CS3219: Final Report

Code Repository URL:
https://github.com/CS3219-SE-Principles-and-Patterns/cs3219-project-ay2122-2122-s1-g27

Group 27

Chia De Xun (A0182800M)

Eugene Tan Yew Chin (A0136174H)

Manas Vegi (A0191674W)

Vineeth Buddha (A0189789U)

Name	**Technical Contributions (in code)**	**Non-Technical Contributions & Roles**
Chia De Xun	Developed hexagonal microservice architecture Implement UserProfileService - User Management (Login, Registration) - Matching Algorithm and logic Integrate external services e.g. MongoDB, Redis, Socket.io Orchestration, Deployment, DevOps: Docker and AWS, GitHub Actions Unit and Integration Testing of services and full stack application Authentication middlewares	Team Lead & Backend Developer Overall architecture Backend API Design Documentation Components Interaction Design
Eugene Tan Yew Chin	Implement QuestionService - Room-User Management - Room-Question Management - Questions Management - Uniqueness of Rooms Implement CommunicationService logic - Text chat Unit Testing User Acceptance Testing	Gather Question Data Backend Design API Discovery Documentation Dedicated Backend Developer API & Schema Design Components Interaction Design
Manas Vegi	Implement Frontend CollaborationPage Integrate Frontend with Backend - Room determinism - Text-chat - Question retrieval - Code Editor Implement CollabSocketService logic - Code Editor User Acceptance Testing	API Discovery Documentation Frontend Developer Components Interaction Design

Vineeth Buddha	Implement Frontend Landing Page Implement Frontend Login Page Implement Frontend Matching Page Integrate Frontend with Backend User Acceptance Testing	API Discovery Documentation Frontend Developer Project Manager Components Interaction Design

Table of Contents

1. Introduction

1.1 Background

In a study conducted by the leading developer economy analytics company (Slashdata, 2019), it was found that the number of software developer jobs worldwide would double in the next decade. This upwards trend suggests that software engineering careers will become increasingly prevalent. Therefore, one would benefit greatly in terms of career prospects from preparing well for technical interviews. However, preparing for technical interviews is challenging as it requires simultaneous practice in many skills, such as communication, computational thinking and problem solving. This often leads to exhaustion and many times people may lose motivation when faced with difficult problems to solve.

Existing solutions, such as *LeetCode* and *AlgoExpert* have been created to empower people to prepare for technical interviews by practicing a myriad of questions spanning different topics and difficulties. However, they are lacking because they do not directly prepare their users in terms of **communication skills,** which is also a key trait employers look out for in technical interviews. Currently, jobseekers who want to practice their communication skills as well could use these platforms together with a friend to conduct mock interviews via a separate conferencing app, like Zoom. This, unfortunately puts jobseekers who may not have like-minded friends in the computing industry at a disadvantage as they have nobody to practice together with.

1.2 Purpose

Based on the identified pain points in the previous section, our group decided to implement PeerPrep, which is a web application with the following purposes:

1. Empower users to prepare for **technical interviews** by providing them with core services that existing players provide, such as a wide selection of typical whiteboard style interview questions asked by top technology companies.
2. Provide users with a **peer-learning platform** where they can conveniently be matched to like-minded peers (in terms of difficulty / topics) and the matched pair can practice collaboratively and conveniently.

2. Software Requirements Specification

In this section, we shall discuss the relevant Software Requirements Specification process and its items, from their creation to the eventual implementation status in the final project submission.

Do note that we have strived to identify comprehensive and user-centric requirements, which have been prioritized and deeply thought through to allow us to have a good visualization of the application we envision to create. However, also note that due to time and resource constraints, not all of these comprehensive requirements will be implemented. Instead, those mentioned in this section are everything the team has managed to gather thus far in-lieu of the long term interests of creating a user-centric and potentially commercially-viable product.

As for the scope of this project, the team recognizes the limited time and resources we have and applied an effective agile scrum process and project management to help us on deciding and implementing which features to prioritize for maximizing the user experience. These will be described in later sections.

2.1 User Stories

The team has gathered the following user stories as part of the requirements gathering process. Please note that user stories that had significant overlaps in functionality were removed as they do not add much value. Furthermore, a single user story may be related to multiple functional requirements (FR) and non-functional requirements (NFRs), via the requirements specification process. We have included the related FRs/NFRs for traceability in the table below.

Do note that the related FRs and NFRs are listed in the table below instead of a Requirements Traceability Matrix (RTM), as we realised that the RTM would be unsightly and confusing due to the sheer number of user stories, FRs, and NFRs. For more information on FRs and NFRs, please visit the respective sections (Refer to Section #2.2 for FRs and Section #2.3 for NFRs).

#	As a / an	I would like to	So that	Related FRs	Related NFRs
1	Computing student who may possibly find coding alone boring	Find people to match with to practice solving algorithm questions with	I can get their feedback and improve my skill-set in technical interview-style questions	FR1a FR1b FR1c FR1d FR1e FR1f FR2a FR4c FR4c1	3, 4, 5, 6, 7
2	Computing student	See a list of friends I	I can send them a	FR4a	-

	who has a few close friends	have previously connected to	notification when I want to code together with specific person	FR4b FR4c1 FR4d	
3	Aspiring Software Engineer preparing for technical interviews	Voice chat while practicing whiteboard interviews live together	I can improve my live interviewing	FR2b FR2b1 FR2b2 FR2b3 FR3b FR3c	2, 3, 4, 5, 8, 10
4	Shy computing student preparing for technical interviews	Message someone via chat while practicing coding questions	I can communicate without having to use my voice	FR2c FR3a	2, 3, 5
5	Programmer who needs more focus on certain topics	Filter on certain topics (e.g. Arrays, Linked Lists, Trees) to work on	I can improve my understanding of them	FR1a FR1b FR2d	9
6	Programmer who already has a reasonable amount of experience	Be matched with people of ‘similar’ skill sets/job roles, or easier difficulty	I can improve my communication while also revising relevant concepts	FR1a FR1g FR2d	11
7	Meticulous computing student	Easily select the difficulty level of the questions	I can organize my learning	FR1a FR2d	9
8	Student wanting to teach Computer Science Concepts	Be able to conduct mock coding interviews for selected students	Easily conduct lessons using the platform	FR2b FR2b1 FR2b2 FR2b3 FR2c FR2d FR2e FR3a FR3b FR3c FR3d FR3e FR4c	1, 2, 3, 4, 5, 7, 8
9	Programmer with little time to practice	Be matched quickly with someone else to collaborate	I can understand and learn from multiple perspectives without wasting too much time	FR1c FR1d FR1e FR4c1	2, 11

10	Experienced programming student	Find new and challenging problems to solve	I can be confident in my skills for interviews	FR2d FR2f FR2g FR4e FR4e1	14
11	Computing student who practices regularly	Set questions as attempted once I have submitted a solution for it	I don't have to do questions I did already	FR2g FR4e1 FR6a	6
12	Lazy computing student who needs more of a push	Earn points / level up in the platform (gamified)	Feel more motivated to come online and attempt questions	FR6a	14
13	Multilingual computing student whose native language is not English	Easily select preferred language of discussion	Have a higher chance of matching with people who would speak in same language	FR1f	5, 9, 14
14	Regular problem solver	Track my consistency in progress	Need not spend too much effort doing manual tracking	FR2f FR4e FR4e1 FR6a	-
15	New user of a personalized educational website	Create a new account	Personalize my learning	FR5a	1, 10, 12
16	Newly registered user of a personalized educational website	Login to my existing account	Continue my personalized learning	FR5b	1, 10
17	User who wishes to exit the current session	Logout of my active account	My status will no longer be online	FR5c	1
18	User who has unstable internet connection	Reconnect to the same room in case of unexpected disconnections	Continue my progress in the collaboration space	FR3f FR3g	7

2.2 Functional Requirements (FRs)

In this section, we elaborate on the Functional Requirements (FRs) our team has refined from user stories and group them under main general features that serve a similar purpose.

Furthermore, we provide the priority level (high/medium/low) of each FR and the main purposes that they seek to address. Priority level was decided based on a variety of factors: urgency, ease of implementation, satisfying core requirements, and benefit to users, which helps to facilitate project planning and management. Inputs and outputs refer to expected actions and behaviors of the users and the systems which are related to the respective FRs they appear in. Lastly, FRs which have been realized (implemented in our final submission) will appear in **yellow boxes**, while those left for potential future developments are currently unimplemented and appear as **blue boxes**. **Orange boxes** indicate those corresponding to main general features, which actually just consist of the smaller ones.

Management and planning will be described in a separate section (Section #3) later on, which provides justification for why features are included or excluded given our limited resources and desire to ensure project quality. In particular, you might notice that social media features, as well as that of audio-video conferencing, have been prioritized as low as they are generally considered out of our main focus; that is, providing a platform for software engineering interviews collaboration, which at least has a basic channel of text communication.

CF: We reference certain User-Interface (UI) specific components, such as the page names (like the Collaboration Page), in the FRs. To have more context, you can refer to Section #4.3.5 to view the actual Frontend to visualise how these pages will look like in our implementation.

<table>
<tr><td colspan="2">FR1 User Matching:
The application must have a pairing feature for users to match up with another user to solve problems, customization features</td></tr>
<tr><td colspan="2">FR1a: The application has a matching system that pairs users who pick questions of similar difficulty or areas of focus</td></tr>
<tr><td>Priority</td><td>High</td></tr>
<tr><td>Purpose</td><td>Find a good match based on similar specified topics and difficulty level</td></tr>
<tr><td>Input</td><td>Users can set their focus topics and difficulty, then press a button in the Web UI to trigger the matching process</td></tr>
<tr><td>Output</td><td>Users will potentially be successfully matched with another suitable user, and transition to the Collaboration page which contains a Code Editor.
If unsuccessful, inform user that the match is unsuccessful and try to match with another user</td></tr>
<tr><td colspan="2">FR1b: The application UI provides the user a way to pick the exact topic they want to solve a question on (Across data structure and algorithm topics such as Arrays, Recursion and Trees)</td></tr>
</table>

Priority	High
Purpose	Users are able to focus on topics that they would like additional practice on
Input	UI is able to multi-select and apply these filters from the Web UI
Output	Potential matches (with the relevant topics) will be filtered in the back-end
FR1c: The application should match users within 30 seconds of a user triggering the matching process	
Priority	High
Purpose	Limit user waiting time to a reasonable level
Input	User presses a button on the Web UI that triggers the match-making process
Output	Will see a loading screen while back-end searches for a potential match. If successfully matched (found partner and within time limit): Transit to the Collaboration Page
FR1d: The application should give a fail-matching indication within 30 seconds of a user initiating a match if unsuccessful	
Priority	High
Purpose	Limit user waiting time to a reasonable level
Input	User triggered the submission process
Output	After match-making process initiated and timed-out, there will be a UI display that informs the user of the failure
FR1e: The application provides matched users with an option to confirm or reject pairing	
Priority	Low
Purpose	Users should be able to see the profile of their matched pairing and confirm if this is someone who they are agreeable with matching with
Input	Upon completion of user matching process, will display the matched user profile and prompt the matched user to accept or reject the match
Output	If accepted, will go to the Collaboration Page, otherwise user will remain on the same page
FR1f: The application provides a preference option for pairing with users of a spoken language	
Priority	Low
Purpose	Users across the globe may have different native languages and may prefer

	conversing with someone familiar in the same language
Input	Users can indicate preference in their user profile, which will affect their eventual match
Output	User will be matched with another user who is more closely aligned in their language choice
FR1g: The application provides an option for users to be able to attempt questions of **easier difficulty** than what was initially indicated	
Priority	Low
Purpose	Users may want to 'help' other users and will be okay with being paired on an 'easier' question than they preferred
Input	Users can indicate preference prior to start of matching, via multiple difficulty options being selected
Output	User may be matched with a user who is 'weaker'

FR2 Question and Attempt: The application brings users to a question-answering page (Collaboration Page) once users are matched	
FR2a: The Collaboration Page automatically **loads a relevant question** in a question panel once users are matched	
Priority	High
Purpose	To display the question relevant to the paired users, allowing them to know that they have been matched and kickstart the paired question-answering process immediately
Input	Matched users accept the pairing by loading the Collaboration Page
Output	The application now displays the question in a question panel
FR2b: The page contains an **embedded** code editor which is shown once the Collaboration Page loads	
Priority	High
Purpose	To allow paired users to collaborate code-wise, being able to make code changes visible to the other party and view them in attempt to solve the question together
Input	Matched users accept the pairing by confirming the pairing
Output	The application now displays and hosts the code editor in a code editor panel

FR2b1: The code editor allows users to edit the code **near real-time**	
Priority	High
Purpose	To allow paired users to collaborate code-wise almost simultaneously, being able to make changes and view these code changes in the editor, to solve the question
Input	Paired users make changes almost simultaneously
Output	The application's code editor reflects changes by both users
FR2b2: The code editor allows users to select the **programming language** of choice	
Priority	Medium
Purpose	To allow paired users to agree on a programming language
Input	Any of the paired user selects the programming language from a drop-down list
Output	The code editor now recognizes code input in the selected programing language
FR2b3: The code editor offers **syntax highlighting** (in the selected programming language of choice; mainstream programming languages only)	
Priority	Low
Purpose	To highlight programming-language-specific entities to facilitate solving of questions, relieving paired users of strenuous eye-checking on their code
Input	Any of the paired user selects the programming language from a drop-down list
Output	The code editor now highlights the appropriate recognized code structures/syntax in the selected programing language
FR2c: The page contains an embedded communication channel	
Priority	High
Purpose	To contain and display a communication channel (see FR3*), which serves basic means for collaboration between paired users.
Input	Matched users accept the pairing by confirming the pairing
Output	The communication channel/tool now appears for users to communicate
FR2d: The application maintains a question bank of >30 data structures and algorithm questions of varying difficulties that the user can attempt to solve	

Priority	High
Purpose	To provide paired users with questions to attempt and collaborate in
Input	Application administrator enters curated questions, along with their difficulty and topic) into the database for storage and retrieval
Output	Questions are stored with respective information for easy storage and retrieval
FR2e: The application UI provides an option to view the solution, or a link to the solution, of the current question	
Priority	Medium
Purpose	To allow users to understand how to solve the question provided
Input	User select option to view solution
Output	The application presents the solution or an external link in a solution panel
FR2f: The application issues a disclaimer that previously encountered questions have been attempted before	
Priority	Low
Purpose	To inform a user that the question about to be matched was previously encountered and may not be as beneficial as one not encountered before
Input	The system prepares to notify the user of a match (a potential pairing)
Output	Users are provided with a match notification with the disclaimer but not yet paired
FR2g: The application has an option for users to indicate 'finished' and exit after attempting the question to mark it as complete	
Priority	Medium
Purpose	To end the pairing session and mark the question as completed for both paired users
Input	User select option to finish and exit, marking the question as complete
Output	The application recognizes the paired session has ended and marks the question as complete, redirecting users back to the homepage.

FR3 In-session Messaging: The application provides paired users with a communication channel

FR3a: The application provides users ability to send live text messages to each other using the Message Chat / chatbox Note this is different from FR2c as this is about providing an actual means of communication between paired users rather than containing the communication channel	
Priority	High
Purpose	To provide a basic means of communication between the paired users
Input	Users type in messages in the text field and enter these messages into the chatbox
Output	The chatbox displays the messages send by both users
FR3b: The application provides users ability to get into an audio call with each other using Audio Chat	
Priority	Low
Purpose	To provide a more efficient mode of communication than text to facilitate easier collaboration among users on coding implementations
Input	Users *unmute themselves* and speak to the microphone of their PC
Output	The application provides audio output of the user speaking to the matched peer through his/her connected speaker or earphones.
FR3c: The application provides users in an audio call the option to mute and unmute, showing the muted/unmuted status.	
Priority	Low
Purpose	To provide users the control to disable or enable sharing their voice or other live audio whenever they want during a call, while also showing this enabled/disabled status
Input	User trigger option to *mute/unmute*
Output	The application will no longer transmit the voice of the user to the matched peer The application displays a muted icon to the user that clicked mute The application will indicate to the matched peer that the other peer is muted with a mute icon on the peer card
FR3d: The application provides users with a video chat feature	
Priority	Low
Purpose	To provide a way for the users to see each other when discussing the solution

Input	User selects the option to use video chat
Output	The application shows the live video feed of the paired user(s) who enabled the video option
FR3e: The application provides users with an option to turn video on or off during video chat	
Priority	Low
Purpose	To provide users with an option to disable or enable sharing their live video/voice to the application while using the video chat
Input	User selects the option to turn the video off
Output	The application shows a placeholder image in-lieu of missing video feed
FR3f: If a user disconnects (e.g. closes tab accidentally), the Collaboration Page remains open as long as the other user is still at the page	
Priority	Medium
Purpose	To improve the overall user experience and ensures that their PeerPrep session is reliable, and to allow rejoining if one of the users disconnect
Input	Either user disconnects
Output	Collaboration Page is still accessible, and will be closed if detected that both users are off the page
FR3g: If either user disconnect and successfully reconnected, is able to see the code that was worked on so far	
Priority	High
Purpose	To allow users to maintain their progress and implementation in the code editor without having to start from scratch if they unfortunately disconnect
Input	User navigates away from the Collaboration page (without indicating 'finished')
Output	Can still re-enter the Collaboration page either through re-entering the web app route through the same unique url, or by clicking a button from his/her own page

FR4 User Profile Interaction: The application has social networking options to interact with other users	
FR4a: The application provides users with an easily identifiable profile, recognized by a unique username	
Priority	High

Purpose	To provide the users information about their account Useful if another user wants to identify who they are matching with, or if they want to match with a particular user
Input	The user is able to see their own profile information, as well as that of others once matched
Output	The application displays their profile-relevant information about the user (e.g. Username) as well as matched users
FR4b: The application offers users ability to search and navigate to other users' profile pages using their username	
Priority	Low
Purpose	To provide users to search for another user's profile
Input	The user enters the username of the user they are looking for
Output	The application shows the profile of the searched user
FR4c: The user profile page provides the user the option to pair up directly with users of choice from their profile (On Demand, Manual Matching)	
Priority	Low
Purpose	To provide users a way to select a peer of their choice to practice with
Input	The user finds the profile of the user they want to practice with and uses the option to match with the user if he/she is also online
Output	The application notifies the user who receives the matching request and navigates both users to the Collaboration page if the request is accepted. If the request to match is rejected, the requester will be notified.
FR4c1: The application notifies **online** invited users once someone sends them an invite	
Priority	Low
Purpose	To inform users that another user is attempting to match with them
Input	Current user needs to be online and signed in Another user will send an invite to current user from the profile page of current user
Output	Application will provide users with options to accept or reject the invite. If accepted, a match will be created and a room will be made for the 2 users. Otherwise, user who sent the invite will be informed the requested user is currently not able to match

FR4d: The profile page shows the online status of a user	
Priority	Low
Purpose	To provide a soft-signal to other users who are browsing current user's profile and want to send an invitation to match with the user
Input	User must be signed in and has an open browser tab with the application
Output	An indication in the user profile's UI to depict whether they are online, offline, or currently in another session
FR4e: The user profile tracks past questions attempted by the user	
Priority	Low
Purpose	To allow user to monitor progress in terms of questions completed
Input	User marks a question as complete and goes to their profile page to view the list
Output	The user profile shows a list of previously attempted questions and dates of attempts
FR4e1: The user profile shows a heatmap of dates of attempts	
Priority	Low
Purpose	To track the user's activity, thereby encouraging them to maintain a streak of question attempts
Input	User marks a question as complete will mean user is active
Output	The UI shows the days in which a user is active in attempting questions, through a coloured heatmap (similar to LeetCode/GitHub)

FR5 Accounts Management: The application allows visitors to create accounts and login for a personalized experience	
FR5a: The application provides visitors with the functionality to create new accounts	
Priority	High
Purpose	To allow creation of a new account
Input	Users enter account details for creating their account and for future logins
Output	Users are redirected into the website to indicate successful account creation and thus login, or prompted to re-enter information

FR5b: The application allows existing users to login into the website for a personalized experience	
Priority	High
Purpose	To allow existing users to login.
Input	Users enter account details for login.
Output	Users are redirected into the website to indicate successful account login, or prompted to re-enter information
FR5c: The application allows logged-in users to logout off the website	
Priority	High
Purpose	To give users a persisted login session
Input	Users selects the option to log out
Output	Users are redirected to the login page

FR6 Gamified Experience: Application provides users with a gamified experience with potential point-based rewards	
FR6a: Application provides users with points whenever a problem is marked as complete, and the users can earn badges for every 100 points received	
Priority	Low
Purpose	To provide users with an additional incentive to complete more questions and hence use the application more
Input	Users mark questions as completed once they are satisfied with the solution they have produced
Output	Users receive points and/or badges and receive notifications for them They can also view their points/badges in their profile page

2.3 Non-functional Requirements (NFRs)

2.3.1 List of NFRs

This section addresses the constraints and optimization criteria of our web application, together with their associated quality attributes.

#	Requirement	Quality Attribute
1	The system should have at least 99% uptime	Availability
2	Request and response handling between client & server should be at most 1 second	Performance
3	The application should be accessible by all modern mainstream internet browsers (Chrome, Safari, Firefox)	Portability
4	The application must support users on Linux, Mac, and Windows operating systems	Portability
5	The application must support interaction between cross-platform users **on web browsers**	Interoperability
6	The **code editor** shared by matched users must work reliably when being edited by both peers simultaneously (i.e. one user's edit should not undo another's)	Usability Reliability
7	The application must reflect the changes to the code editor to the peers with very minimal latency (< 1 second)	Performance
8	The voice chats must not be stored beyond the session to protect the privacy of the users' conversations	Security Safety
9	The application should have **a small number of clicks** (within 5 clicks) to access important features (e.g. looking for a coding friend, starting a coding question, opening user profile etc.)	Usability
10	The sensitive information of the user profiles (eg: password, address etc.) must be encrypted	Security
11	The application must be able to support at least 1000 concurrent users without major degradation to performance	Scalability Performance
12	The application must be able to serve at least 5000 unique users without major degradation to performance	Scalability Performance
13	The application must have high test coverage (>75%) and	Integrity

	include both unit tests as well as tests to ensure components interactions work as required	Verifiability
14	The system implementation must be modular and declarative so that it can support new features with ease	Modifiability

2.3.2 Prioritization for Quality attributes

Our team decided to adopt a $100 approach to prioritize NFRs. Given $100 to decide on how much to invest in each quality of the NFRs, we have allocated funds proportionally to their importance and criticality. While web applications often have a trend for certain qualities such as high availability and interoperability, the distribution taken by our team reflects one in which we consider:

1. How much resources we have to dedicate on top of already provided technology
2. Reflect the needs of the users most
3. Market standards (*LeetCode*, *AlgoExpert* etc.)

Quality Attribute	**Cost ($)**	**Rationale:**
Usability	15	Target users come from a generally younger demographic and are hence used to applications with intuitive user interfaces. Therefore, the application should have a focus on usability for it to appeal to users
Performance	15	The application must perform fast to serve the best user experience - lag will make the learning experience extremely slow
Scalability	13	The application architecture should be able to scale by creating more service instances to handle a higher user load
Reliability	11	The application must be able to handle all user operations as intended and correctly, especially for user-to-user communication
Integrity	11	The application should be able to handle a variety of behaviors, including actions that would otherwise lead to errors/exceptions, and ensure the application does not fail/break during usage. This is especially important since the application will handle many instances of free form input
Modifiability	10	Since this is an early stage application, it should be easy to add new features or polish current features
Availability	9	Target users are students who have varying timetables and

		sleeping schedules so the application should be generally available 24/7
Verifiability	9	The application should seek out to satisfy requirements it aims for and stay correct and efficient. However, as this is an early-stage application, we prioritize coming up with functional features first while maintaining relative ease of testing
Portability*	2	As the application is a web app, it is already considered to be portable since users with different operating systems can still use the application via a standard web browser (e.g. Chrome). Therefore, we can achieve portability without much extra resources
Interoperability*	2	Web apps are mostly highly interoperable, as all data requests and responses follow standard REST API protocol. Therefore, we can achieve interoperability without much extra resources
Security	2	Not much sensitive user data apart from emails/username
Safety	1	The software only contains minimal information about the user and has minimal possibility of harming the user in any way. Hence, this is taken for granted

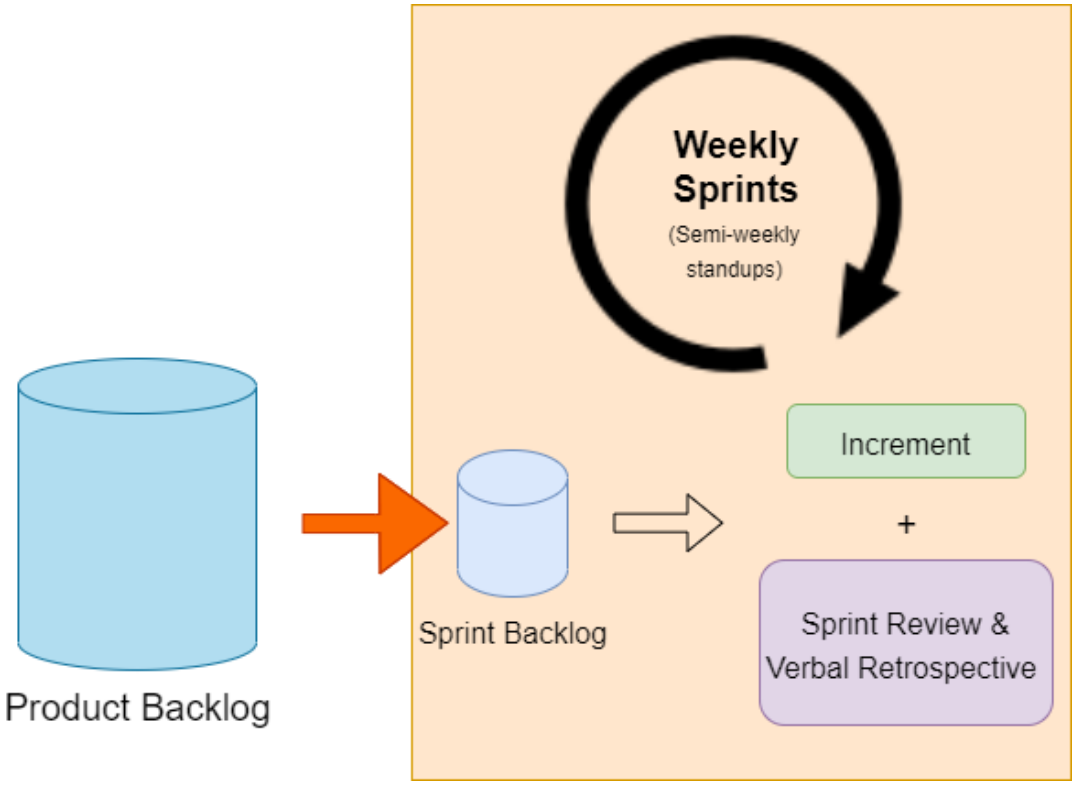
3. Software Development Process and Management

3.1 Software Development Process

For most software projects, it would be beneficial to adopt a systematic approach towards the development of the product, hopefully one that is both suitable for the team and allows for the greatest productivity. Not only does this cut down on communication overhead and the time needed to come up with standardized processes, but it also provides a template to which the team can adhere to on a regular basis -- thereby allowing developers to channel their energy towards actual development rather than trying to fit the mould into an unfamiliar environment.

Agile: Scrum Framework

Our team recognizes these facts, and have decided to adopt the agile scrum framework for our software development life cycle. This is mainly because we are a small team who wishes to move fast, while also building horizontally (breadth-first) to allow work in parallel so that we can develop both incrementally and quickly. Moreover, this also allows for earlier detection of any unexpected bugs, blockers, and issues, which facilitates communication as soon as they arise. Throughout the project, the team has therefore learned to appreciate the benefits that scrum brings -- better communication, smoother development flows, earlier detection of bugs, and more importantly happier developers who are able to tackle issues productively as they come.



Overview of Sprint: Steps undertaken

As per the typical software project, in each sprint, our team has engaged in the phases of Requirements Analysis, Design, Implementation, Testing, Deployment, and Maintenance.

In requirements analysis, having a broad understanding of how the system behaves and already having crafted a product backlog, we choose related and relevant tickets and place them into the sprint backlog. In the Design phase, we initially refer to that in which we have previously proposed, and then work to refine and improve upon it in terms of architecture, component, and API design to allow everyone to have a general understanding of the system. This also facilitates horizontal breadth-first development as we apply concepts of information hiding and abstraction to use components which the rest of the team is developing. Moving on, the implementation phase involves the team writing clean and understandable code adhering to the agreed upon designs, to which is then tested in the Testing Phase where basic unit tests cover all cases within each component. We then perform further tests to ensure components interact correctly with one another, before deploying either locally or to the cloud. Lastly, the maintenance phase involves us fixing any post-deployment bugs found, as well as potential addition of new features. We also incrementally write documentation for our implemented features.

Benefits: Improved Communication

For the development process, our team chose to follow an agile scrum framework which entailed weekly sprints and semi-weekly standups. During the standups, we communicate any blockers, issues, or unexpected bugs which may be hindering our development so the team is updated and can adjust the workload accordingly. This also aids in project management where we learn the competencies and preferences of each team member. In addition, keeping in line with the spirit of agile, whenever we needed any additional clarification or meetings, we also held them outside of the semi-weekly team stand-ups where we checked on progress of each member. Here, we hoped to be able to produce small but reliable increments each sprint, and held informal sprint reviews and a verbal retrospective towards the end of the sprint, which we then began on sprint planning for the next sprint.

Benefits: Requirements Sharpened

As for the roles, in each sprint, our team members took turns playing as product owner, scrum master, and the development team, allowing us to appreciate and experience the different angles to which each role approaches a situation. Moreover, it also allowed us to understand the corresponding different perspectives, thereby allowing the entire team to uncover requirements which we did not find before, which may be attributable to the concept of groupthink.Through this practice we adopted, we also realized that the features which we initially thought of as already 'good-enough' could have been improved further, leading to our eventual focus on delivering a good user experience instead of having many small but less beneficial features. Therefore, we also realized the fact that the scrum framework was in fact

particularly applicable in our case, as requirements can potentially change at each step of development.

Testing

To ensure that the code written by the team is correct and up to specification, we write unit tests for each major service provided by the components, where applicable. Furthermore, the team has also performed manual testing between components / microservices to ensure that their interactions work as required. Tools used include automated testing using Mocha/Chai framework(s), and Integration Testing via Postman, among others. Additionally, we made use of GitHub Actions for Continuous Integration, which automatically runs our unit tests defined when we raise pull requests.

3.2 Notable Workflow and Product Management Practices

As per mentioned before, the team has chosen to adopt good software engineering practices in our work. This section addresses other complementary practices which we have learned in the module and adopted, as well as other basic ones we find applicable and prefer to use.

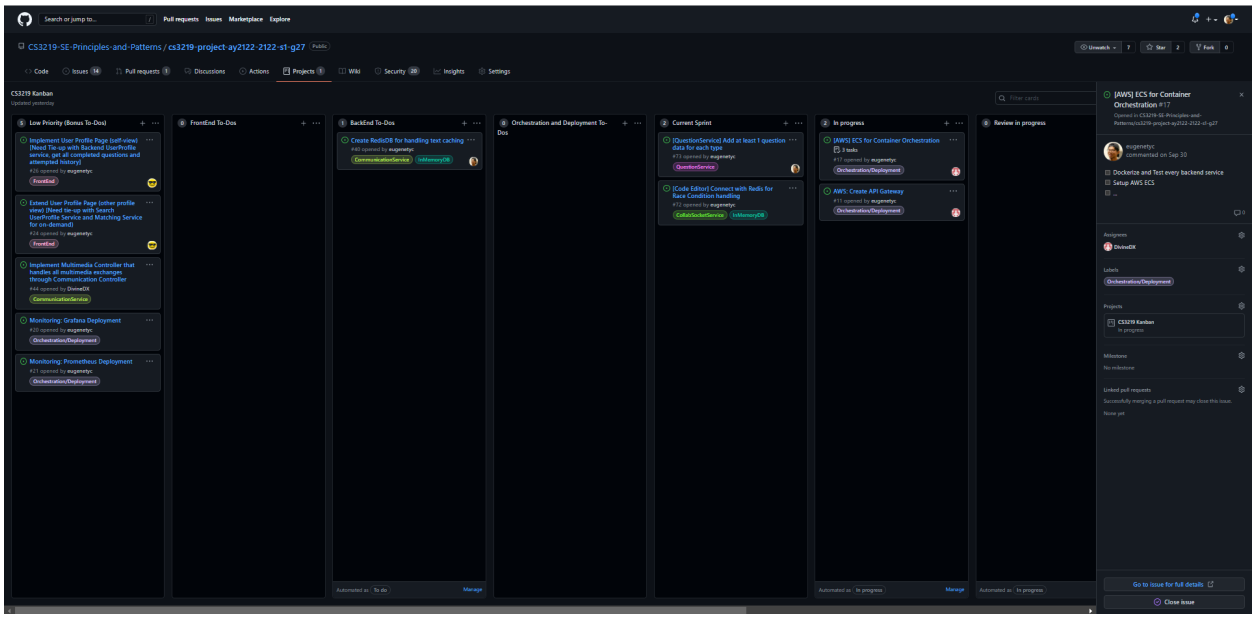
Use of a Kanban Board to help chart Sprint Planning

The team uses a Kanban Board with customized headers suited for our preferences. We find that this helps us easily view our tasks and facilitates their estimation for completion. Here, tickets fill up the Kanban Board each under a single header, and each ticket is tagged with the relevant tags (e.g. CommunicationService, Orchestration/Deployment). In particular, we have the following board headers:

1. Low-Priority (Bonus To-Dos) [you can refer to the section on Project Management for this]
2. FrontEnd To-Dos
3. BackEnd To-Dos
4. Orchestration and Deployment To-Dos
5. Current Sprint
6. In progress
7. Review in progress
8. Reviewer approved
9. Done

Note that headers 1 to 4 form our *Product Backlog*, header 5 is the *Sprint Backlog*, headers 6-8 are for the *current sprint*, and once at the end of each sprint when we have all agreed that the current sprint ticket is completed, we shift it to the 'Done' header which is essentially like a queue, with tickets at the top being the most recently completed. All completed tickets are stored in 'Done' in case we would like to review them; we therefore do not delete any tickets after completion for logging purposes.

Below is a screenshot of the Kanban Board we used, under Github Projects. As the Board is rather big, we have used a zoomed-out version of it. You can view the actual board in our Github repository link.



Separate Branches to work on code, PRs to master

The team has agreed upon the simple yet effective git workflow where a new branch is created for each new feature we seek to develop, and only upon completion of development work (or when we perceive it to be completed), do we create a pull request (PR) to the master branch. This helps to prevent clogging of PRs which make the codebase messy and intimidating to reviewers.

PRs require code reviews before merging

The team takes to heart the importance of good and clean code. To this end, all PRs require that another team member **must** approve of the changes before merging. For our team, those checking have to primarily check for 3 things:

- Coding standards are consistent
- The code adheres to the agreed upon designs
- Can it be done better? What are some alternatives

In particular, the last point often leaves comments for the PR-raiser to reflect upon and improve the code. This leads to what the team calls 'Engineering Excellence', where we consistently seek to remove technical debt from the start and guide each other on best practices that we know but others may not. This is therefore a good practice that not only ensures our codebase is working, good and clean, but also one that encourages peer learning, just like PeerPrep.

Continuous Integration with GitHub Actions

As taught in CS3219, the team understood the benefits of automation and found in particular, continuous integration suited our needs. This is since we considered deployment to be rather complex and would prefer to do it manually, while we generally understood the workings of git and pull requests and found it comfortable to automate the process of integrating new code into the existing database. To this end, we have used continuous integration with our services, harnessing its power to perform checks and prevent software regression.

Coding Styles and Standards

To enforce a uniform coding style and standard between all developers, we made use of some developers tools that will enforce a certain clean coding standard. For a start, we standardized the use of Visual Studio Code as our IDE. Then, we added Prettier, a code formatter that is able to automatically format the code for us according to the rules we defined in a configuration file, such as the indentation sizes. Lastly, we added ESLint, a code linting tool that will be able to highlight code smells and potential code errors. This greatly improved our engineering quality and lead to a smoother workflow.

3.3 Project Management

Everything above in section #3 then leads one to question, "Okay, so you seem to have a solid development plan, and it is one that is flexible and allows for change as per user requirements changing. You have also used state of the art tools in ensuring that your codebase is working and any potential changes introduced are not breaking the current code before merging. But how will you manage the complexity? How can you best manage your resources while minimizing compromises on quality?". Naturally, this leads to our subsection on Project Management.

As mentioned in previous sections, our team is rather ambitious, laying out a full ground plan for the future of the application we are developing. This meant that we had plans that span far beyond the semester of the CS3219 module, should potential developers decide to continue on it. This certainly sparked the entrepreneurial spirit of our development team, inspiring and motivating us to write clean and productive code.

However, ambition is like a cup, and just how far we can go in filling the cup with productivity is what leads to our actual final progress of the project. While it is good to shoot for the moon and land among the stars, we need to also be realistic and keep to what's possible. To this end, we recognized that we had limited time and resources, and practiced judgement on which features ought to be prioritized first to produce an application that met our minimal expectations, before moving onto additional features that provide icing on the cake.

With reference to the section on Software Requirements Specification #2 (in particular #2.2 and #2.3), we weighed both the difficulty and complexity of producing certain features against what was needed for a basic application and what we thought would bring about the best user experience. More importantly, we also had to factor in the notion of limited resources as per time, energy, and mental capacity of developers to learn new technology stacks, against the quality of the final product we want to deliver. This led to some initially planned features not being implemented in our final submission as they are deemed less important. However, we do hope that the teaching team can understand our priority weighing rationale as this is also part of Software Engineering practices in the industry.

The above decisions eventually led to features like audio and video chat being pushed to future developments, while we managed to produce basic features like text chat. Although this was a process that was occurring throughout the entire project, we preferred to represent these in the table below instead of a timeline as the table allows readers to more clearly understand the rationale for prioritizing some features over others. Also note that the team has managed to achieve all FRs that are medium priority and above (after adjustments where we revalued FRs as we discovered more about them), so any features being less prioritized correspond to the FRs which are of low priority.

Features Prioritized	**Features Less Prioritized**	**Rationale**

User Matching based on selected topics and difficulty	Option to confirm or reject match pairing (FR1e)	Users click match because they want to match, so it is less likely users need to have an option to reject the pairing
User Matching based on selected topics and difficulty	Option to match with users who speak a particular language (FR1f)	English is a general universally-understood language. Users can simply liaise among themselves if they can and want to speak another language once matched. Also, users can simply un-match by indicating finished and search for another match if they are unable to communicate with their current partner
Ability to collaborate with users on questions	Providing a disclaimer for questions that have already been done before by the current user	It does not hurt users to redo a question previously attempted; these questions may have been done long ago and they might have forgotten. Moreover, users who wish to focus on the communication aspect may find questions attempted before more beneficial for learning. From a product owner point of view, allowing users to do questions previously attempted also increases the matching rate of users, so other users need not wait as long.
Providing a text chat functionality for collaboration	Providing audio and video chat functionality for collaboration (FRs 3b, 3c, 3d, 3e)	Users simply require a means to communicate. While it is true that audio and video may be useful as well in facilitating smoother communication, these features may not even be used by users as they are often communicating with strangers on the internet. Audio and video is also more technically challenging, and the team is relatively new to the ideas of sockets for text and more so the related technologies for audio and video. Lastly, audio and video transmission may also require more expensive servers and services which we are unable to afford given the project's budget. Therefore, based on the last point alone, this is infeasible if we were to provide secure channels for communicating via audio and video as we require some form of ownership.
Ability to collaborate with users on questions	Having social media network features (FRs 4b, 4c, 4c1, 4d, 4e, 4e1)	Our application's primary objective is to provide users with opportunities to practice both algorithm questions and communication. To this end, the focus of having a social media network is well beyond the scope of our primary objective, which already requires significant amounts of effort in learning entirely new techstacks and concepts. Moreover, users are still able to match and collaborate on questions, and the profile-specific options and details are more towards an icing on the cake. We would recommend future developers

		to take this challenge up as our version of PeerPrep becomes more commercially viable.
Ability to collaborate with users on questions	Gamified Experience (FR 6a)	Gamification ties in closely with personalization and profiles as mentioned in the row right above this. Therefore, gamification is more of a personal incentive for using our platform, and therefore something more of a commercialization step for the application to launch in the future.

4. Overall System

4.1 Overview of Software Architecture

PeerPrep features several complex features, which, along with our non-functional requirements, dictates the architecture that we will be adopting for this project. While our expected deliverables are not extremely large, one can easily imagine PeerPrep extending and morphing into a much larger and complex application with many new features. Therefore, we decided to adopt a service-oriented architecture, specifically the hexagonal architecture where the different parts of our application are split into microservices.

The driving reason for our choice is scalability and performance, while also seeking to maintain high usability. PeerPrep should be built to handle high user loads, and this usage is not trivial like sending and receiving API Requests. A large part of its functionality is handling real-time communication text chat, as well as live code collaboration which is enabled by WebSockets. The load on a server is expected to be high, and services must scale to be able to meet the demand. Moreover, the identified services seek to address user needs and requirements. Adopting a monolithic architecture — where our entire back-end is packaged together — may not be as sensible as the load is not equal across every service. The services that are expecting higher loads can be deployed with a greater number of running instances.

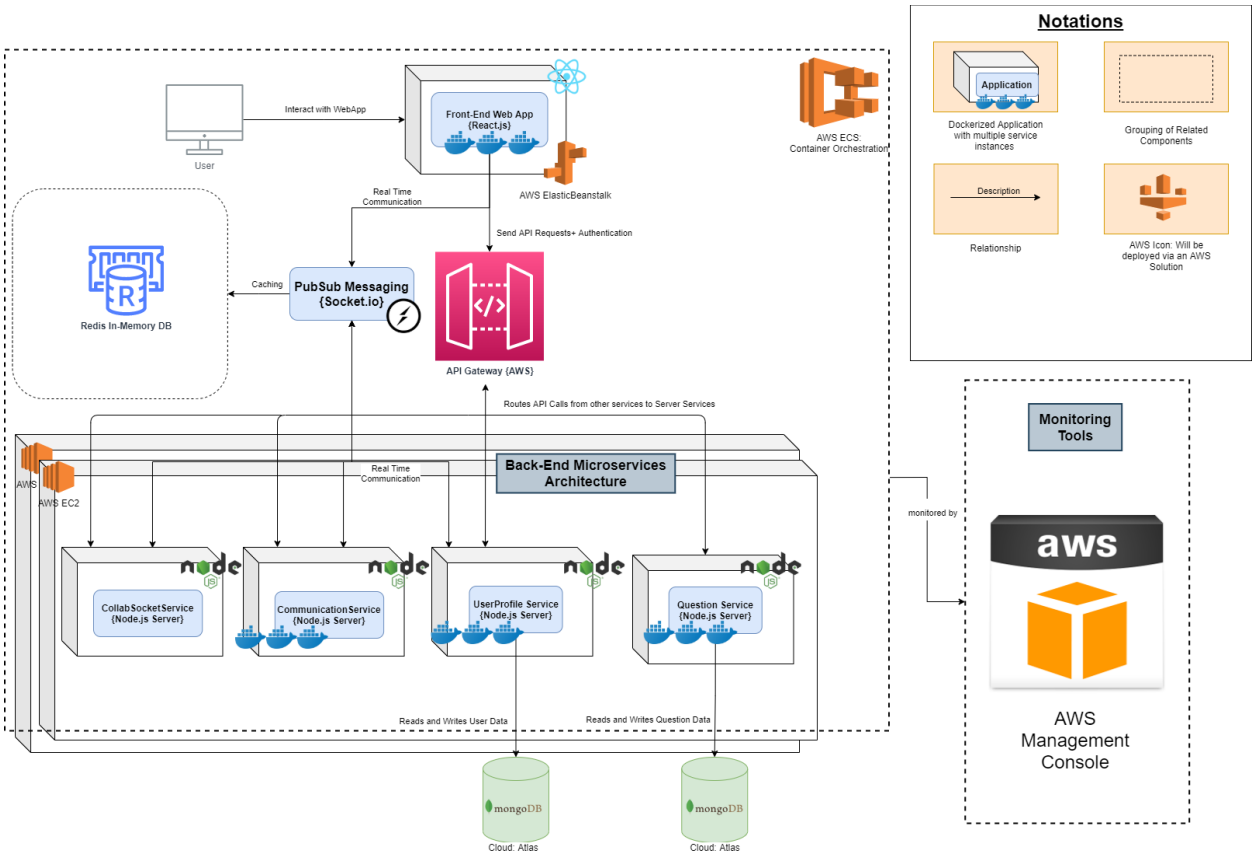
Additionally, adoption of a hexagonal architecture would make our application highly modular. It would make our application easier to test in isolation with the use of dependency injection, which is closely related to the dependency inversion principle that is the core of the hexagonal architecture philosophy. The ‘ports’ and ‘adapters’ are replaceable; for an evolving application, this is great because we can always swap out a service for something similar quite easily when we require a functionality change.

Our entire application can be expected to be deployed and orchestrated on the cloud. The microservices approach is also great for creating cloud applications because cloud providers like AWS have a massive range of solutions to fit our individual needs for each service.

Below, we will first introduce our architecture diagrams, design, a subsection on interactions between each service and frontend so that we have some semblance of a service-responsibility-collaborator model (adapted from Class Responsibility Collaborators model), and lastly what techstack we used.

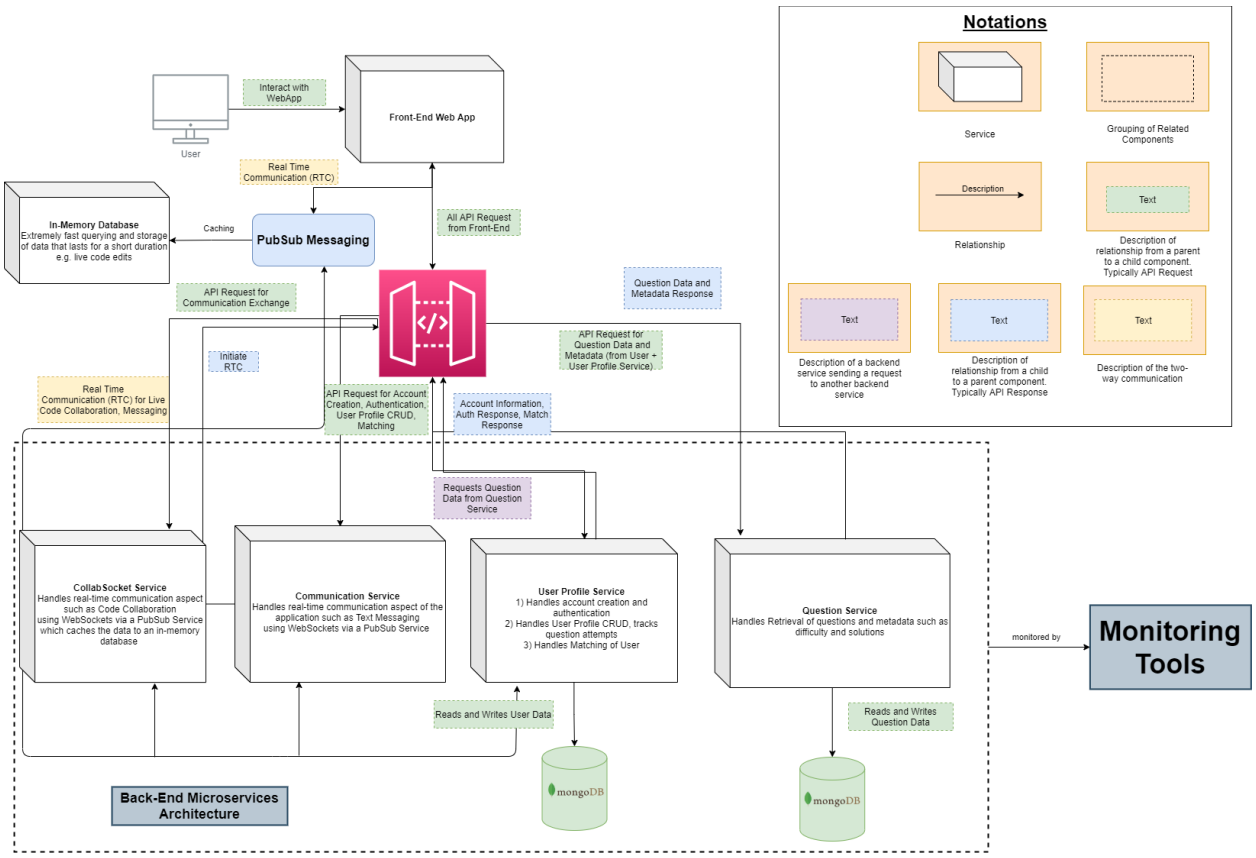
4.2 Architecture Diagrams

4.2.1 Overall Architecture Diagram



The above shows our overall software architecture, most notably displaying the relationship between our different services in a microservices architecture as well as our tech stack used under each service, along with their cloud deployment tool (typically an AWS Service). All of the cloud-deployed services are orchestrated using AWS ECS, and each service will have multiple containers to support high reliability and uptime.

4.2.2 System Context Diagram



Under our system context diagram, we removed the implementation details such as the technology/service used, in lieu of the hexagonal architecture where we have decided to adopt, where services are *adapters* that can be easily swapped out for something else that follows the same interface. Instead, this diagram mainly displays the data flow such as API Request and Responses between the different parts of the application.

4.3 Container-level Services

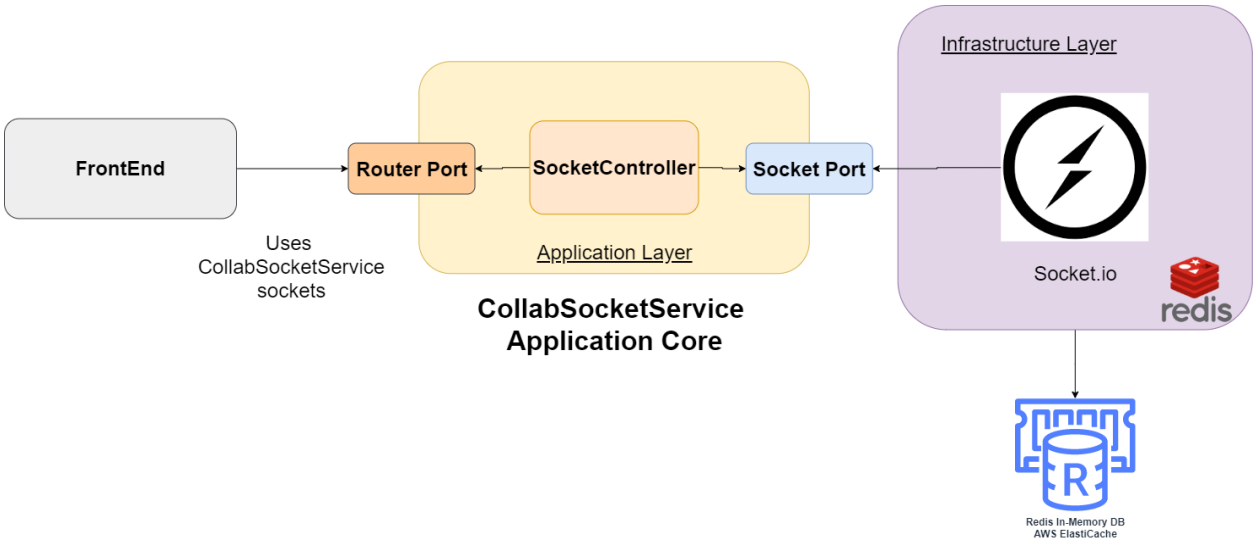
This section describes the main components of the entire system, as well as shows the component-level interactions within each service (single component) and between services.

There are 5 main subsections (namely for CollabSocketService, CommunicationService, QuestionService, UserProfileService, and the Frontend Web App), each describing (where applicable) the overview, design, application programming interface (API) endpoints, and design decisions involved for the relevant service. Do note that external services (such as Redix, Socket.io, and MongoDB) do not have subsections and we will simply use them in our implementation. Moreover, a few services do have shared technology stacks and tools used, so we shall avoid repeating the explanations once they have been described before.

Most of our microservices have API/Websocket endpoints, and we have unified their routes to make it highly easy for an API Gateway to route requests to the correct service. For instance, all API endpoints under UserProfileService starts with */api/user/...*

Please refer to subsections 4.3.1 onwards for the services we have developed.

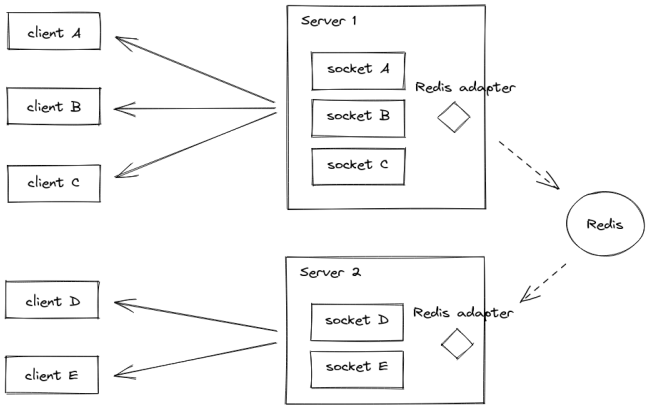
4.3.1 CollabSocketService



Overview

The CollabSocketService is the service that enables near real-time code-editing in the collaboration page. It handles two pieces of information per pair of matched users: the code in the code-editor and the programming language of choice of the pair. As seen in the above diagram, we have sought to apply the hexagonal architecture by having a central controller that handles all communication between the 2 services. While less apparent here, we have sought to use the hexagonal architecture throughout all our backend services. For instance, the QuestionService is a good example of demonstrating the architecture.

The CollabSocketService utilizes a Socket server from Socket.io and Redis caching to achieve it's desired functionality. More notably, a Redis adapter for Socket.io was also used to allow for communications between clients who have connected to different server instances. For instance, the image below from the official Socket.io website clearly describes this. Therefore, this aids in allowing our system to be scalable facilitating our Microservices approach.



(Image Credit: https://socket.io/docs/v4/redis-adapter/)

Sockets Interaction

Events listened	Interaction
room	To join room
disconnecting	To leave and delete the room
coding event	To effect changes in the code editor to the room and its users
lang event	To effect changes in the programming language selected to the room and its users
finish	To leave the room and disconnect

Technology Design Decisions:

Socket.io:

HTTP requests would not be suitable for real-time collaborative code editing due to the slow speeds. Moreover, for such collaboration, we would need a communication channel where both the client and server can initiate communication, which is not possible with Rest API calls. To achieve FR2b1 successfully, faster socket communication is necessary.

Socket.io provides excellent abstraction for socket communication with code that is very easy to learn and implement. It provides powerful features like a ‘room’ that are directly relevant for our collaboration page. Hence, we utilized Socket.io to enable the real-time communication of the code and programming language.

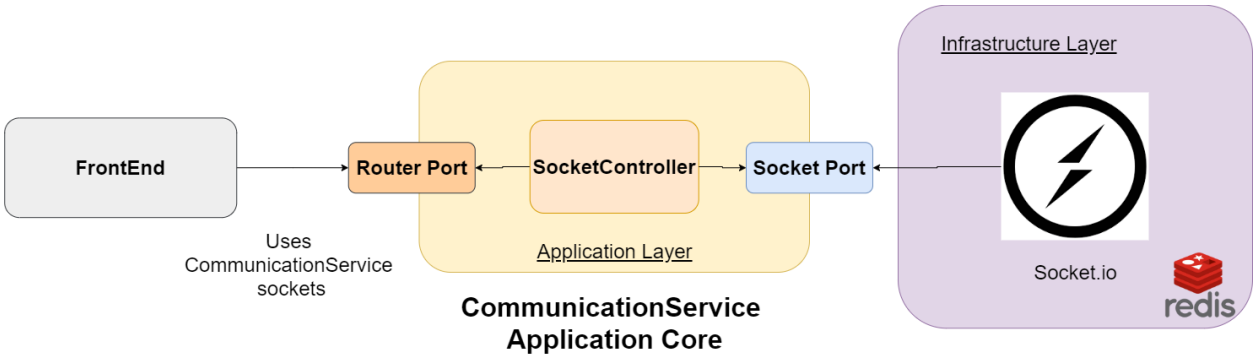
The CollabSocketService defines all the socket events it will serve for sockets within a particular room. The UI Collaboration page would then implement a socket client that connects with the CollabSocketService with roomId from QuestionService and both emits and listens to events from it.

Redis:

We used Redis in-memory storage to keep a single source of truth for the code and language per room of matched users. This is because of the very high read and write speeds that Redis offers, which are very apt for the low latency communication we want to serve. A code or language change from any of the users will cache the data to the in-memory Redis DB before broadcasting.

The CollabSocketService also interacts with QuestionService. When *both* the users leave their matched room, the CollabSocketService sends an API call to the QuestionService to destroy the room.

4.3.2 CommunicationService



Overview

The communications service's main responsibility is to facilitate Real Time Communication (RTC) between matched users on the Collaboration Page. To achieve RTC, we use Socket.io for the reasons mentioned in 4.3.1 (Technology Design Decisions). Similarly to CollabSocketService, a Redis adapter for Socket.io is also used to allow for communications between clients who have connected to different server instances.

CommunicationService is actually very similar to CollabSocketService. However, unlike the CollabSocketService, we decided that there is no need for a Redis store for chat messages since they are differentiated by user, unlike the code. Moreover, when one user disconnects and joins back, it is not as crucial to see the chat history as it is to see the common code, as long as they can continue to chat properly.

Also, while CommunicationService uses Socket.io with room information very similar to CollabSocketService, due to its not needing Redis and for managing a better load, we split communications into its own service.

Sockets Interaction

Events listened	**Interaction**
room	To join room
interact	To send messages into the room and its users

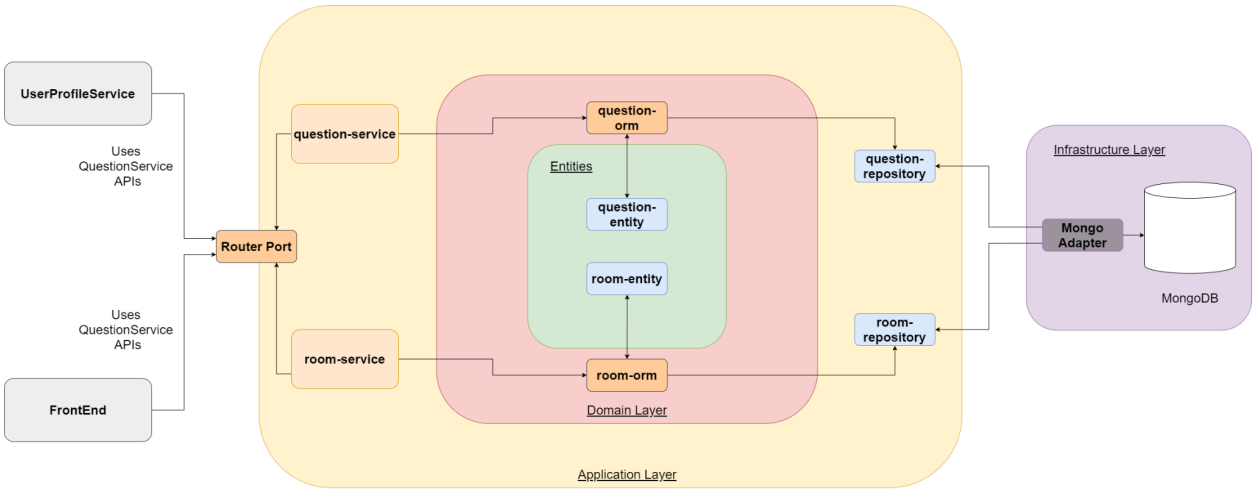
Design Decisions

Design Decision 1: Should we produce a separate CommunicationService, or place its functionalities within CollabSocketService?

Criteria	**Separate Service**	**Combined Service**
Separation of Concerns, cohesion, and coupling	Having a separate service allows better separation of concerns, since each service does a specific purpose. This makes code more cohesive as related code is grouped together. Moreover, CommunicationService is less crucial and the team has decided not to use a Redis cache for this. This reduces potential coupling between the cache and the separated service.	Combining services means that a service now fulfills more purposes, making it more general. This also makes it less cohesive.
Performance	Due to microservices architecture, having separately deployed services allows for better load balancing. However, this may require more connections to separate services.	Having a combined service means that fewer connections may be needed for clients to access services, which may be good for performance.

Final Decision for Design Decision 1: We decided to go with CommunicationService as a separate service, due to its better potential for load balancing and the fact that we would like to keep our concerns separated. Moreover, although certain client functions may require more connections to separate services, we find that the load balancing aspect is more prioritized over this, since text chat messages and code editing both potentially contain many read and write-heavy operations, which we would like to optimize.

4.3.3: QuestionService



QuestionService Application Core

The QuestionService plays a core component in the retrieval of questions and their solutions, as well as the creation of question-specific collaboration rooms for matched users. Here, Rooms that contain information on matched users, as well as their questions of choice, are stored and made available for retrieval by other components/services, namely the UserProfileService and the FrontEnd. API endpoints exposed by the QuestionService are available later in this subsection.

Overview

As per the other services, the QuestionService adopts the Hexagonal Architecture, seen in the diagram above. Components like the UserProfileService access the QuestionService component via a router, which exposes the APIs and specifies what each application layer service should do. Here, we have 2 application layer services: question-service and room-service. The question-service handles everything related to algorithmic coding questions, while the room-service builds on the question-service as well, but primarily focuses on room-related operations (for brevity and simplicity, the above diagram only shows room-orm subcomponent interacting with room-entity and not question-entity). To modify the relevant objects, they make use of the domain services question-orm and room-orm, which use question-repository and room-repository. The respective question-repository and room-repository ports specify operations that driven adapters must implement, which is primarily the Mongo Adapter that points to a MongoDB

APIs provided by the service

API	Purpose

GET /questions/id/:id	Gets a specific question by its question id
GET /questions/all	Gets all questions stored in database
GET /questions/metadata	Gets question metadata for selecting questions
POST /room	Creates a room given the relevant request body
GET /room/:roomId	Gets a room's question given the specified room id
DELETE /room/:roomId	Deletes a room given the specified room id
GET /room/username/:username	Gets the latest room that the specified username is assigned to

For more lower-level details, you can simply refer to the codebase, to which all backend services adopt a similar file structure.

Design Decisions

While coming up with the design for the QuestionService component, we faced a number of decisions we had to make before implementation. Among these include those shown below.

Design Decision 1: Should we combine everything into 1 application layer service?

Criteria for Comparison	**Split into question-service and room-service**	**Package everything into a single application layer service**
Separation of Concerns, Cohesion	Splitting into multiple application layer services allows for better separation of concerns as each application layer service now seeks to perform operations more closely related to its different, specific purpose. Code becomes more cohesive since methods which are more related are now grouped together instead of with other less related ones.	There is a poorer separation of concerns as a single application layer service has now more than one main focus / responsibility. Code becomes less cohesive because methods which are less related are grouped together in 1 bigger service.
Single Responsibility Principle	Each service class adheres well to the Single Responsibility Principle because we now only need to change code in the service files which involve the bug.	There is weaker adherence to the Single Responsibility Principle because the service class is now larger due to code carrying different responsibilities. In this aspect, there are more reasons to change the class

		should a debug be detected in any of its many methods.

Final Decision for Design Decision 1: It is clear that splitting into 2 application layer services is the better approach as it not only provides better modularity where there is higher cohesion (since related code is packaged together), but also allows easier debugging since we can be sure that code which has caused a bug is likely to be in one of the two files, which has a smaller amount of code to review through.

Design Decision 2: What should we use to ensure better uniqueness of each collaboration room's id? The team had initially considered a hash or just a simple username concatenation, and decided to go with a hash. However, among hashing, many factors can come into play to come up with a difficult to crack hash (that is, to find collisions). This is important to prevent unwanted users from entering the same room once they manage to obtain this room id.

Criteria for Comparison	**Use hash functions which have changing output hash values given the same input**	**Use hash functions (like SHA-256) which consistently give the same output hash given the same input**
Security	In general, this is more secure as hash functions are generally already difficult to crack and thus some time is needed to find the hash for a room. If a hash function has different hash values for the same input at different times, then all the work done by a potential 3rd party thus far becomes wasted, thereby reducing the likelihood of finding collisions. This implies fewer uninvited guests into rooms.	This is less secure because once a 3rd party or hacker knows the 2 usernames and additional inputs (like salts), he is able to get the room id since the hash function consistently gives the same output for the same inputs. This means a higher likelihood of uninvited guests.
Issues when allowing users to reenter the room (e.g. accidental disconnects)	This requires storage of the initial hash value computed somewhere, with respect to the user's username. However, if there is an attack on our application or servers, say DDOS, then this can potentially generate many different hashes for the same 2 usernames, which can either exceed existing storage space, or overload our compute power with the additional work	Similarly, this requires storage of the hash value computed previously. There is no point computing a new hash for the same 2 users as it is always the same. Having such a hash function makes database management easier as hash ids are now more traceable, and we can also exploit the fact that we have stored hashes computed once before for the same 2 users, if we want to do caching. We therefore have less

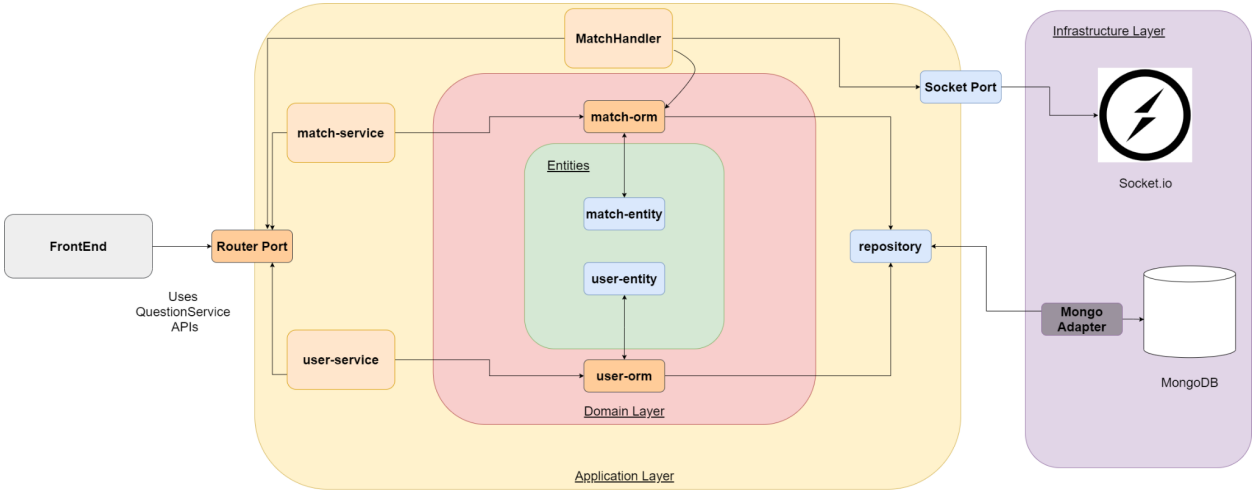
	needed in computing a fresh hash each time. We cannot exploit caching for performance due to uniqueness of new hashes, and combined with the above reasons, this thereby leaves room for attackers to exploit weaknesses. Even if we do have sufficient compute power and space, this also means constantly updating the room id and requiring the same 2 users to rejoin a new room with a new hash id, making the user experience terrible.	potential to be successfully attacked, with less vulnerability on performance.

Final Decision for Design Decision 2: The team has therefore decided to go with the approach on using hash functions that consistently give the same hash output given the same inputs.

However, many hashing methods exist and we want to have a hash that is consistent whenever we provide the inputs for computing the hash, while also satisfying conditions of pre-image resistance, second pre-image resistance, and collision resistance. In simpler terms, this hash has to be difficult to derive from any 2 usernames, has to be consistently the same when given the same 2 usernames, and it is difficult to find the 2 usernames given the hash.

For our purposes, we used a cryptography-industry standard: the SHA-256 hash which we believe to be suitable for our purposes.

4.3.4 UserProfileService

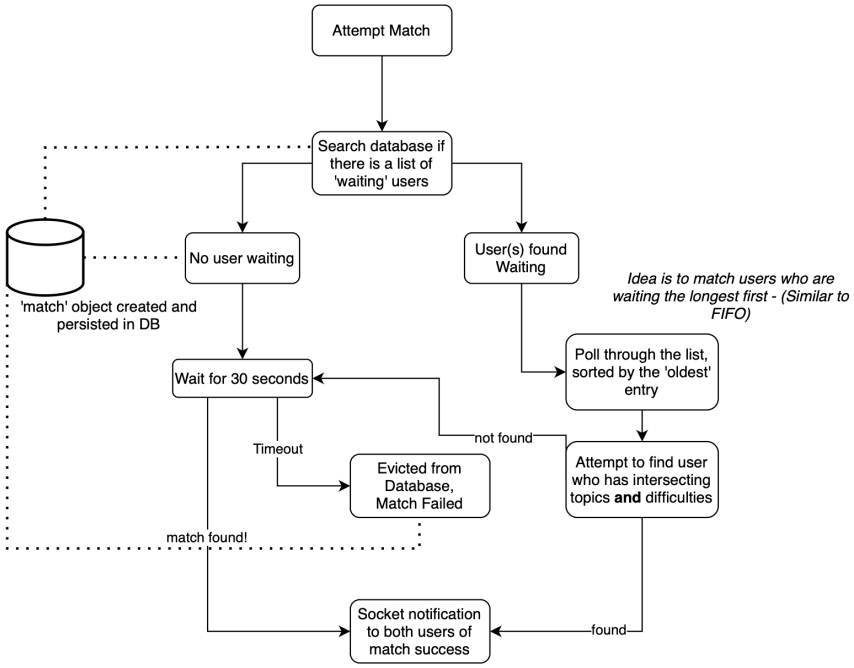


UserProfileService Application Core

The User Profile Service handles tasks related to account creation and login, and matches users together based on their expressed preferences. Do note that we have chosen the name ‘user profile’ here although we do not actually have a user profile, as we have planned for it to be extensible where users are able to eventually have profile specific information such as their history of questions done, for instance.

Overview

The user service adopts a Hexagonal Architecture as seen in the diagram above. The user service handles user account specific operations, such as account creation and login. Login with an account will return a JWT Authentication token, which will be required for all subsequent routes with other services, else a HTTP 401 or 403 error will be issued. The match service handles the matching algorithm for users to find a matching user that they could code with. As it requires informing the client (user) of the outcome of their match, it requires a socket.io connection, which facilitates duplex communication and enables push notifications to the front-end client.



User Matching Algorithm

APIs provided by the service

API	**Purpose**
POST /user/create	Create a user by providing a username and a password
POST /user/login	Login to an account by providing a username and a password

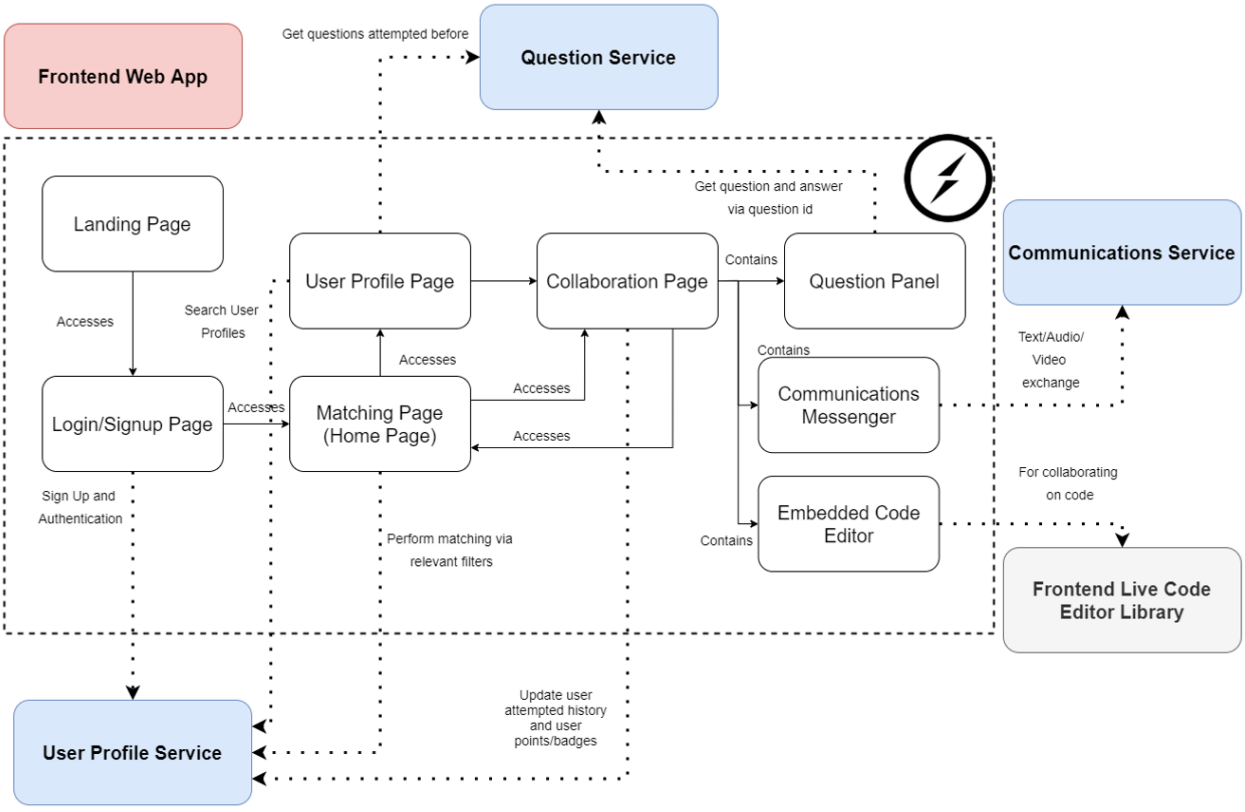
Sockets Interaction

Events listened	**Interaction**
match	Attempts to find another user match by providing an array of topics and difficulties that the user has chosen Returns either 1) matchSuccess: containing the username of the other user matched and the roomID 2) matchFail: Usually due to timeout, inability to find a match 3)

4.3.5 Frontend

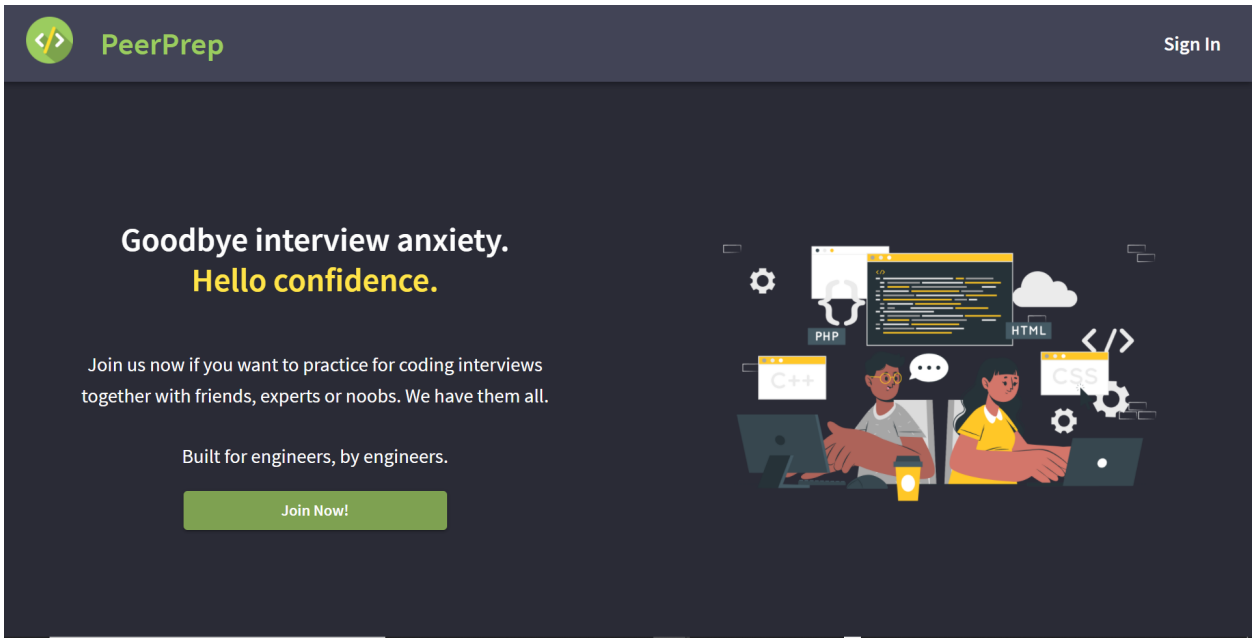
Given that visual appeals are important in today's world, and more so given the stresses of potential software engineering interviews, the team has focused on building a frontend which not only feels comfortable and is visually appealing, but also capable of reliable communication with the backend.

After all, it is our goal to produce a system that not only delivers value to the user, but also makes it a pleasant experience while interacting with PeerPrep. Moreover, since our focus is on a good user experience, we have carefully decided upon the following curated pages, as seen in the diagram below:

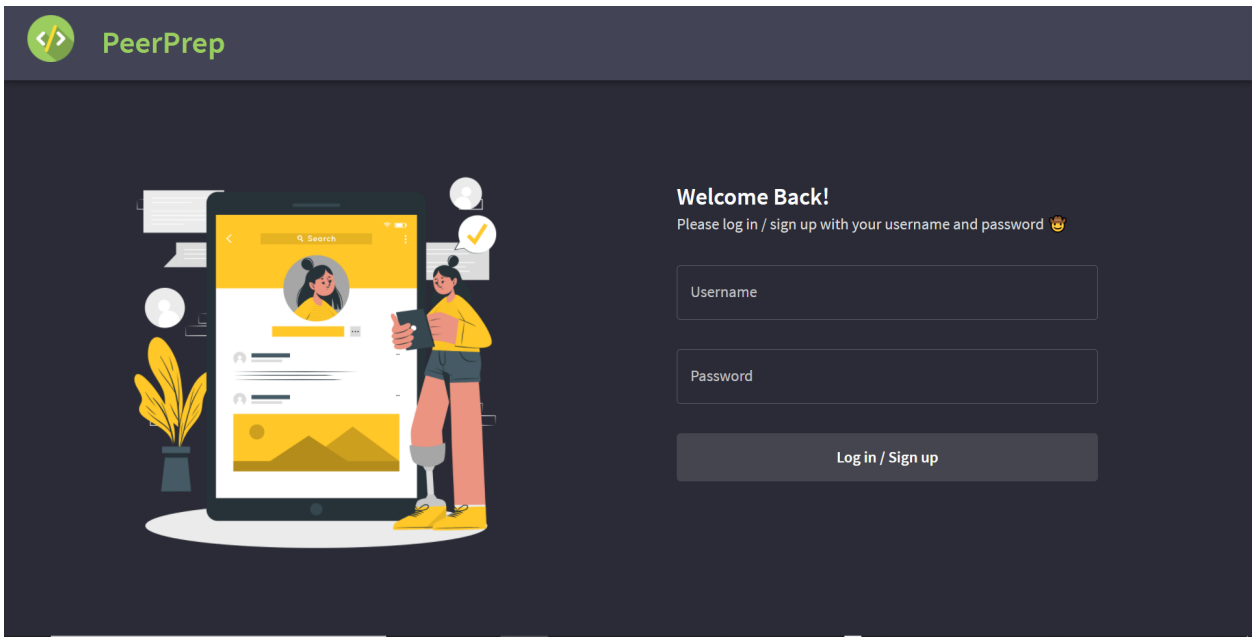


With respect to the above diagram for Frontend Web App, each component contains a single page within the application, and possibly containing sub-components within that page.

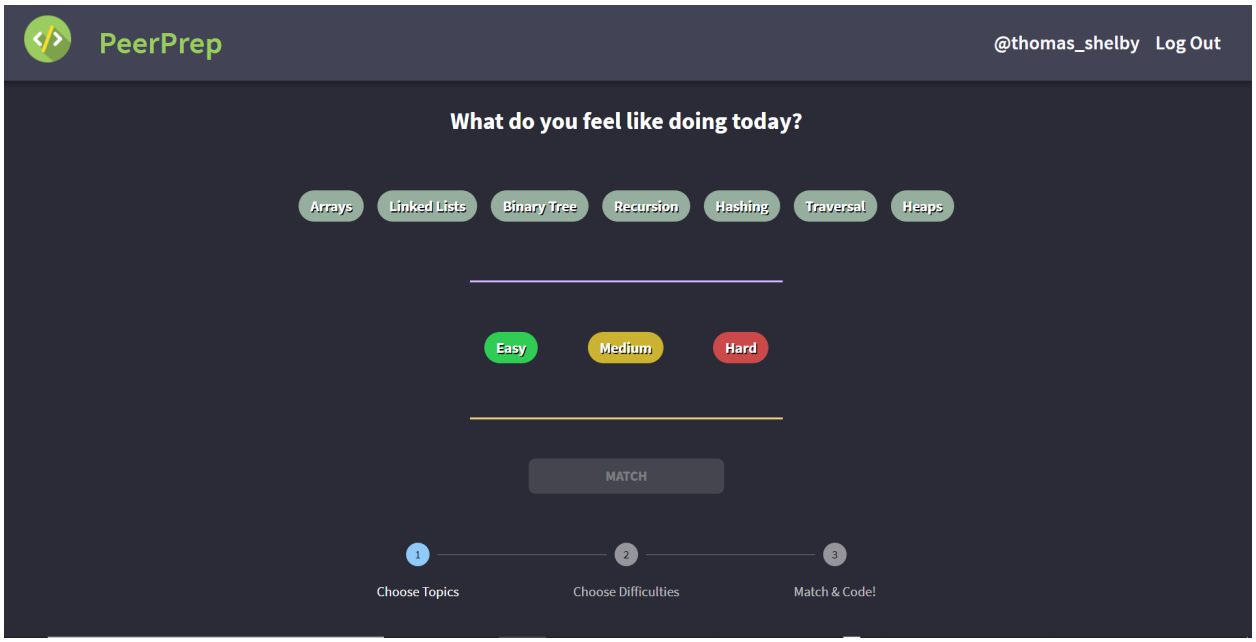
Firstly, the Landing Page serves as a base page that introduces to users what PeerPrep is about.



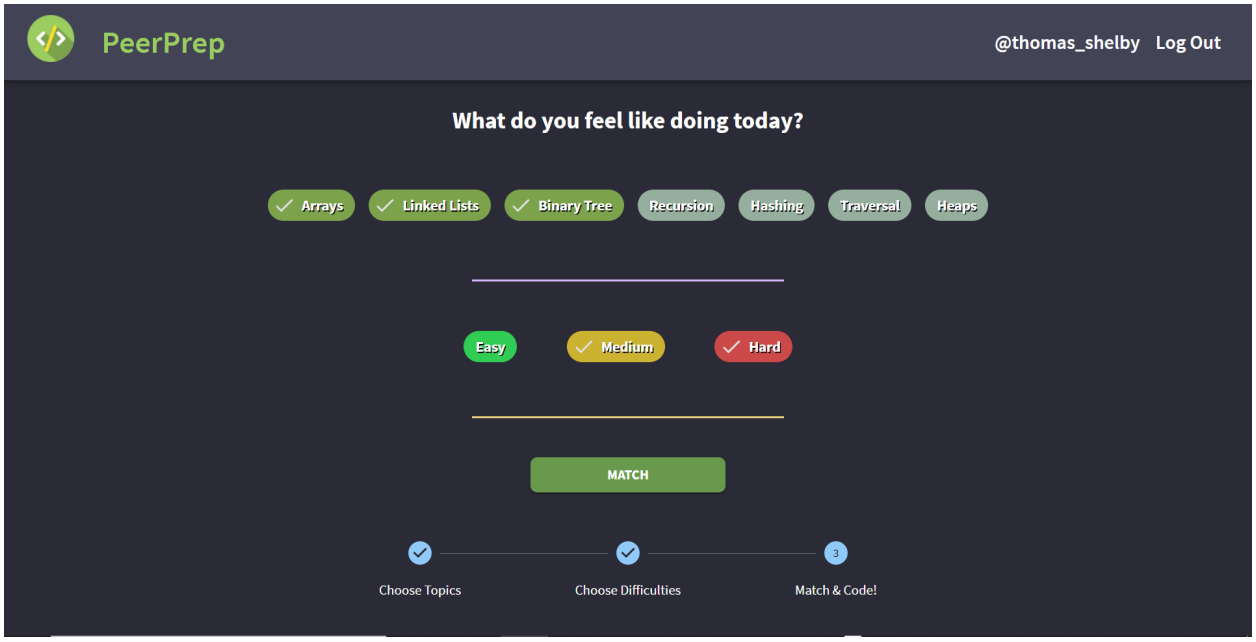
From there, users are prompted to the Login/Signup Page, where they can either login with an existing account or sign up for a new one. If a user enters details and no such username currently exists, we smoothen the process by directly creating a new account with the details provided and then logging them in, this makes for a more seamless user experience.



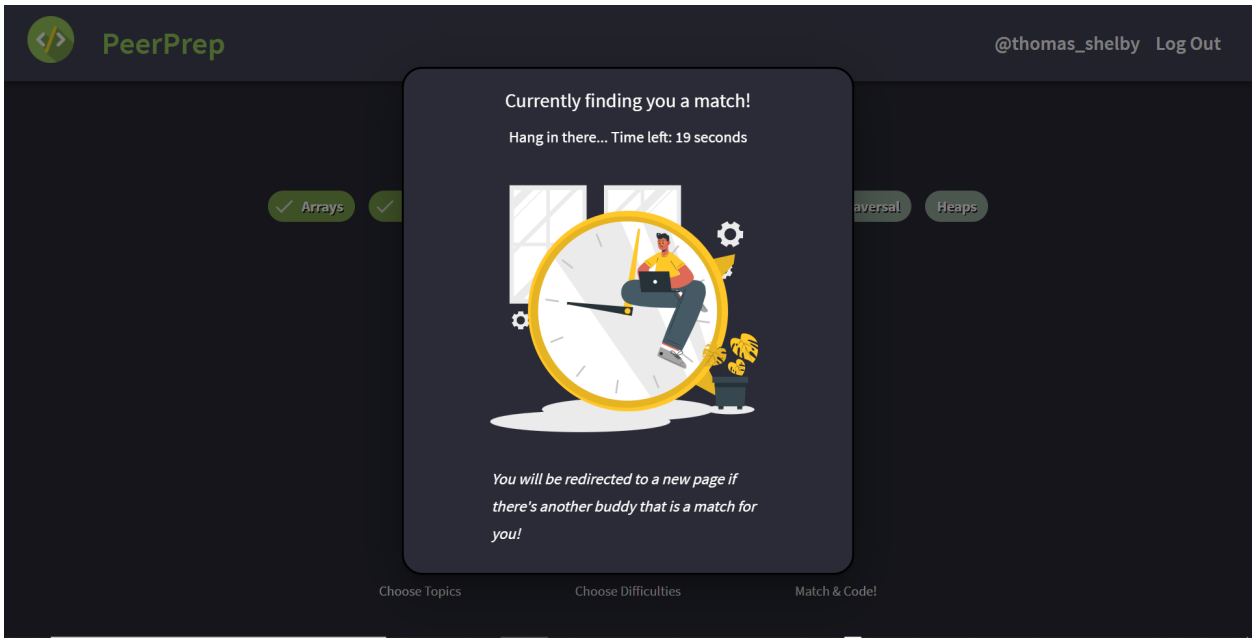
After which, users are directly to the Matching Page, which is where they select topics and difficulties provided and initiate the matching process. Note that the selection applies an "OR" instead of an "AND" so multiple selections increase the chances of matching with others.



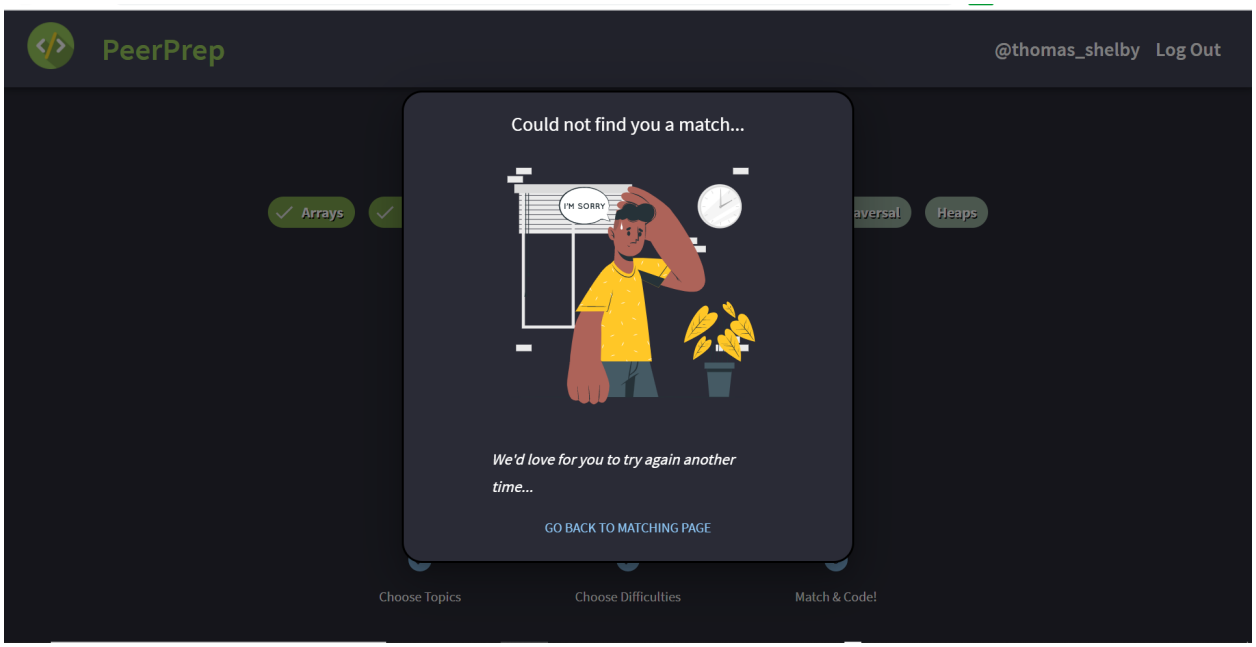
Note that only after topics and difficulties are selected, users are allowed to match, as indicated with the instructions below and the "MATCH" button lighting up. Refer to the following diagram for a better visualisation.



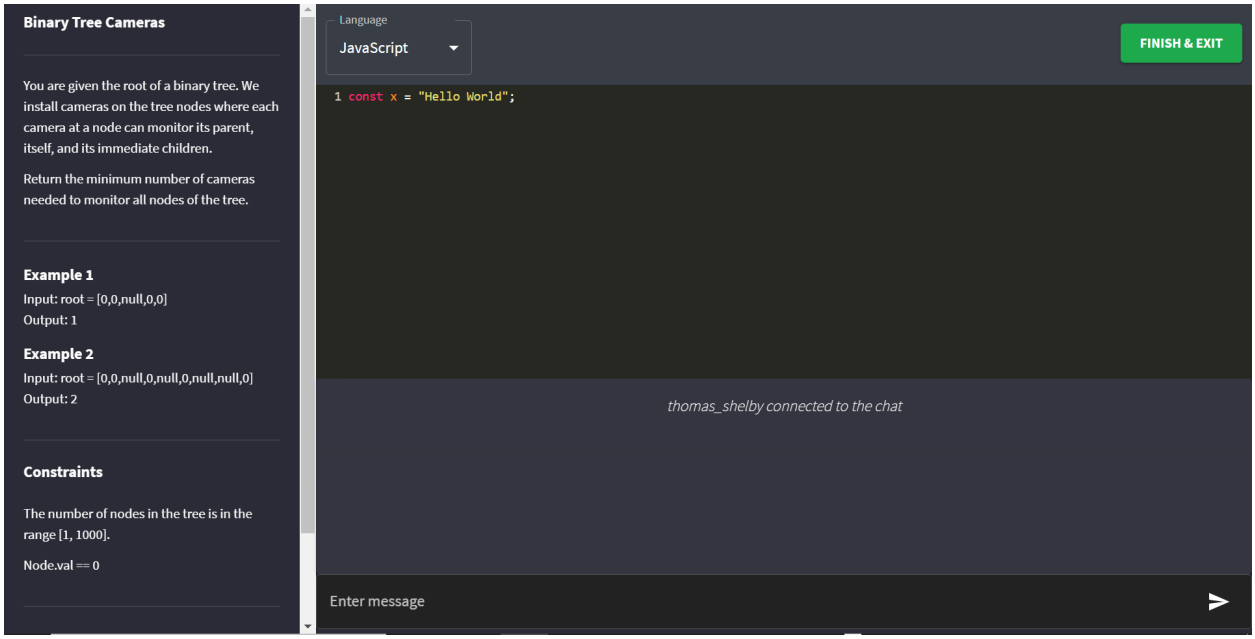
Upon hitting "MATCH", a modal pops up informing them about the ongoing wait time to be completely transparent to the user. This modal also includes instructions for the user about what happens in a successful match.



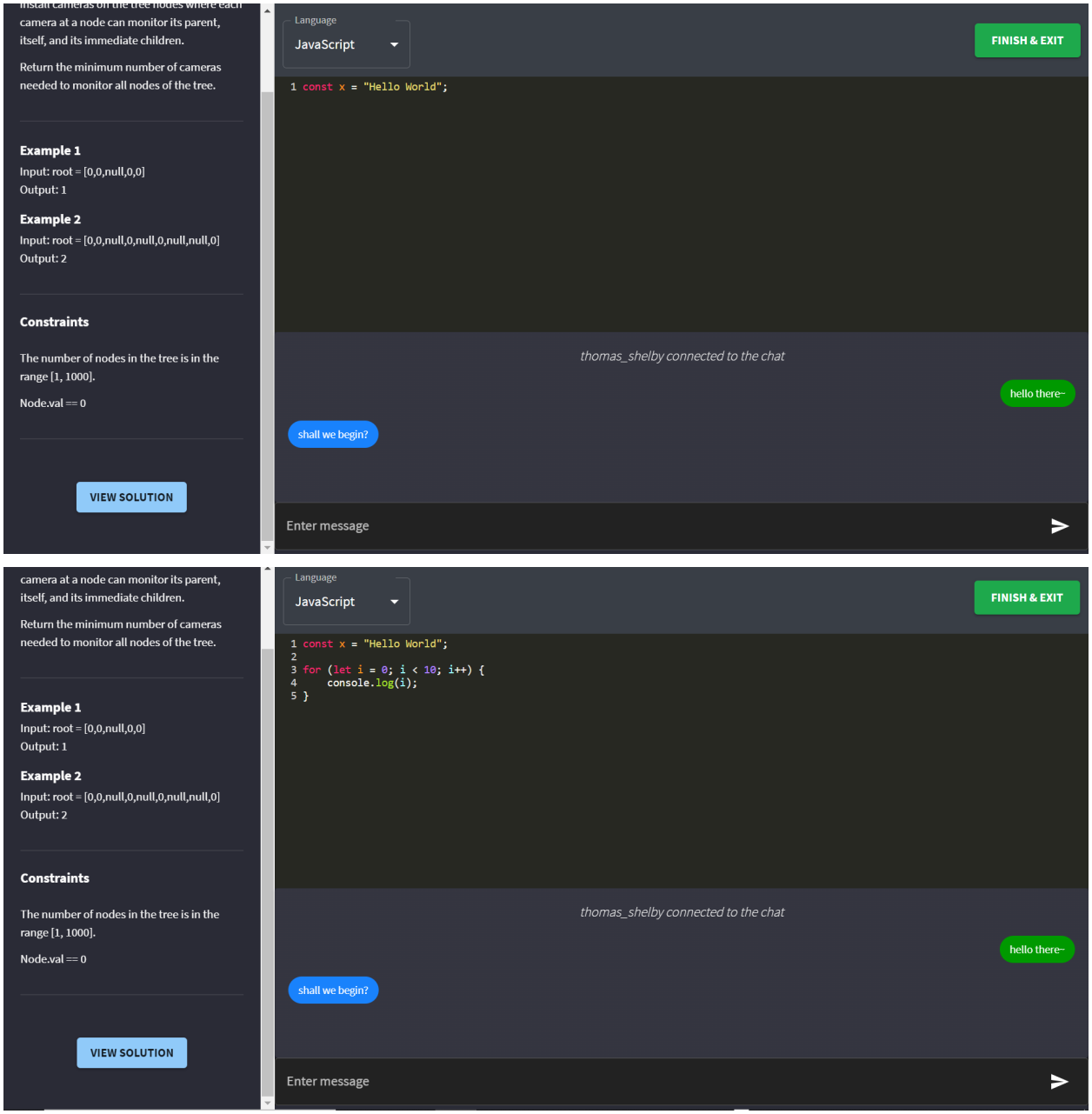
In the case 30 seconds times out, User Profile Service socket returns a "matchFail" event, which triggers rendering of the following notification to the user, that their request to match has not been successful and they could try again some other time.



In the case there is a successful match, the user is redirected to the Collaboration Page. This page offers the main functionality of the application: the collaborative code-editor. Besides that, it provides a text channel for the matched buddies to communicate freely. We've also made use of CodeMirror to provide support for syntax highlighting in popular programming languages like JavaScript, Java, Python and C++.



The Pub-Sub Messenger that we are using, Socket.io, features a client API that will be hooked onto the whole application. Therefore, every component is able to receive communication from the server, such as push notifications. The different components of the Frontend Web App will be communicating to the respective back-end services via RESTful API calls. Kindly note the text messages in the following diagram as well. There is also a "View Solution" button for users who are unable to solve the question, which opens a third-party solution page in another window/tab.



4.3.6 User Experience and Interactions between Services/FrontEnd (i.e. Within Microservices, Frontend-Backend)

This section deep-dives into the data flow between our various services, whenever a user interacts with our system. The following explanations are ordered from start to end of the user experience.

User loads Landing Page

At the landing page, the user is able to navigate to the Authentication Page using the login or signup buttons. At this point, there are no interactions with other services.

User attempts to Login/Signup

When user inputs the username and password, the following interaction occurs:

Responsibility	Collaborator	Service Invoked	APIs Invoked
Login / Signing up a User	Frontend Web App Service: ● Authentication Page	User Profile Service	● /api/user/create ● /api/user/login

Upon successful confirmation from the above API requests, **a JWT access token** is returned to the Frontend Client, which is used for every subsequent REST API request or socket connection request. The same JWT access token can be shared by the numerous microservices we have because we make use of the same secret code for all our services.

User attempts to match with another

The logged in user is now redirected to the Matching Page, where he/she is prompted to select difficulties and topics of interest to attempt matching with another peer. To ensure this is smooth and dynamic, instead of hardcoding the topics and difficulties, we fetch a list of possible topics and difficulties from the Question Service.

In addition, the Matching Page also sets up a socket connection with User Profile Service, which is in charge of matching users.

After selecting the topic(s) and difficulty/ies of interest, for example, "Easy" and ["Arrays", "Linked Lists"] respectively, the user is now allowed to match with another user, and then sends a "match" signal to the socket server at User Profile Service. It then waits on "matchSuccess" or

"matchFail" to determine whether to create a room and redirect the matched users to the Collaboration page or not.

Responsibility	Collaborator	Service Invoked	APIs Invoked
Getting list of possible topics and difficulties	Frontend Web App Service: ● Matching Page	QuestionService	● /api/question/questions/metadata
Finding a match	Frontend Web App Service: ● Matching Page	UserProfileService	● Emit socket message: "match" ● Listen for events: "matchSuccess" or "matchFail"

User finds a match and is redirected to Collaboration Page

Upon successfully finding a match (i.e. receiving a "matchSuccess" message), the matched users are both redirected automatically to the same Collaboration Page.

This page sets up sockets for Collaboration Socket Service (used for Text Editor) and Communications Service (used for the Text Messenger).

It then fetches the question for this specific room using QuestionService.

Responsibility	Collaborator	Service Invoked	APIs Invoked
Getting list of possible topics and difficulties	Frontend Web App Service: ● Matching Page	QuestionService	● /api/question/questions/metadata
Finding a match	Frontend Web App Service: ● Matching Page	UserProfileService	● Emit socket message: "match" ● Listen for events: "matchSuccess" or "matchFail"

User finishes question

After tapping the "Finish & Exit" button, the sockets are disconnected and Collab Socket Service triggers a request to delete the room on Question Service.

Responsibility	Collaborator	Service Invoked	APIs Invoked
Deleting Room on server-side	Frontend Web App Service: ● Collaboration Page Collaboration Socket Service	QuestionService	● DELETE /api/question/room/:id
Editing Text Changes	Frontend Web App Service: ● Collaboration Page	Collaboration Socket Service	● Refresh code every time "receive code" event is triggered ● Change chosen programming language for syntax highlighting whenever "receive lang" event is triggered
Sending Chat Messages	Frontend Web App Service: ● Collaboration Page	Communications Service	● Update list of messages to render whenever a "message" event is triggered

4.4 Technology Stack

In this section, our team specifies the various technologies and tools we have selected to implement our web application.

Factors that have led to this selection:

1. Prior Experience
2. Industry Standards

Category	**Chosen Technologies**
Front-End	**React.js and Material UI** Gentle Learning Curve, Declarative Components
Back-End	**Node.js and Express.js** Scalable and highly performant server application
Database	**MongoDB** Highly popular general purpose NoSQL database. It is highly scalable in horizontal terms, and suits the development of an evolving application with a fast changing schema
In-Memory Store	**Redis** Caching of temporary data, specifically code editor information
Deployment, Orchestration and Cloud	**AWS** Deployment of Microservices and Front-End. Provide API Gateway, Load Balancing, Scalability, Monitoring functionality as well
Pub-Sub Messaging	**Socket.io** (Client & Server) To enable real-time bi-directional communication between clients and the server
CI/CD	**GitHub Actions** Can easily integrate CI/CD workflow with our codebase on GitHub

5. Orchestration and Deployment

We aim to simulate delivering a product to a customer by deploying on the cloud. For this, we have chosen Amazon Web Services (AWS) as our main deployment provider, for its widespread usage, advanced configurations and a large developer community

5.1 Overview of Cloud Deployment and Architecture

For AWS, the main solution stack we used are:

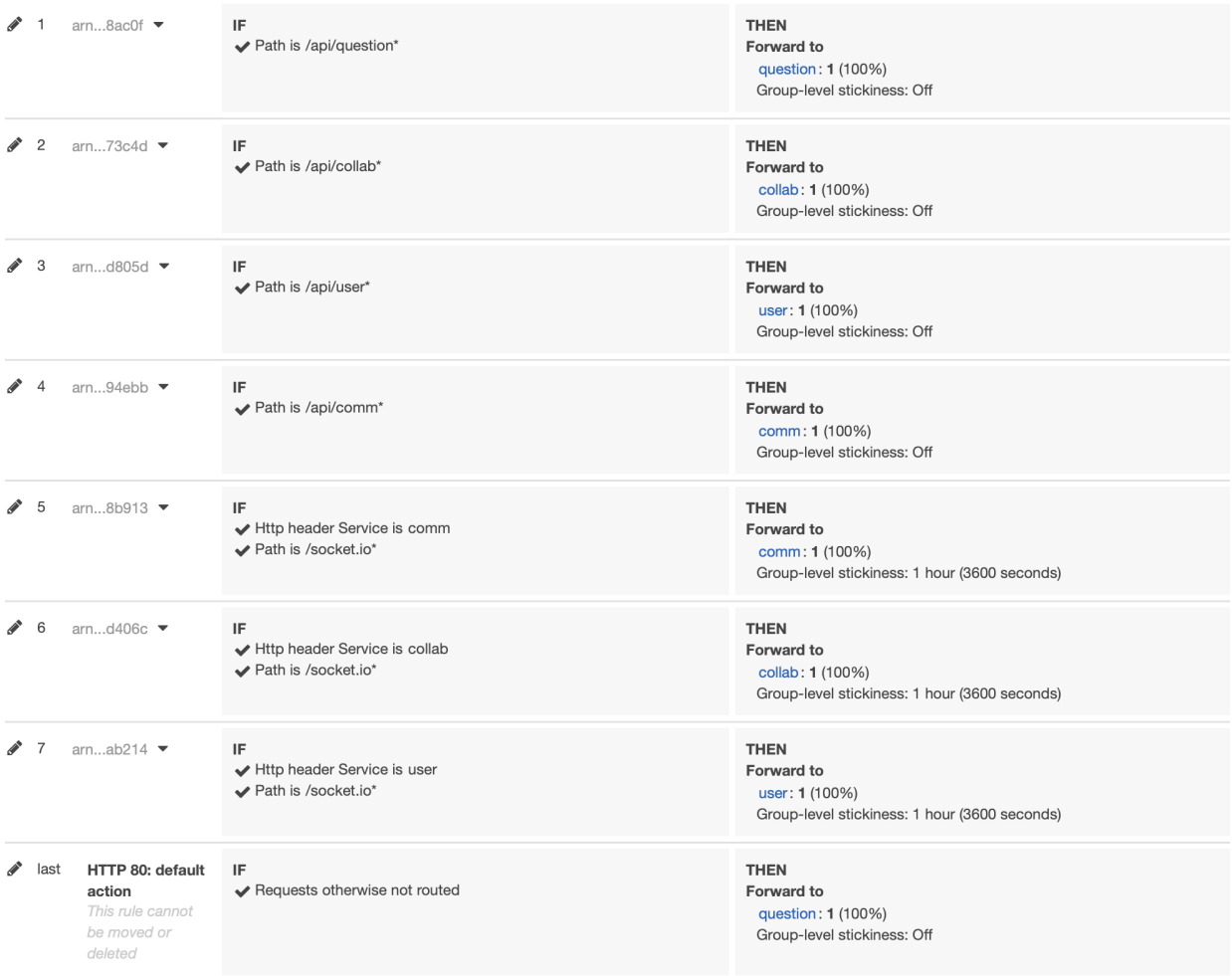
- Elastic Container Registry: remote storage for our Docker Containers
- Elastic Container Service: Container Orchestration and Management
- EC2: Cloud Compute Capacity. Includes Application Load Balancer that doubles as an API Gateway
- CloudWatch: Logging, Alarms, Monitoring
- Route 53: Service Discovery, DNS Resolution

AWS is useful as it is able to provide us with a range of functionality that are useful for real world production applications, such as a load balancer, API gateway, and more, which will be described in the sections below.

For our specific configuration, we deployed 2 instances of t2.micro EC2 Instance. This is sufficient to demonstrate the scalability of our architecture as we are able to deploy an additional number of instances of some of our microservices which requires more servers to handle the load.

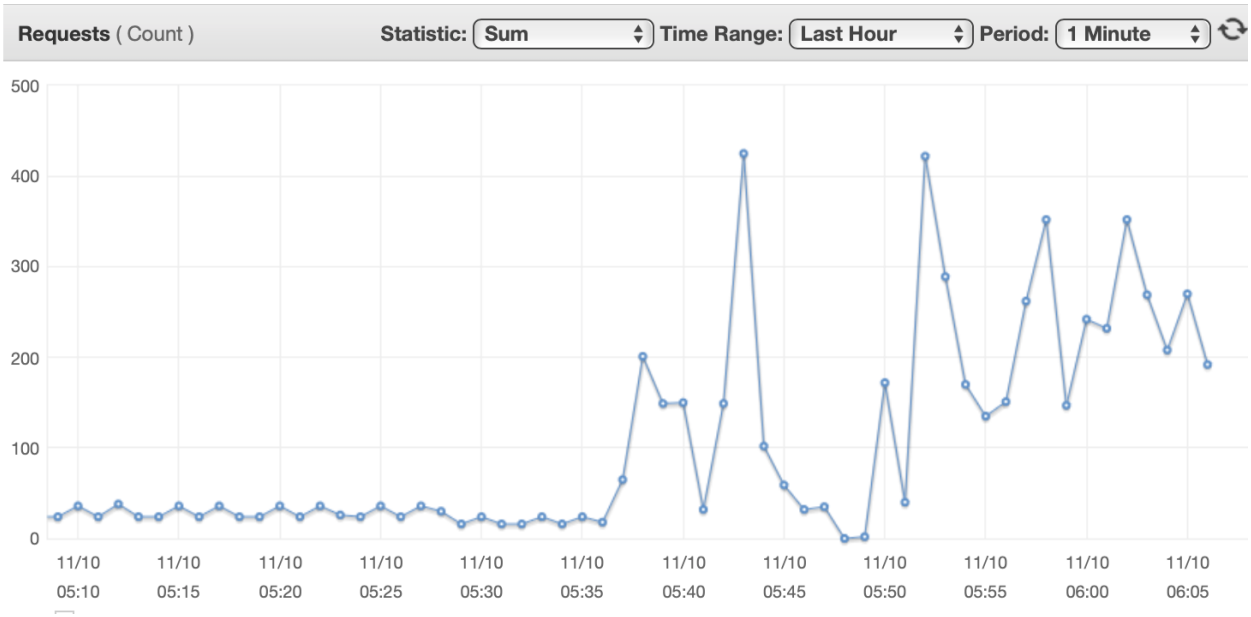
5.2 Load Balancer / API Gateway

EC2 Cluster provides a few types of Load Balancers to distribute the workload between its nodes. For our case, we chose the **Application Load Balancer** type, which implements listeners that only accept requests that match a certain type, and route them to that particular service. As a result, it also functions similarly to an API Gateway. The microservices deployed on the EC2 Node receive regular health checks, and will be graded on a healthy/unhealthy basis. If they are determined to be the latter, the microservice will automatically be shut down and a new one will be spun up to take its place. If there are multiple instances of a microservice deployed, it can also split the requests the load equally between all of them.



API Routing Rules listed on our Load Balancer listener

It also provides monitoring information about our requests to the server, enabling us to set alarms or trigger auto-scaling



Monitoring Chart of the number of Requests we were getting on our Load Balancer

5.3 Service Discovery

ECS also has a built-in feature of service discovery that enables services to be discovered by each other using a unique DNS Address that is resolved by looking up at **Route 53**, which essentially acts as a key value store to translate them to the actual private IP addresses to find other services. This is especially useful in the microservices architecture, where you need to

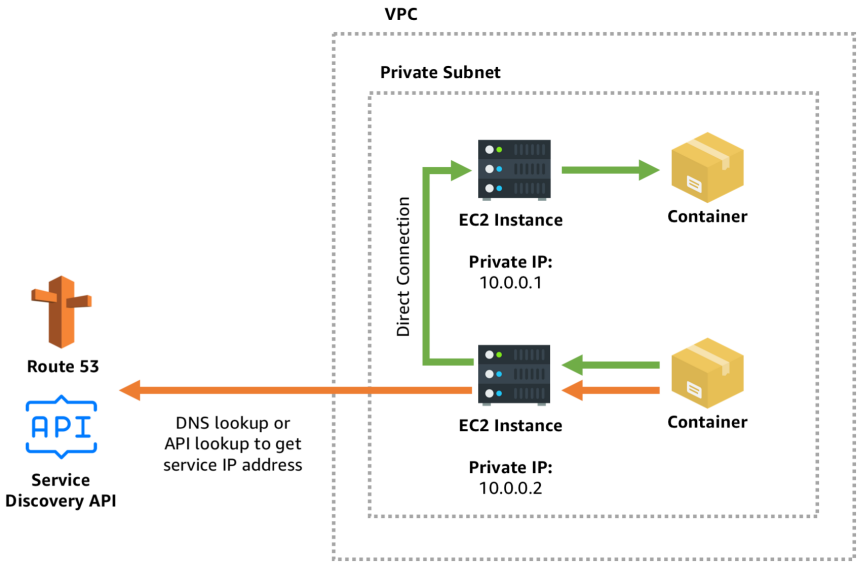
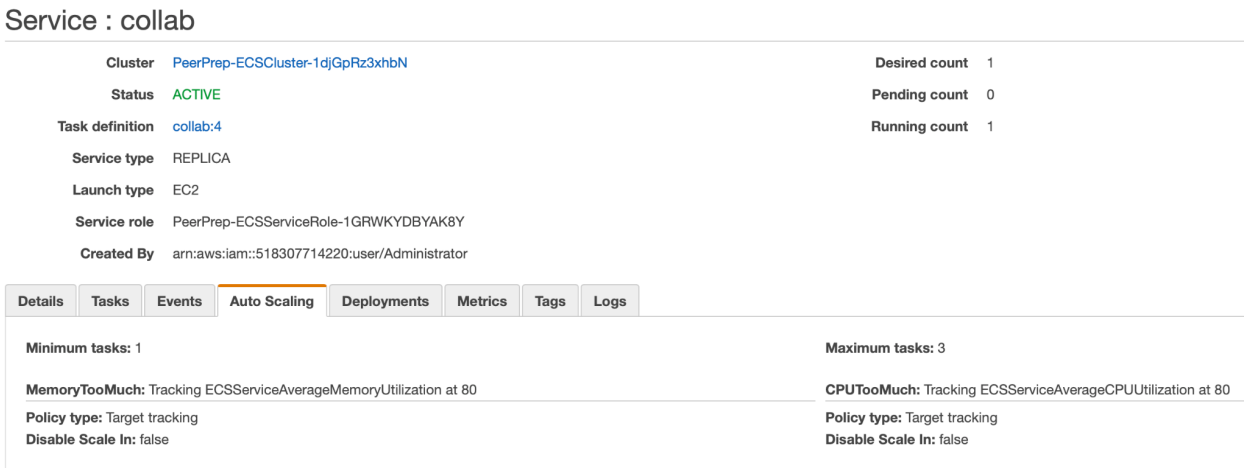


Diagram illustrating how Service Discovery works in AWS

5.4 Scalability

Orchestrating and starting up services according to templates (task definitions) is just one aspect of what AWS ECS does. It also publishes CPU and memory usage to another service known as CloudWatch, which would automatically scale our architecture (spin up more instances of microservices(s) to deal with high demand at peak periods, and conversely scale in to reduce costs during periods of low utilization. It offers a variety of scaling policies, and we have chosen the **Target Tracking Scaling Policy** which will increase/decrease the number of tasks based on Memory usage metric.



Example Auto Scaling Configuration on our CollabService

6. Remarks

6.1 Challenges Faced

Implementing PeerPrep and bringing it to the cloud was definitely challenging, given our team's initial limited experience in DevOps.

Furthermore, there were numerous edge cases we had to meticulously note when performing manual integration testing between frontend client and the respective microservices. These edge cases often pertained to creation and connection of web sockets from socket.io. Through solving each of these edge cases, our team gained a much better understanding of web sockets and React lifecycle hooks.

6.2 Potential Future Extensions

As mentioned in the Project Management Section (#3.3), the team was ambitious, brainstorming and providing the teaching team with many potential functionalities for PeerPrep. In this section, we shall address these additional functionalities which have not yet been implemented, but are good to have (and placed as low priority in our CS3219 semester project plan).

For simplicity, we have labelled these potential future extensions according to their relevant FRs. FRs will be grouped according to similarity of the proposed feature, and we shall also indicate the expected difficulty of implementing the corresponding feature.

Also note that while the Project Management Section describes feasibility and suitability, this section simply ignores those notions and provides a brief general approach towards implementing features not yet included in our current submission.

FR1e: The application provides matched users with an option to confirm or reject pairing FR1f: The application provides a preference option for pairing with users of a spoken language
Difficulty: Simple
Approach: ● Modify Matching Page to include a drop-down list for language preferred ● Instead of redirecting immediately as the current submission is doing, simply present a modal when loaded. ○ If user accepts, redirect (as per the submission) ○ If user rejects, simply use socket.io to delete the room and leave it ● UserProfileService will now have to store languages of each user for matching

FR2f: The application issues a disclaimer that previously encountered questions have been attempted before

FR4e: The user profile tracks past questions attempted by the user

Difficulty: Simple

Approach:

- Modify Collaboration Page to include a disclaimer (such as in the form of a React Card) to indicate the question has been done before.
- UserProfileService will now have to store past completed questions of each user, and be called to obtain this information for checking against the current question

FR3b: The application provides users ability to get into an audio call with each other using Audio Chat

FR3c: The application provides users in an audio call the option to mute and unmute, showing the muted/unmuted status.

FR3d: The application provides users with a video chat feature

FR3e: The application provides users with an option to turn video on or off during video chat

Difficulty: Difficult

Approach:

- Modify Collaboration Page to include a component for video interaction (mute/unmute, show/hide video)
- Implement or import some form of audio/video transmission tool/technology
- Depending on transmission tool/technology, you might have to implement transmission protocol / interactions between audio/video transmission among users and between the frontend-backend

As you can see, further research is required for this feature, and it is out of scope for our project.

FR4b: The application offers users ability to search and navigate to other users' profile pages using their username

FR4d: The profile page shows the online status of a user

FR4e: The user profile tracks past questions attempted by the user

FR4e1: The user profile shows a heatmap of dates of attempts

FR4c: The user profile page provides the user the option to pair up directly with users of choice from their profile (On Demand, Manual Matching)

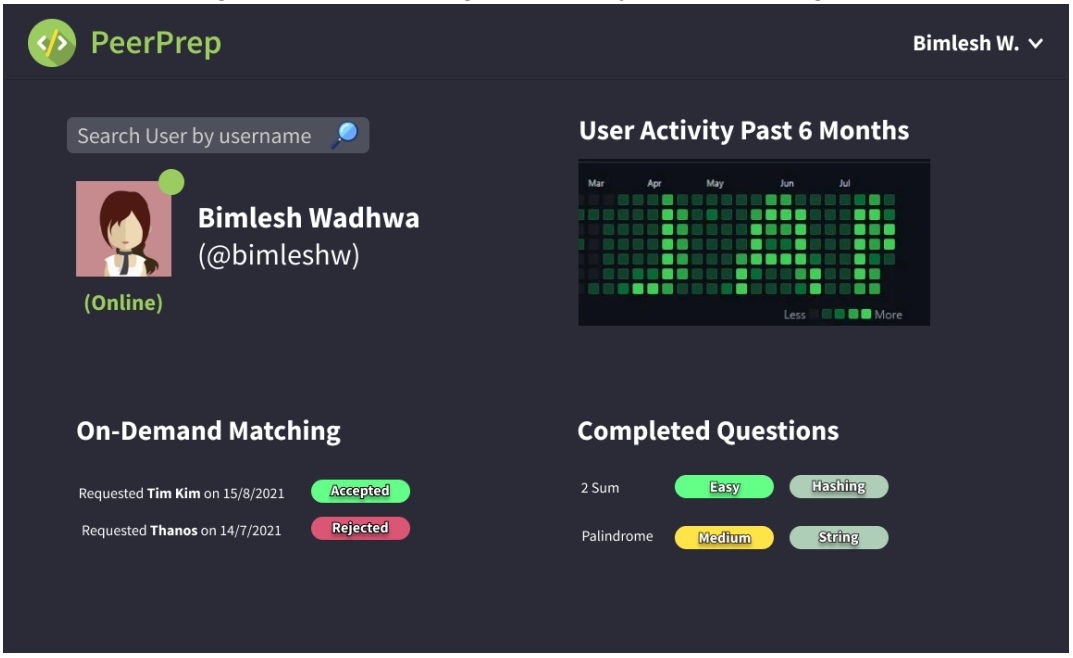
FR4c1: The application notifies **online** invited users once someone sends them an invite

FR6a: Application provides users with points whenever a problem is marked as complete, and the users can earn badges for every 100 points received

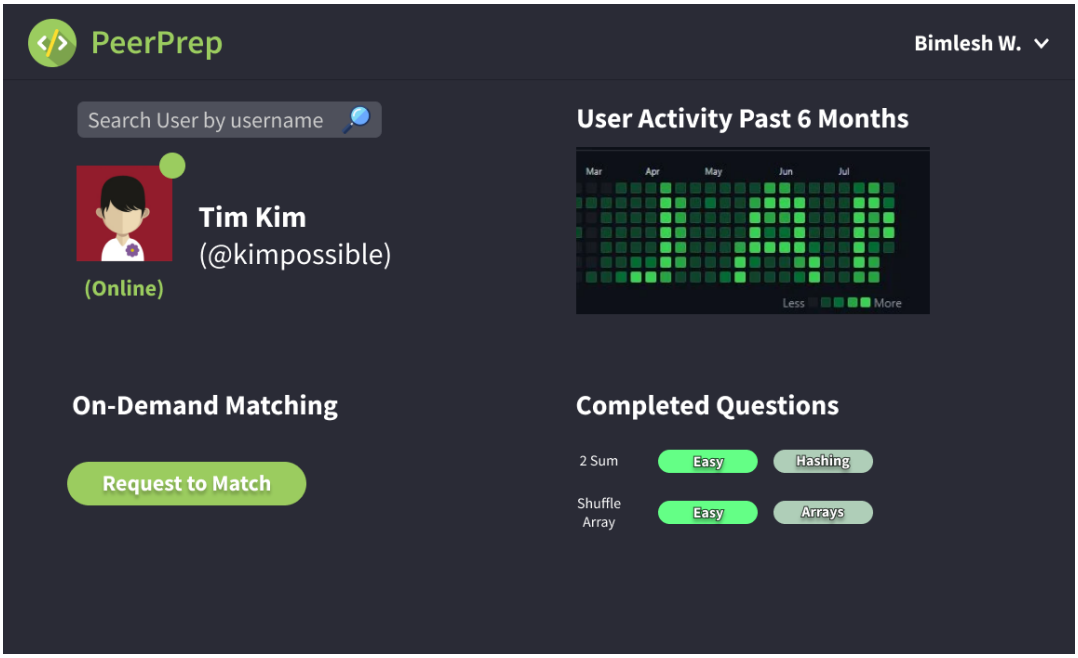
Difficulty: Medium-Difficult

Approach:

- Create a new page, User Profile Page, to display the profile page of a user, such as



- Implement online status indication, requiring access to backend's stored list of online users, depending on how the team chooses to implement this.
- Implement search feature for users, and ability to view other users' pages, which is slightly different from viewing a user's own page. This requires a new search method in UserProfileService, and changes in the frontend. When view others' pages, it should look something like



- Track questions attempted by user, and history of past matches by storing in UserProfileService. This requires the backend to store the relevant information. Frontend requires displays to present these information.
- Implement an on-demand Matching, requiring a new method in the UserProfileService, and the respective calls and its button on the frontend.
- Implement socket for receiving on-demand Matching invites, as well as their modals for notification.
- For gamification, simply use the backend to store and accumulate points, implementing a frontend component to present this information.

As you can see, this set of features are very inter-related and can be grouped as a social media feature. It is therefore complex and expected to be difficult to implement. Although it is deemed as out of scope for our project, we implore future developers to work on this aspect, if they would like to amplify the social aspect of PeerPrep, or to commercialize it.