated content data. This event is emitted to the room channel that both users are subscribed to. The frontend of the other user (interviewer) on receiving the event “documentUpdated” will display the changes made by the interviewee in shared space.

The sequence diagram (Fig 6.2.4.6) below illustrates what has been explained above and shows the interactions between components when interviewee types into the shared space:

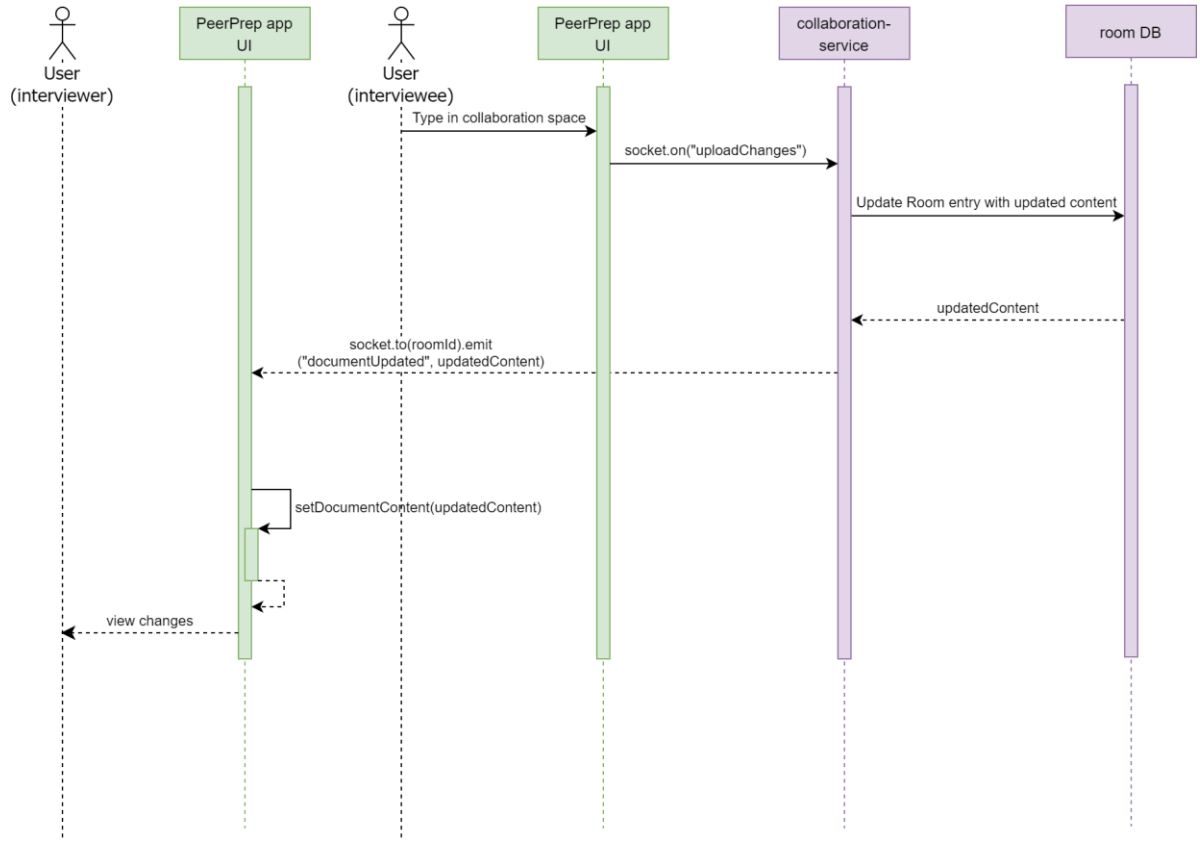


Fig 6.2.4.6: Sequence diagram of user typing in collaboration space

Checking if session has completed

A session ends when the user clicks on the ‘Finished!’ button or when the time counts down to 0. A session is complete when 2 sessions have passed and both users have swapped roles.

When a session ends, the user is redirected to the SessionEndedPage. On initial render of this screen, the frontend will emit the event “sessionEnded” via the socket connected to the collaboration service. The collaboration service listening to the “sessionEnded” event in index.js is informed, and calls the `room-controller#updateRoom(req, socket)` method. The body of the req passed to the method specifies the roomId. The room-controller then calls the `room-orm#ormUpdateRoom(roomId)` method. This method checks how many sessions have passed.

If the session is complete, the room-controller will emit an event “sessionComplete”. The frontend on receiving the event “sessionComplete” will then display the session completed screen. If the session is not complete, the `room-orm#ormUpdateRoom(roomId)` method will

update the number of rounds in the Room entry associated to the roomId. The frontend is not notified of the “sessionComplete” event and displays the default screen with a countdown of 10 seconds until the next session.

The sequence diagram (Fig 6.2.4.7) below illustrates what has been explained above and shows the interactions between components when checking if the session is complete:

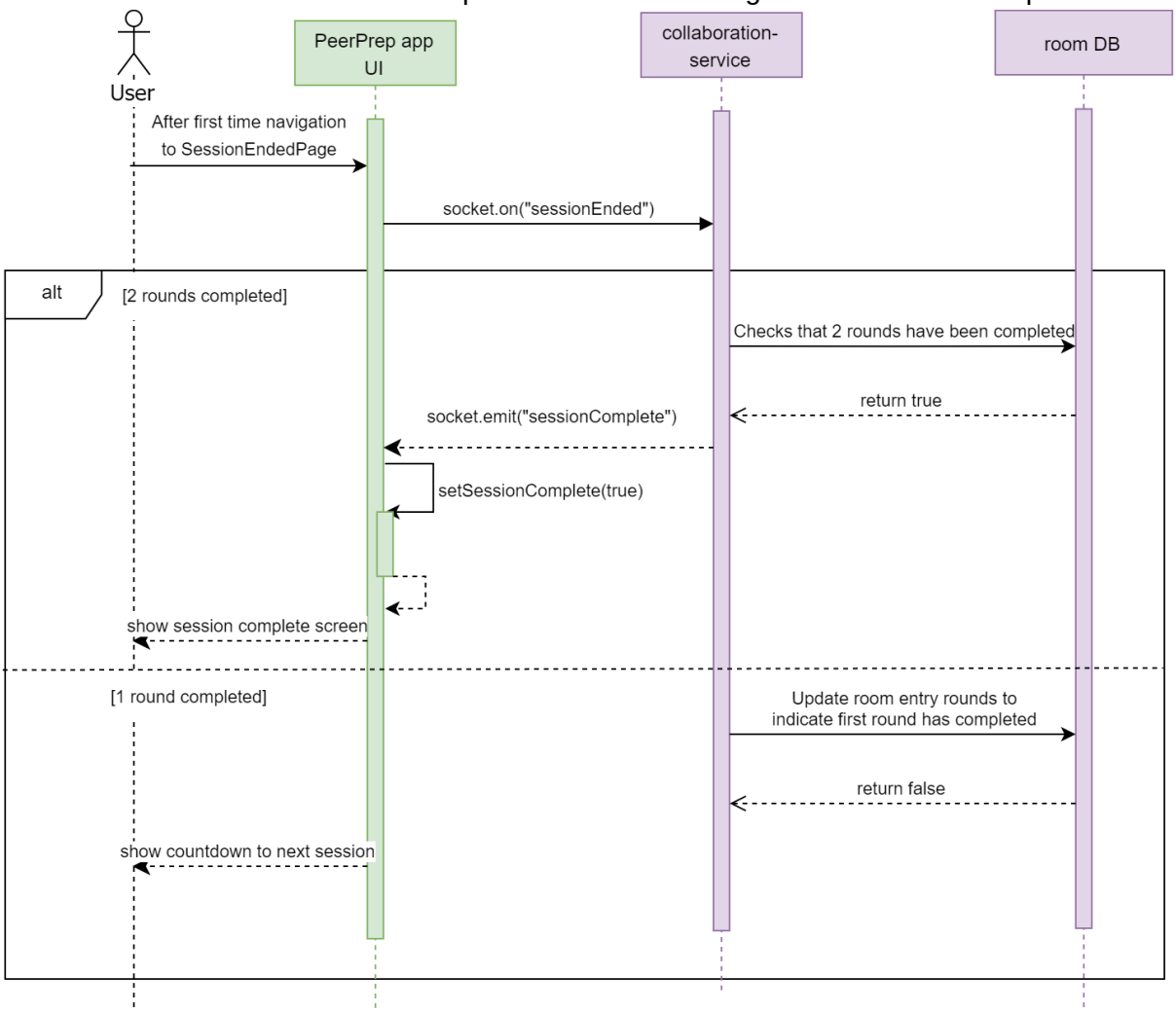


Fig 6.2.4.7: Sequence diagram of checking if session has completed

Swapping roles at the end of a session

At the end of the first session, both users will swap roles so they both get to experience both ends of the interview process.

When the first session ends, the user is redirected to the SessionEndedPage. On initial render of this screen, after checking that the session is not complete the frontend will emit the event “switchRoles” via the socket connected to the collaboration service. The collaboration service listening to the “switchRoles” event in index.js is informed, and calls the `room-controller#switchRoles(req, socket)` method. The body of the req passed to the method specifies the roomId. The room-controller then calls the `room-orm#ormSwitchRoles(roomId)` method. This method updates the ‘interviewer’ field in the Room entry associated with the roomId so that user 2 in the room is now the interviewer.

On successfully updating the Room entry, the room-controller will emit an event “switchedRolesSuccessful”. The frontend receives the event “switchedRolesSuccessful” and is notified that the role switching has been successful.

The sequence diagram (Fig 6.2.4.8) below illustrates what has been explained above and shows the interactions between components when swapping user roles:

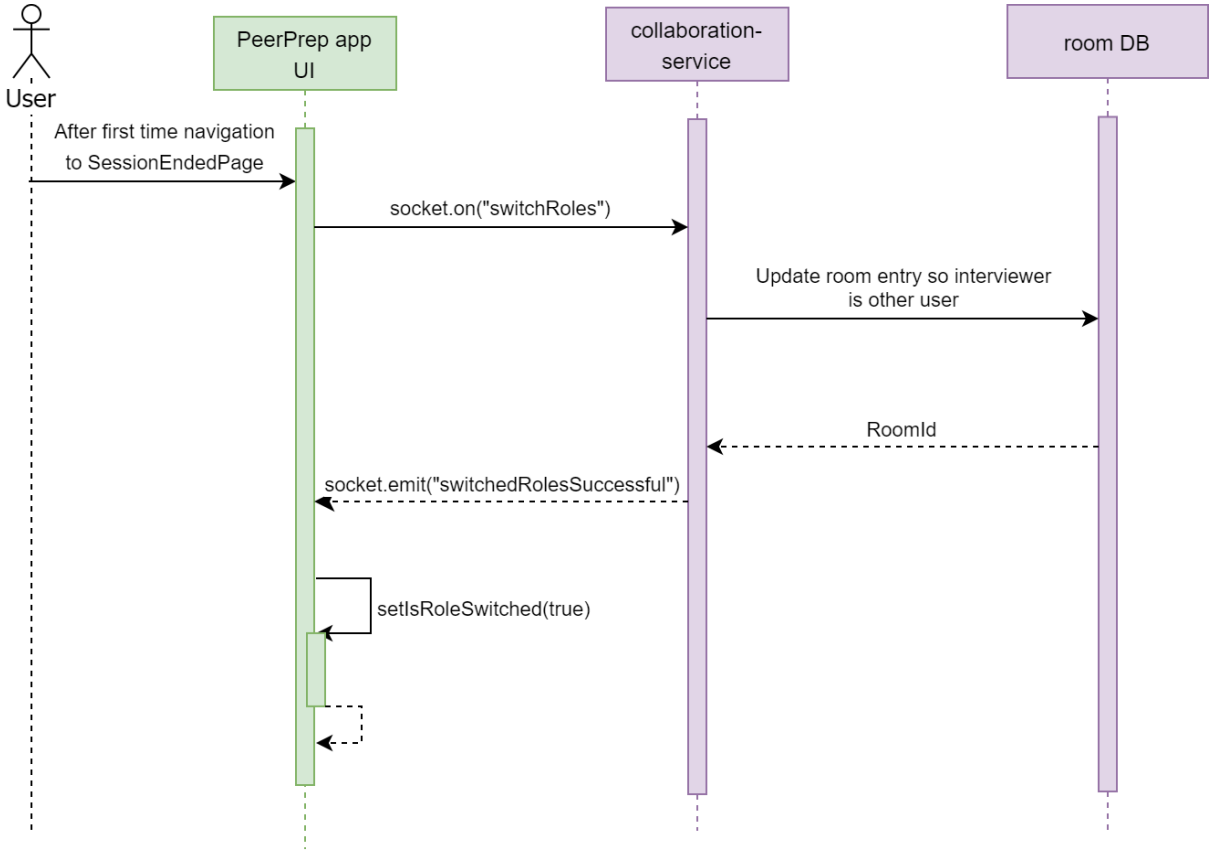


Fig 6.2.4.8: Sequence diagram of role swapping

Room entry deletion (user leaves session early/user returns to Home after session complete)

The user is allowed to leave at any time in the session, either at the collaboration space or the end session screen by clicking the ‘Back to Home’ button. When that happens, the room entry is deleted, since 2 users are required to continue collaboration. The match entry is also deleted, as detailed in the deleting of matches section above.

When the user clicks the ‘Back to Home’ button, the frontend will emit the event “sessionComplete” via the socket connected to the collaboration service. The collaboration service listening to the “sessionComplete” event in index.js is informed, and calls the `room-controller#deleteRoom(req, socket)` method. The body of the req passed to the method specifies the roomId. The room-controller then calls the `room-orm#ormRemoveRoom(roomId)` method. This method removes the Room entry associated with the roomId from the database roomDB.

On successful deletion of the Room entry, the user is redirected to the Home page. The room-controller also emits an event “oneUserLeft” to the room channel that both users are

subscribed to. The frontend of the other user on receiving the event “oneUserLeft” will change the display based on which screen they are currently on. If they are at the first SessionEndedPage, the countdown will stop and a message will inform the user that their partner has left the session, with a button to return to the Home page. If they are at the second SessionEndedPage, a similar view will be shown. If they are at the MatchedRoom, the collaboration space will be cleared to be replaced by the same message and button.

The sequence diagram (Fig 6.2.4.9) below illustrates what has been explained above and shows the interactions between components when deleting a Room entry:

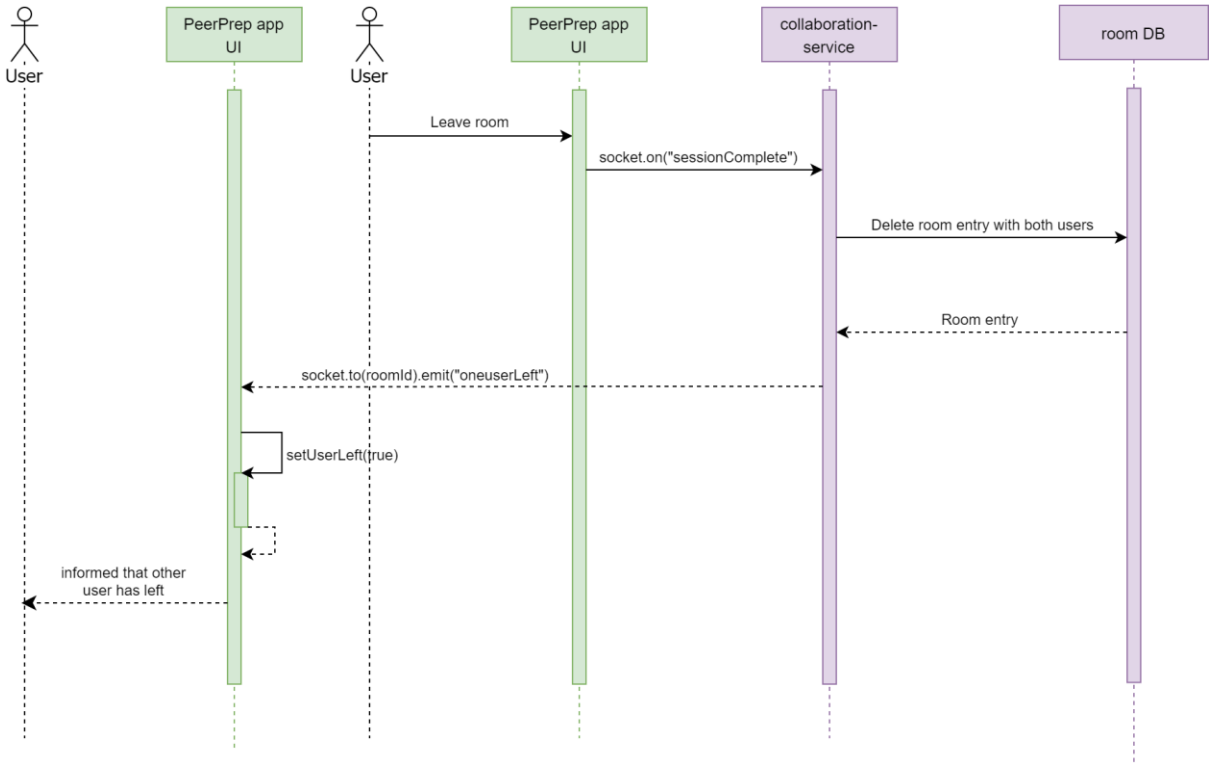


Fig 6.2.4.9: Sequence diagram of Room entry deletion

The activity diagram below (Fig 6.2.4.10) shows the process flow for the user in the role of **interviewee** of checking if a session is completed, swapping roles and room entry deletion. The extended activity diagram for User leaves early is shown in Fig 6.2.4.12:

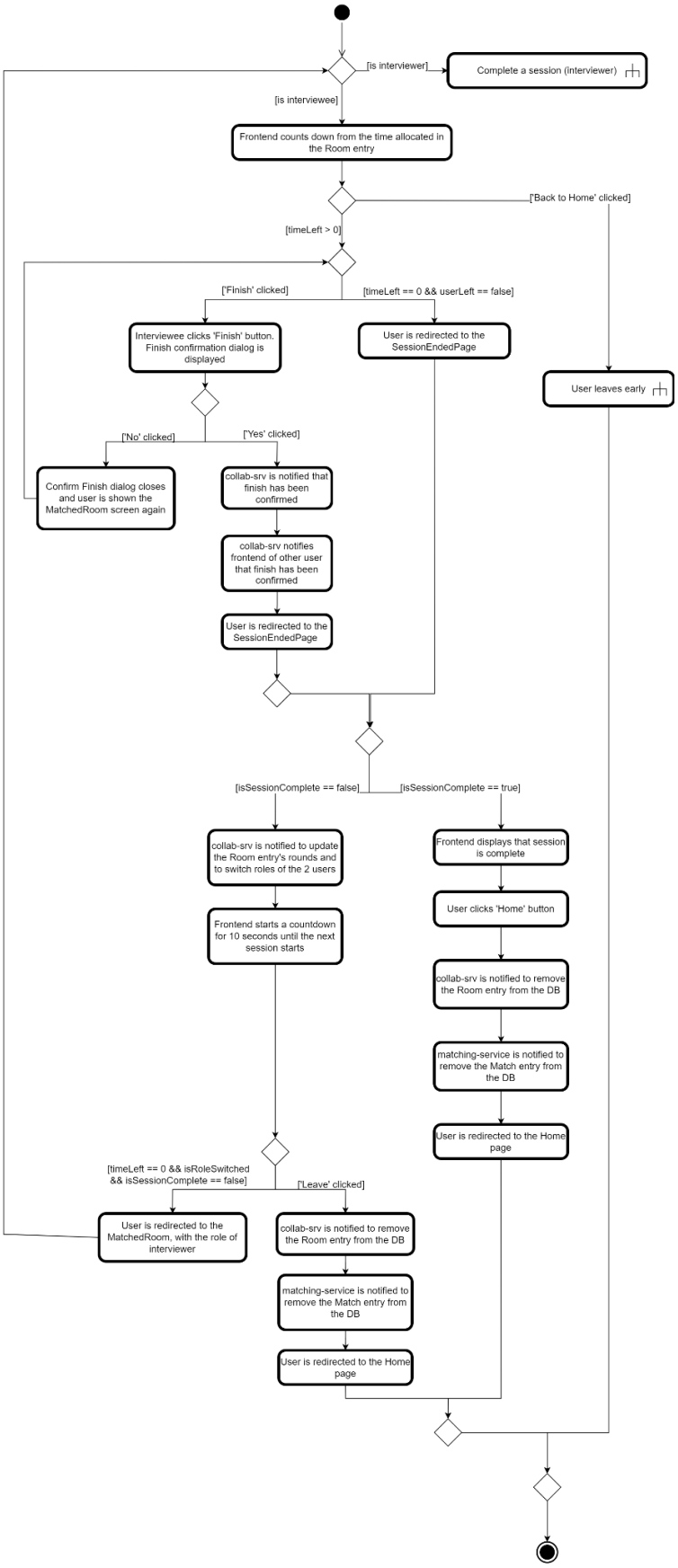


Fig 6.2.4.10: Activity diagram of interviewee completing an entire session

The activity diagram below (Fig 6.2.4.11) shows the process flow for the user in the role of **interviewer** of checking if a session is completed, swapping roles and room entry deletion. The extended activity diagram for User leaves early is shown in Fig 6.2.4.12.

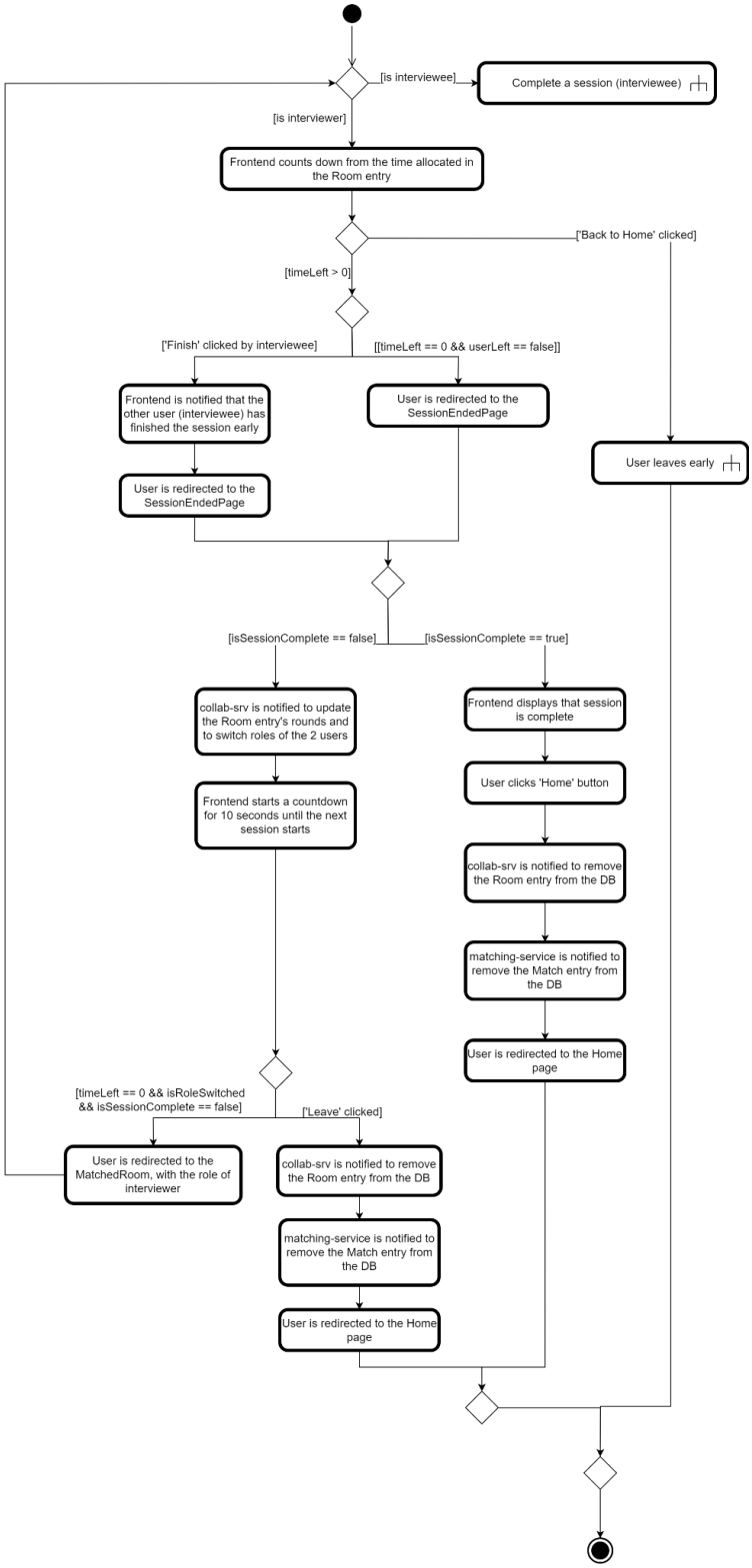


Fig 6.2.4.11: Activity diagram of interviewer completing an entire session

The activity diagram below (Fig 6.2.4.12) shows the process flow for a user leaving early:

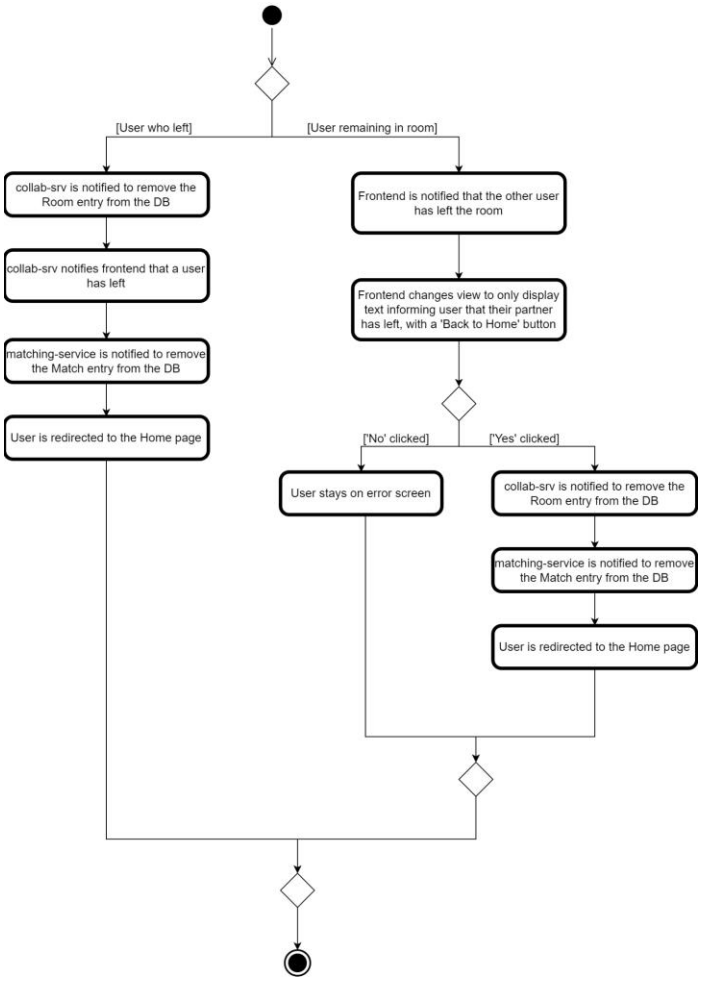


Fig 6.2.4.12: Activity diagram of a user leaving early

6.2.5 Question Service

The question service is responsible for retrieving a question of difficulty level specified by the users in the same room. The question chosen must not be previously given to the users.

Question Service contains REST API that interacts with the frontend.

Architecture

This section shows the structure and logic of the question service of PeerPrep. It shows a generalised view of question service microservice.

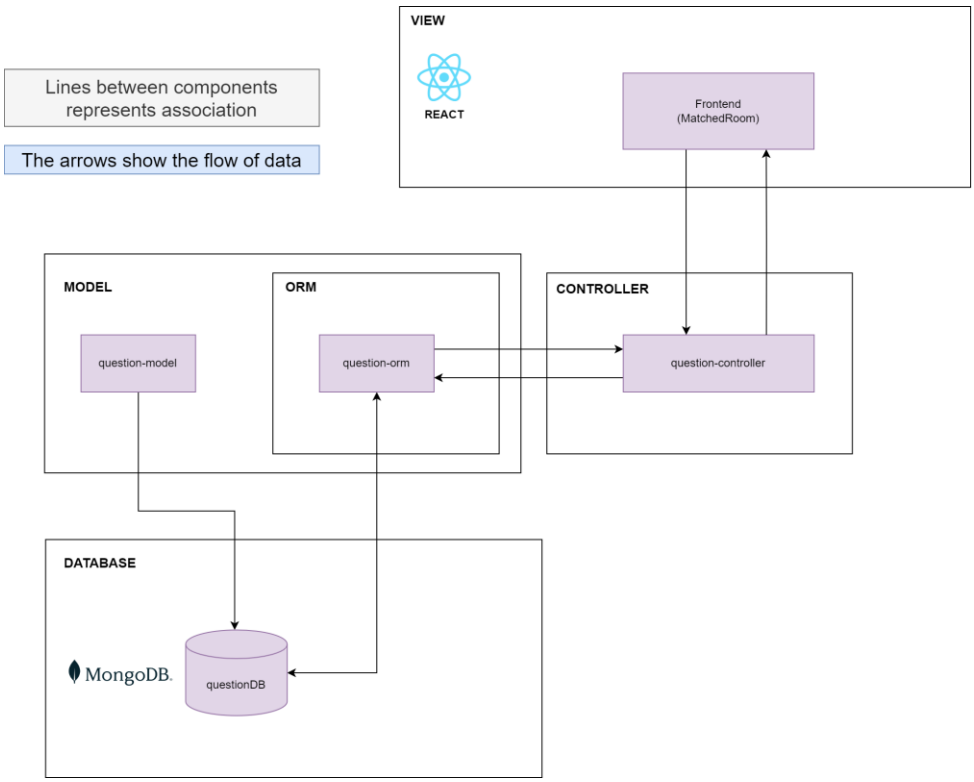


Fig 6.2.5.1: Architecture of question service

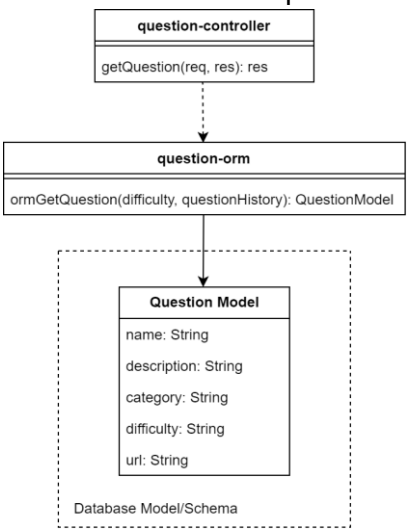


Fig 6.2.5.2: Class diagram of question service

- The `Question Controller` interacts with both the `Frontend` and `Question ORM` and acts as a Mediator to send data to and from the 2 classes
- The `Question ORM` interacts with the database directly and deals with the retrieval of information from the database

QuestionModel Database Schema

Questions were manually collected from LeetCode.

Field	Data type	Field Explanation
name	string	Name of the question
description	string	Description of the question
category	string	Topic that the question tests on
difficulty	string	Difficulty of the topic – easy, medium or hard
url	string	Link to the LeetCode website for the question

Features aka APIs

Retrieve Question Feature

Question retrieval occurs automatically after a room is successfully created. Frontend calls the API to retrieve a question and and relevant data from the room (difficulty level and question history) will be sent to `question-controller#getQuestion` method in Question Service to be validated. After successful validation, question-controller passes the data to `question-orm#ormGetQuestion` method in Question Service and a question is retrieved.

Below are the status codes and messages that the frontend will receive after the API call:

Status Code	Message	Explanation
200	Question retrieved	
400	Difficulty and/or Question History are missing!	API was not provided necessary data
500	Database failure when retrieving question from database!	This is most likely due to connectivity issue with the database or network

Fig 6.2.5.3 and Fig 6.2.5.4 show the process of question retrieval.

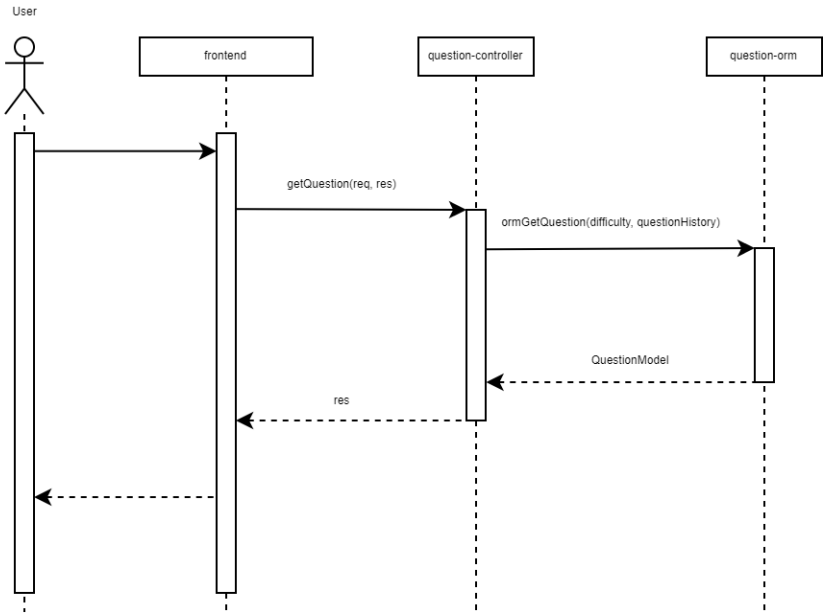


Fig 6.2.5.3: Sequence Diagram for question retrieval

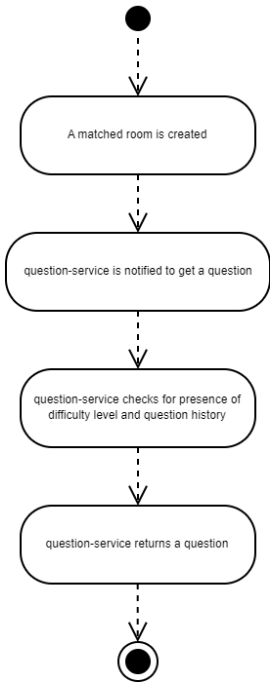


Fig 6.2.5.4: Activity Diagram for question retrieval

6.2.6 Communication Service

The communication service is responsible for providing chat functionality in the collaboration screen. The chat allows the users to discuss and work on the problem together or allows the interviewer user to ask technical questions. The communication service should also inform users when a new chat message is available to them.

Architecture

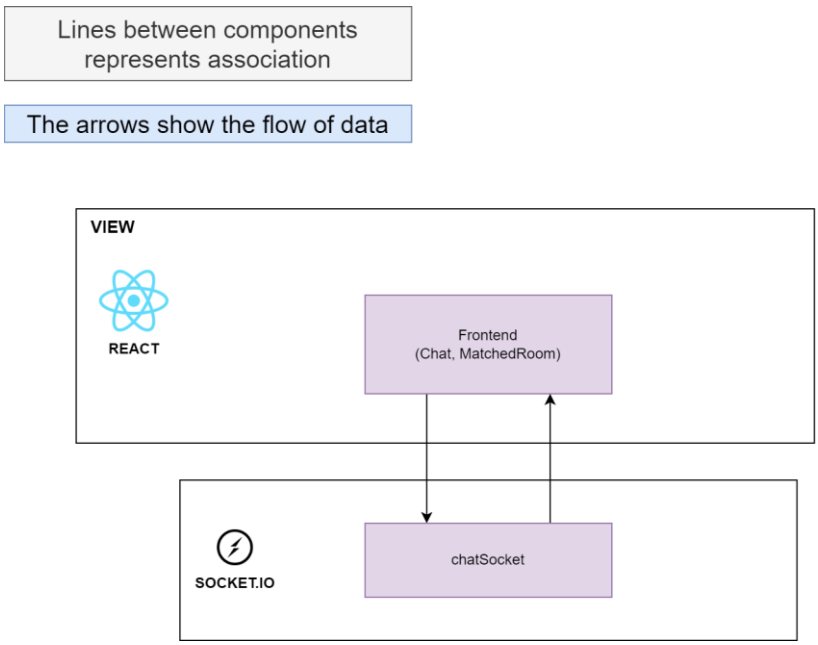


Fig 6.2.6.1: Architecture of communication service

The figure below shows the class diagram of the Chat component in the frontend:

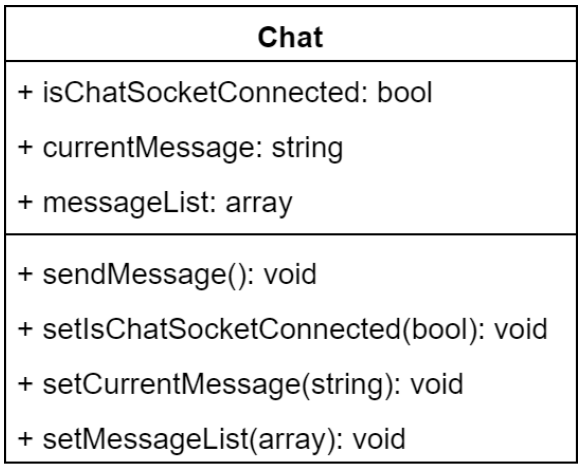


Fig 6.2.6.2: Class diagram of communication service

Features aka APIs

Sending and receiving chat messages

When the chat box first gets rendered, the frontend emits the event “start chat” via the socket connected to the communication service. The communication service is notified of the event and subscribes the chat socket to a channel with the same roomId.

When the user sends a chat message, the frontend will emit the event “send message” along with the message data via the socket connected to the communication service. The communication service is notified, and emits the event “receive message” and message data to the room channel that both users are subscribed to. The frontend of the other user is notified of the event and adds the message to its current state of messages. The chat message is thus synchronised and shown on both users’ screens.

The communication service also emits the event “hasNewMessage” to the shared channel. The frontend of the other user is also notified and shows the notification for a new message.

The sequence diagram (Fig 6.2.6.3) below illustrates what has been explained above and shows the interactions between components when a user sends a chat message:

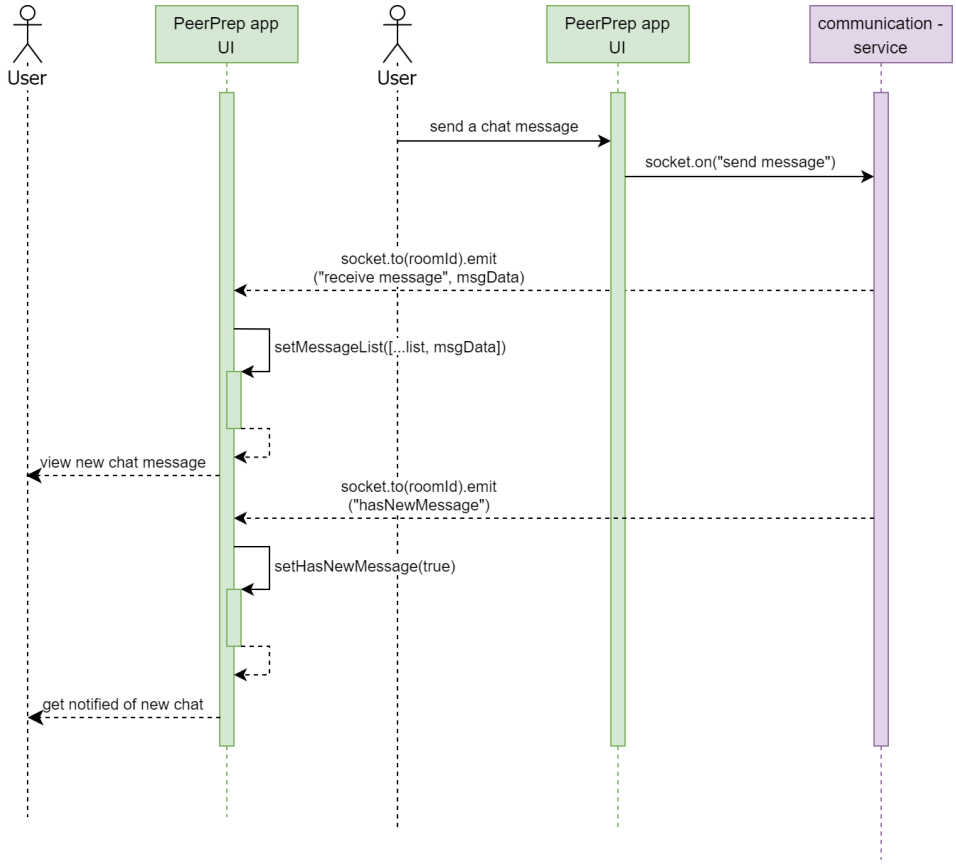


Fig 6.2.6.3: Sequence diagram of user sending a chat message

The activity diagram below (Fig 6.2.6.4) shows the process flow for a user sending a chat message:

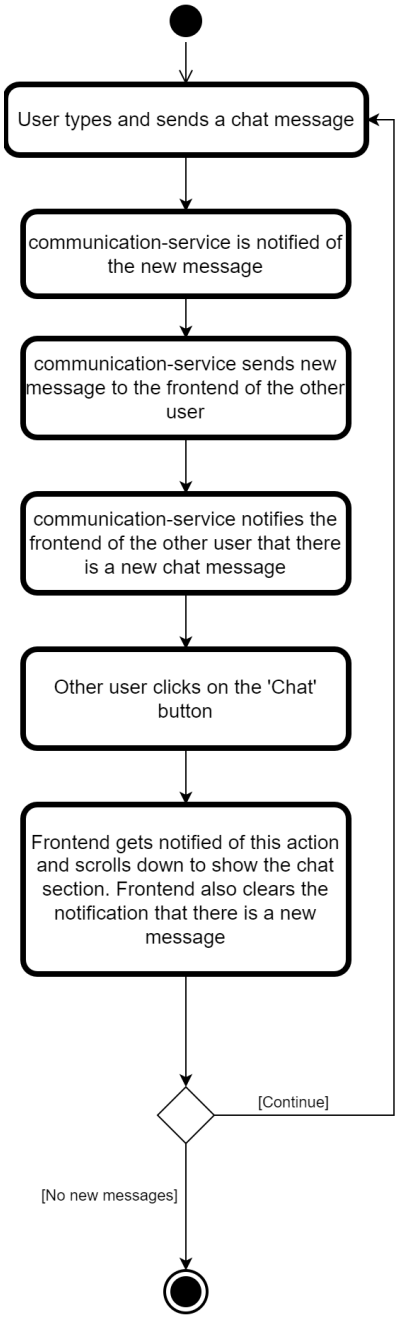


Fig 6.2.6.4

6.3 Development Process

Project Management

- Before each major milestone, we have a sprint planning by deciding on the features to be developed, writing out the user stories and coming up with functional requirements based on the user stories. We then split the workload from there
- We conduct weekly stand ups on Thursdays to update each other about what has been done, and to plan for the features to be developed by the next stand up.
- We also use GitHub issues to keep updated on each others progress

Continuous Integration

Continuous integration was set up to ensure that new contributions to the project do not affect the correctness of the application. Since the automated test cases are run with every pull request or push to the ‘main’ branch, incorrect behaviour can be detected earlier. This allows us to rectify errors promptly. Continuous integration also makes the testing process less time-consuming compared to triggering the automated tests manually on a local machine every time changes are merged into the main branch.

The figure below (Fig 6.3.1) shows the overall flow of the continuous integration:

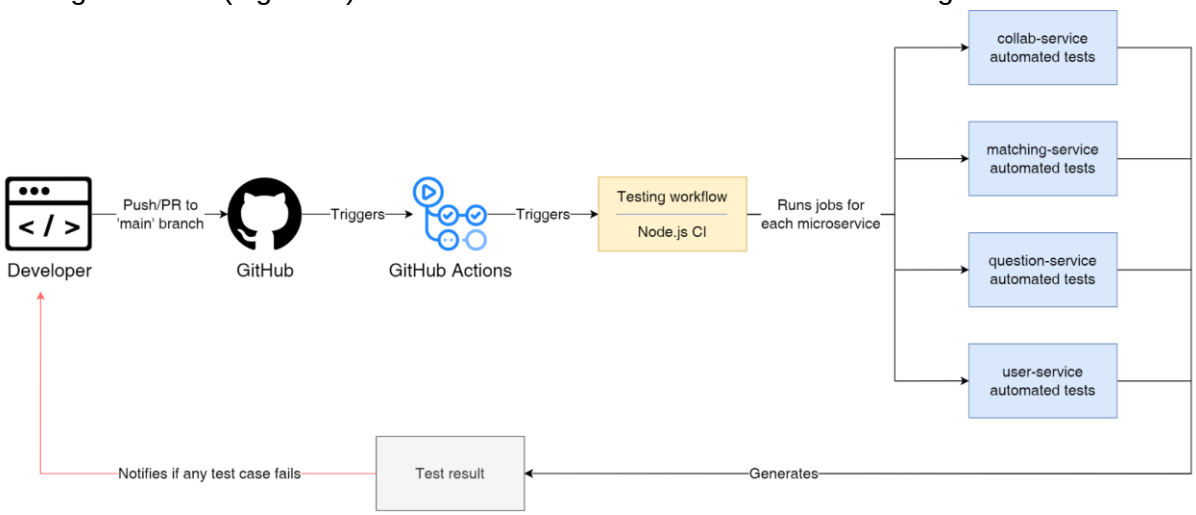


Fig 6.3.1: Overall CI flow

GitHub Actions were used to run automated tests for continuous integration. The CI workflow consists of 5 jobs, and each job runs the tests cases for one of our backend microservices: user service, matching service, collaboration service, question service and communication service.

Each job is tested on 2 versions of Node.js, 16.x and 18.x. The user service and question service jobs are also tested on 2 versions of MongoDB each, 5.0 and 6.0. On pull request or push to the ‘main’ branch, the CI workflow runs all 5 jobs and displays the results of the testing. In case of failed jobs, the developer who opened the PR or pushed a commit to the ‘main’ branch will be notified. The errors will then be rectified and pushed to the repository.

6.4 Deployment

PeerPrep can be deployed both on the local machine using native technology stack, and to a local staging environment via Docker. This fulfils our NFR2 and NFR3.

Deployment using a native technology stack uses 6 ports, one for each of the backend microservices, and one for the frontend. The frontend is connected to the backend microservices explicitly via the ports each backend microservice is being run on. To run the application, the user has to start up each microservice individually.

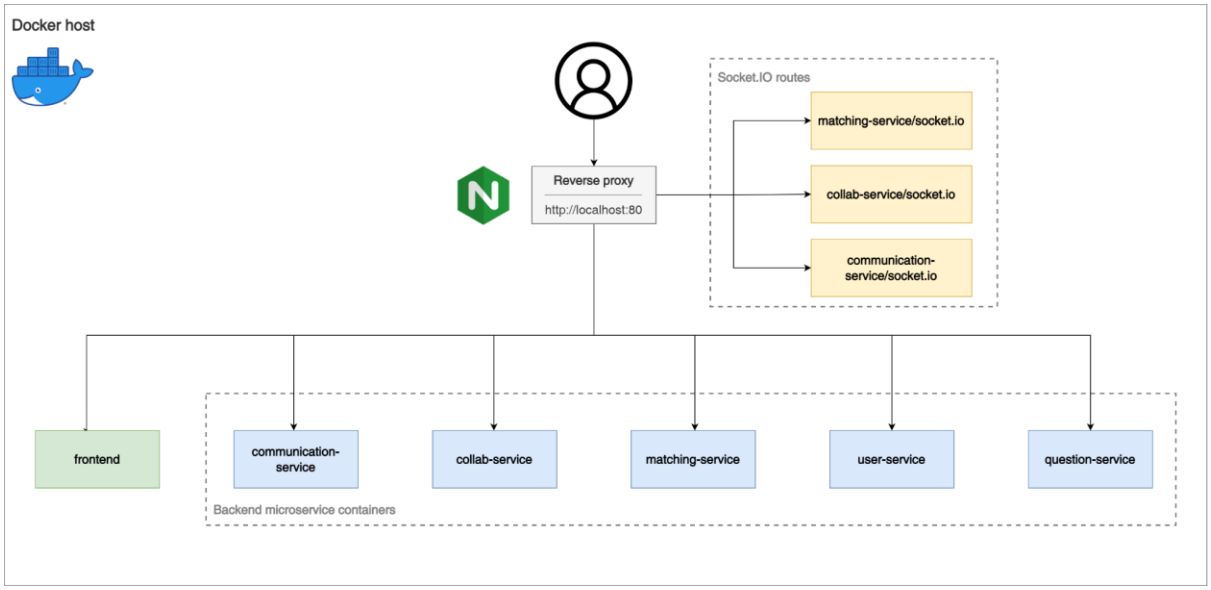


Fig 6.4.1: Docker containerisation

The Dockerized application consists of 7 containers: one for each backend microservice, one for the frontend, and one for the Nginx reverse proxy. This is shown in the figure above (Fig 6.4.1) An Nginx reverse proxy allows the user to directly access the application via a single port without having to understand the backend connections and routing of the microservices. Due to the usage of Socket.IO for 3 of our microservices, the Nginx reverse proxy also has to handle the routing of messages received via each socket.

The frontend of the application is served on http://localhost:80, and each of the backend microservices are served on http://localhost:80/<microservice name>. Hence, after building the Docker containers, the user can access the application via http://localhost:80.

7 Improvements & Enhancements

Due to the tight timeline of the project, limited features could be implemented. We have identified features that we would have developed given more time. Furthermore, we also considered some improvements that could have been made to existing features.

7.1 Improvements

Firstly, the UI of our application can be made more cohesive. Selecting and applying a uniform style across all pages provides the user with a more seamless experience. This leaves a good first impression on new users and encourages them to continue using PeerPrep.

Currently, the application only allows users to select the difficulty level of the question they would like to attempt. The application can also allow users to select the topic of the question, on top of selecting the difficulty level. Different users are weak in different topics. Implementing a topic selection allows users to get more practice in the topics they are weak

in, rather than practising topics they are already familiar with. This allows users to improve more quickly by focusing their practice on specific topics.

Another improvement that can be added is a code editor with syntax highlighting features. Users are used to seeing code that is neatly indented and highlighted when coding with code editors or IDEs. However, PeerPrep currently displays the code as plain text. To allow users to view code in a format that is more readable and familiar to them, syntax highlighting and indentation features can be added. This removes the need for users to worry about the syntactic accuracy of their code, since the code editor will point out errors to them.

Lastly, support for multiple programming languages can be integrated into PeerPrep. Building on to the previous improvement, the code editor can be configured to perform syntax highlighting for various languages. Each user has different languages they are familiar with or would like to practise with, and with this improvement, users can practise in their chosen language. This makes the practice process more comfortable and user-friendly. For users hoping to familiarise themselves with a specific language, the syntax highlighting provided by the code editor will be useful in learning the correct syntax as well.

7.2 Enhancements

One enhancement that can be added to the application is the inclusion of learning pathways. This provides the history of questions that have already been attempted by the user.

After the users have attempted a question, the application can also check for the correctness of the answer entered by the interviewee. This helps users identify where they have gone wrong while writing their solution. The checking of correctness can be done by running the interviewee's code through some test cases. The test cases should check common cases, as well as corner cases to ensure that the interviewee's solution covers them all.

Lastly, on top of the chat, a voice call or video call feature can be added to allow for more convenient communication. Since the interviewee will be busy typing their solution to the question, it is inconvenient to reply to the chat. With the voice or video call, the users can type and discuss at the same time.

8 Reflection

This project gave us many takeaways, both in technical and non-technical aspects.

In technical aspects, we have applied what we were taught in lecture to this project. For example, learning the different architecture designs such as MVC, MVA or MVVM helps us to structure our project well and gives us direction on how to design each microservice. Furthermore, the importance of functional and non-functional requirements also helps us to narrow our focus on requirements that are of higher priority and value given the limited time we have to do the project. Learning about the various design principles also helps to ensure that our code is clean, understandable and structured.

In non-technical aspects, communication and teamwork are the main factors for the project to succeed. We set weekly goals for ourselves and do our best to achieve them. We also set up weekly stand-ups to update on our progress and also to settle any doubts/blockers that we had. Everyone was extremely busy this semester, but was understanding and willing to help out whenever needed.

Overall, It was also a great experience for us to build a web application from scratch and to build something that is relevant and useful to us.

9 Appendix

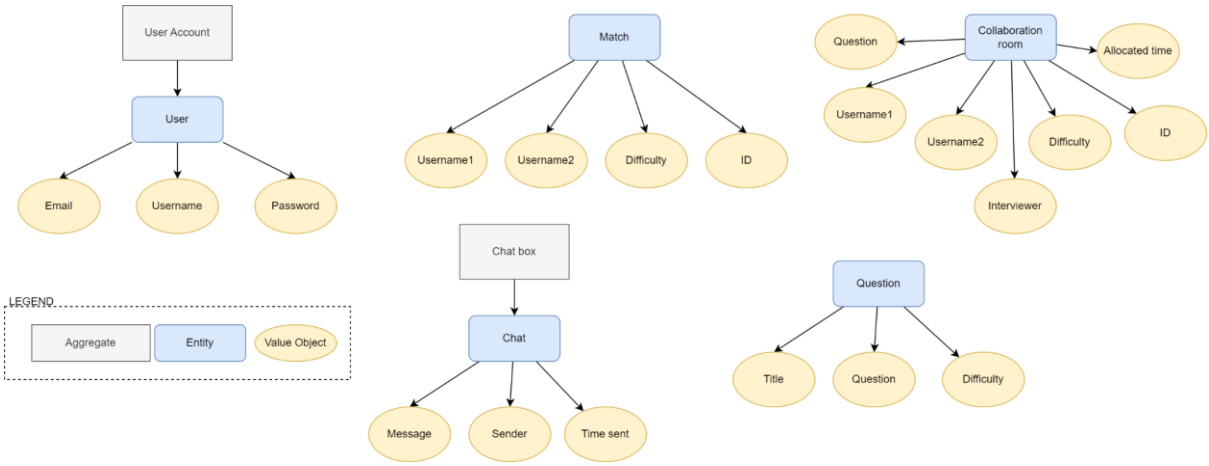
9.1 Justification for High Level Architecture

Before we developed the microservices, we went through the process of identifying the services that should be implemented and what responsibilities each service should have. The following section explains our domain analysis and subsequent microservice identification.

Domain decomposition:

- User creates an account and logs in
- User selects the question difficulty level to attempt (easy, medium or hard)
- User waits until he is matched to another user who has selected the same difficulty level
- Time out after no successful match in 30 seconds
- After successful matching, user is provided a question and a text field to type the solution
- Text field should be updated in real time for collaboration
- User can send a chat to other user
- User can Finish the session when he is done with the question
- User can then log out

Entities and value objects in the domain:



We broke down the domain to the following sub-domains:

- User log in, log out
- User sign up
- User update password
- User reset password
- Difficulty selection
- User matching based on difficulty selection
- User getting question based on difficulty selection
- User real-time collaboration
- User real-time communication

From the sub-domains, we decided on some bounded contexts:

Sub-domains	Bounded context
• User log in, log out • User sign up	User authentication
• User update password • User reset password	User account management
• Difficulty selection • User matching based on difficulty selection	User matching
• User getting question based on difficulty selection	Question management
• User real-time collaboration	Collaboration space
• User real-time communication	Chat space

Microservices identified from bounded contexts:

Bounded contexts	Microservices
User authentication	User service
User account management	
User matching	Matching service
Question management	Question service
Collaboration space	Collaboration service
Chat space	Communication service

9.2 Justification for FRs and NFRs

This section explains the reasoning behind the addition of some select FRs and NFRs

9.2.1 FRs

The full list of FRs can be found in Section 4.

S/N	Functional Requirement	Justification
User Service		

FR1.7	The system should allow users to reset password by providing email	To provide users who forget their password a way for them to still continue using PeerPrep, by resetting their password through their email
FR1.8	The system should allow users to create an account with email.	Users should be able to create an account and verify their account through a valid email and this ensures fraud prevention In future extension, email can be used to reply to notifications
FR1.9	The system should reject invalid passwords based on password length and strength restrictions	With complex password, any user's account will not be hacked into so easily
Matching Service		
FR2.2	The system should be able to match two waiting users with similar difficulty levels and put them in the same room.	Matching users who chose the same difficulty level allows both users to practice a question of a level that is comfortable for them.
FR2.3	If there is a valid match, the system should match the users within 30s.	To add an upper limit to the time the user waits for a partner to be found so they do not wait indefinitely
FR2.4	The system should inform the users that no match is available if a match cannot be found within 30 seconds.	To ensure that the user does not keep waiting when no partner can be found and the user can perhaps try finding a partner for another difficulty level instead
Question Service		
FR4.4	The system should not return a question previously attempted in the same room.	To prevent users from wasting their time to do a question that they have already attempted.
Collaboration service		
FR3.4	After the time is up for the first interviewee, the interviewer and interviewee switch roles and a new question is given	This allows both users to experience both roles in the simulated technical interview setting
FR3.5	Matched user should be able to leave the room at any time	Users should not be forced to stay in the application and should always be free to exit when needed
FR3.8	Interviewee can finish the session early if they have completed the question	It is likely that the user may finish solving the problem before time runs out. In that case, the user will not have to wait until time runs out until they can proceed to the next

		session
FR3.9	Interviewer should move on to the next session as well when interviewee ends the session early	This is to ensure that both parties are synced and are able to collaborate in real time
FR3.10	Collaboration session should end when one of the users leaves the session	
FR3.12	The system should give users 10 seconds to choose if they want to leave before the next session starts	This gives users some time to decide if they want to continue the next round of the session or quit early
Communication service		
FR5.1	Users in the same room should be able to see each others chat messages	This ensures that both users can discuss the problem together
FR5.2	Users should be notified when a new chat message arrives	This ensures that the receiver of the new chat messages does not have to constantly scroll down to check if there are any new messages

9.2.2 NFRs

The full list of NFRs can be found in Section 5.

S/N	Non-Functional Requirement	Justification
NFR1	A user can create an account successfully more than 90% of the time.	To ensure that our application has user-friendly UI and is accessible to new users for them to practice their technical interview skills. The other 10% is for errors out of the application's control such as email servers going down for example.
NFR2	The system should be able to run locally with the native tech stack on Windows, Mac and Linux	To ensure that all users can run the application
NFR3	The system should be able to run as a Docker container on Windows, Mac and Linux	To make the process of running the application more convenient by removing the need for users to manually start up each microservice individually
NFR4	The system shall backup the data typed in the collaboration space in	To ensure that if there are data loss, the users can get the last backed up data (for a future extension). Both users can then view

	the database to ensure information typed on both users ends will not be lost	the correct code that was entered by the interviewee so they can discuss it
NFR5	The system shall sync the information typed in the shared collaboration space within 5 seconds	To ensure that the users view the latest code so they can discuss it
NFR6	The system should load the room within 15 seconds of users being matched	To ensure that the practice process is efficient and users do not waste time waiting
NFR7	The system should only allow users to access the main pages e.g. Home, MatchedRoom etc after they have logged in with a valid username and password	To prevent unauthorised access to the application

9.3 User Stories

In the process of coming up with the FRs, we considered and prioritised the user stories listed in this section. The FRs mapped from each user story is listed in the FRs column. A full list of FRs can be found in Section 4.

9.3.1 User Service

User Story	Priority	FRs
As a new user, I want to create an account, so that I can start using PeerPrep.	High	FR1.1, FR1.8, FR1.9
As an existing user, I want to be able to log in to my account.	High	FR1.3
As an existing user, I want to be able to log out of my account.	High	FR1.4
As an existing user, I want to delete my account, so that I can remove my data from PeerPrep.	High	FR1.5
As an existing user, I want to reset my password, so that I can continue using my PeerPrep account after forgetting my password.	Medium	FR1.7
As an existing user, I want to update my password in case someone knows it	Medium	FR1.6
As an user, I want to make sure that I have an unique username so that I would not be worried of	High	FR1.2

another having the same username and thinking that my account got hacked		

9.3.2 Matching Service

User Story	Priority	FRs
As a user, I want to select the difficulty level of the question provided so that I can attempt a question that matches my ability level.	High	FR2.1
As a user, I want to be matched with a partner who selected the same difficulty level within a short amount of time so that we can simulate an interview situation.	High	FR2.2, FR2.3
As a user, I want to be notified if a partner who chose the same difficulty level cannot be found after a period of time so I do not wait to be matched indefinitely.	High	FR2.4, FR2.7
As a user, I want to be notified when the session is over so I can leave the session or choose to continue practicing with the same partner.	High	FR2.6

9.3.3 Collaboration Service

User Story	Priority	FRs
As a user, I want to be added to the same collaboration room as my partner, so that I can collaborate on a problem with my assigned partner.	High	FR3.2
As a user, I want to be assigned a role randomly that is different from my partner, so that we can simulate a technical interview.	High	FR3.11
As a user, I want to try both roles in a collaboration session, so that I can experience both roles in an interview.	High	FR3.4
As a user in the interviewee role, I want to type my solution into a shared space, so that my partner can see my thought process and we can collaborate.	High	FR3.1
As a user in the interviewer role, I want to see what my partner has typed in the collaboration space, so that I can correct my partner's misconceptions and we can collaborate.	High	FR3.1
As a user, I want to finish the session early once I	Medium	FR3.8, FR3.9

am done with the question, so that I can move on to the next question.		
As a user, I want to be able to leave the room at any time, so that I am free to stop the session whenever I need.	Medium	FR3.5, FR3.10
As a user, I want to see the same question as my partner, so that we can refer to and collaborate on the same question.	High	FR3.2

9.3.4 Question Service

User Story	Priority	FRs
As a user, I want to choose from questions with different difficulty levels, so that I can train my technical skills in a progressive manner.	High	FR4.1
As a user, I want to get a question of the difficulty level that I have specified.	High	FR4.3
As a user, I want to get a question that I have not attempted before in the current room, so that I ensure I do not waste my time.	High	FR4.4

9.3.5 Communication Service

User Story	Priority	FRs
As a user, I want to chat with my partner in real time, so that I can discuss the problem with them.	High	FR5.1
As a user in the interviewer role, I want to chat with my partner in real time, so that I can ask them technical questions.	High	FR5.1
As a user in the interviewee role, I want to chat with my partner in real time, so that I can answer any technical questions they have posed.	High	FR5.1
As a user, I want to be notified when I get a new chat message, so that I can continue working on the problem without scrolling constantly to check the chat for updates.	Medium	FR5.2

9.4 Individual Contributions

9.4.1 Technical

Name	Contribution
Dora	● User service + Question service implementation backend logic ● Account management and authentication frontend (Login, Signup, Reset password, Delete account, etc) ● Home page frontend ● Implement mailing system backend logic ● Add testing for user service
Nicole	● Implement Matching service + Collaboration service backend logic ● Implement matched room page frontend ● Implement websocket communication between matching service and frontend ● Implement websocket communication between collaboration service and frontend ● Add testing for matching and collaboration service ● Set up continuous integration ● Docker deployment
Shen Yang	● User service + Question service implementation backend logic ● Introduction page frontend ● Implement question service frontend logic ● Improve UI for account management and introduction pages frontend ● Add testing for question service
Sihui	● Implement countdown page backend ● Implement difficulty selection component ● Implement match entry deletion on match finding timeout (matching service) ● Implement collaboration space where users see each others' changes in real-time (collaboration service) ● Implement session ended screen and backend logic ● Implement communication service frontend and backend ● Add testing for communication service

9.4.2 Non-technical

Name	Contribution
Dora	● Documentation ● Sequence, activity, architecture and class diagrams
Nicole	● Documentation ● Sequence, activity, architecture and class diagrams
Shen Yang	● Documentation ● Sequence, activity, architecture and class diagrams

Sihui	• Documentation • Sequence, activity, architecture and class diagrams

9.5 References

1. *Introduction*. SocketIO RSS. (2022, October 11). Retrieved November 7, 2022, from https://socket.io/docs/v4/
2. Richards, M. (no date) *Software architecture patterns*, *O'Reilly Online Learning*. O'Reilly Media, Inc. Available at: https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/app01.html#pattern-analysis-summary (Accessed: November 7, 2022).