Peerprep Report

CS3219_Group_56

KANG YUE RAN

LINUS LEE HONG FENG

LOW JIA HAO

PEI CHENG YI

BRYAN YANG

Background

In the landscape of software engineering job applications, the technical interview stands as a significant challenge, particularly for those entering the job market for the first time. While numerous tools and platforms offer preparation assistance, they often lack specificity and tailoring to the unique requirements of distinct user groups, which can be critical for effective interview preparation.

Purpose

The Peerprep platform is meticulously designed to bridge this gap, targeting the precise needs of novice job seekers. Our primary audience encompasses first-time job applicants from the National University of Singapore (NUS), specifically targeting first-year students in their summer break, given that these students would be new to the interview process and that they would be preparing for interviews in their second year. Peerprep is engineered to not only prepare these candidates for the technical rigors of interviews but also to instill in them a nuanced understanding of real-world collaborative problem-solving, as reflected in Functional Requirements (FR) like F2.3, which emphasizes the ability for users to communicate effectively during a mock interview session.

Additionally, the platform's design, informed by Non-functional Requirements (NFR) such as NFR2.2, which stipulates low latency for a smooth user experience, ensures that the simulated interview environment is as realistic and interactive as possible. Peerprep's commitment to providing a user-friendly interface that is consistent across various browsers (NFR4.1) shows our dedication to accessibility and usability, catering to the diverse technological preferences of our user base.

Through these carefully considered functionalities and requirements, Peerprep aspires to equip first year computing students with the confidence and competence to excel in their future technical interviews.

Work Allocation

Individual

Name	Work done
Bryan Yang	1. Question service: added middleware, parameter validation and implemented endpoints 2. Matching service: added middleware, cancel matching functionality, implemented linked list + map for efficient queuing/ dequeuing 3. Room service: added middleware to ensure create room is only called from matching service 4. User service: added middleware 5. Frontend: added middleware and implemented the following: a. Left side of collab room (question display + interviewer notes) b. Home page (questions table) c. Matching page d. Onboarding page e. Question page (individual question [edit/view modes]) 6. Added a bash script for frontend dockerfile entrypoint to inject environment variables at runtime.
Pei Cheng Yi	1. Set up, implemented, and dockerised User and Question service 2. Implemented the signaling server for Y-WebRTC. And added it into the video token service to 3. Implemented collaborative code editor using Code Mirror and Y-WebRTC. 4. Implemented the view page for individual past collaboration.

	5. Implemented the Right side of the Collaborative Room 6. Planned and proofread the report
Low Jia Hao	1. Integrate the frontend service and backend services 2. Test the system after integration and check for bugs 3. Project Report 4. Setup CI/CD pipeline 5. Added automated unit testing to question, matching and user services
Linus Lee Hong Feng	1. Set up and implemented video_token service and room_service. 2. Implemented video calling in the collab room using agora. 3. Implemented history table which stores the past collaborations that the user had. 4. Set up judge0 locally using docker-compose so that code in the collaboration can be executed. 5. Link the flow from matching service (match found) into the collaboration room.
Kang Yue Ran	1. Establish websocket connections for matching service 2. Develop serverless function and scrape external API to populate question database 3. Set-up and configure Kubernetes cluster in Minikube environment 4. Deploy on Google Kubernetes Engine 5. Set-up Nginx API gateway as an external load balancer on GCP 6. Use Kubernetes' built-in DNS service for intra-cluster communication 7. Use Kubernetes' Horizontal Pod Autoscaler to scale application according to load

Sub-Groups

Sub-Group Category	Members	Nice-To-Have features	Extra notes
Development	Bryan Yang, Pei Cheng Yi, Linus Lee Hong Feng	**N1**: Communication: Implement a mechanism to facilitate communication among the participants in the collaborative space (other than the shared workspace) e.g., text-based chat service and/or video (+voice) calling service. **N2**: History: Maintain a record of the questions attempted by the user e.g., maintain a list of questions attempted along with the date-time of attempt, the attempt itself and/or suggested solutions. **N3**: Code execution: Implement a mechanism to execute attempted solution/code in a sandboxed environment, and retrieve+present the results in the collaborative workspace. **N4**: Enhance question service to enable managing questions, for example, tagging (by topic,	

		popularity, etc.,), retrieving questions on the fly during a session initiation. **N5**: Enhance collaboration service by providing an improved code editor with code formatting, syntax highlighting for one language, syntax highlighting for multiple languages.	
Dev/Ops	Low Jia Hao, Kang Yue Ran	**N8**: Extensive (and automated) unit, integration, and system testing using CI. Teams can also use various test frameworks or demonstrate effective usage of CI/CD in the project **N9**: Deployment of the app on the production system (GCP) **N10:** Scalability - the deployed application should demonstrate easy scalability of some form. An example would be using Kubernetes horizontal pod-scaler to scale up the number of application pods when there is a high load. **N11**: The application should have an API gateway of some kind that redirects requests to	N8: Application does not have extensive testing but have some testing and CI/CD

		the relevant microservices. An example would be using an ingress controller such as NGINX ingress controller if using Kubernetes (https://kubernetes.GitHub.io/ingress-nginx/). **N12**: The application should demonstrate service discovery or implement a service registry of some kind. For example, Kubernetes has built-in DNS features and service networking to pods (https://kubernetes.io/docs/concepts/services-networking/dns-podservice/).	

Requirements of the Application

Functional Requirements

Description	**Mx/Nx Refinement**	**Reasoning**
FR1 Account / User Management		
FR1.1 Account can be created by users.	M1	Users have accounts to store their information to use our website

FR1.2 Users access the service by logging into the account.	M1	Users login to authenticate their access to our services
FR1.3 Users can update their profile details.	M1	Users can change information stored about them to facilitate their usage of our website
FR 1.4 The database will store the latest edited datetime for the attempt on the question by the user	N2	This allows us to determine the duration of the interview and feedback this to the user so they may use this to monitor their performance.
FR2 Matching System		
FR2.1 Users can select the difficulty of the question(Easy, Medium, Hard).	M2	Users should be able to choose the difficulty of questions they want to practice with
FR2.2 Users should be matched with other users who select the same difficulty of the question.	M2	Users should match with others who select questions with the same difficulty so they can collaborate to work on the questions they wish to do
FR3 Collaborative Space		
FR3.1 Both users can see the question.	N5, N6	This allows both the interviewer and interviewee to clearly access the question and be on the same page when discussing their approach / tackling the question.
FR3.2 Both users can edit the	N5, N6	This collaborative experience allows

code simultaneously with the changes reflected in real-time		the interviewer and interviewee to cooperate in solving the problem, mimicking actual interviews
FR3.3 Users should be able to leave the session without issue.		This allows users to end the interview session if they feel uncomfortable during the interview
FR3.4 Display output of code to stdout from the execution of code in a separate window	N3	This allows the user to test their code to debug and assess the correctness of their approach
FR3.5 Users should be able to communicate with one another.		In a technical interview, the interviewer will ask the interviewee to walk through the thought process which is mimicked in the application
F4 Question Manager		
FR4.1 Users of higher role level can add, delete and update questions.		This prevents normal users from adding bogus questions and polluting the question pool. Only maintainers who are authorised to add questions should be able to do so
FR4.2 Users can sort and filter questions based on difficulty	M3	This helps users to narrow down questions within their comfortable range
FR 4.3 All users can view the questions		This allows users to access a wide range of questions and preclude the situation where one participant sees the question while the other does not

FR5 History View		
FR 5.1 Lists past collaboration sessions with details	N2	Allows users to review past interviewers to improve
FR 5.2 Lists session duration	N2	Allows users to gauge how long they are spending per interview and improve accordingly

Non-functional Requirements

Description	**Mx/Nx Refinement**	**Reasoning**
Security		
NFR1 Account Management		
NFR1.1 Users can only request the magic link every 60 seconds.		To reduce workload on the database
NFR1.2 The system will support the following file formats for display picture: jpg, png, gif of maximum	M1	This gives the user flexibility to the type

size 5MB		
NFR 1.3 Profile page have to be protected and only allow the users to access their own page		For security reasons, as users should not able to perform CRUD on other users' data
Usability		
NFR1 Matching System		
NFR1.1 A timeout period of 30 seconds should be placed in the system.		The time limit prevents users from queuing for long periods of time and will be notified when no match is found.
NFR1.2 It should be able to handle 200 concurrent users queuing.	M2/N10	200 users is the amount of concurrent users we expect given our target audience. This ensures that our system is able to support our intended bandwidth. This number can easily be increased with more budget to deploy more pods on more nodes across multiple regions.
NFR2 Collaborative Space		
NFR2.1 There should be sufficient nodes to handle 100 ongoing collaborative spaces.	M4/M6/N9/N10	200 users can share 100 collaborative spaces, which is the amount of concurrent users we expect given our target audience.

		This ensures that our system is able to support our intended bandwidth. This number can easily be increased with more budget to deploy more pods on more nodes across multiple regions.
NFR2.2 The latency should be less than 200 ms to ensure a smooth user experience.	M4	Peerprep aims to imitate a real interview experience, so having a low latency of less than 200 ms would ensure that there can be a smooth conversation held.
NFR 2.3 Both users can edit the same code space instances simultaneously	M4	The collaborative nature of interviews may require input from both users at the same time. The collaborative code space would provide both users an easy way to collaborate.
NFR 2.4 Supports Java, Python3, Javascript	N3, N5	These are the most commonly used language in year 1 and should be familiar to most year 1 computing students.
NFR 3 Question Manager		
NFR 3.1 Maintainers can add questions with Markdown and KaTex syntax.		Allowing maintainers to add questions using Markdown and KaTeX syntax provides a flexible and efficient way to create and format content within the system. Markdown is a lightweight markup language that allows maintainers to structure and stylize questions with headings, lists, and emphasis, enhancing readability. Additionally, incorporating KaTeX syntax enables the inclusion of

		mathematical equations which can be necessary for some types of questions.
NFR 3.2 The system will support classifying of questions by difficulty levels (easy, medium, hard)	M3	Questions of differing difficulty levels would appeal to users of differing ability. Someone just starting off would start on easy questions and progress to solve more difficult questions as their skills improve.
NFR 3.3 The database should be able to handle 2000 questions	M3	Leetcode, a premium tech interview practice platform, holds 2891 questions. Being able to handle at least 2000 questions should provide a large and diverse enough question bank for our users.
NFR 3.4 Timestamps are stored in Singapore time(GMT+8) / Standardised to the same time format	N2	Standardising the time format allows us to display the user's interview question in chronological order. It also enables us to display the time elapsed for an interview.
NFR 4 User Interface		
NFR 4.1 The user interface should be bug-free on Chromium, Chrome, Firefox, Safari, and Edge browsers.	M5	These browsers are the major ones being used today and are crucial for providing a consistent experience for a diverse user base. Each browser has its unique rendering engine and user interface nuances. Bugs or compatibility issues

		may arise due to these differences, potentially leading to functionality breakdowns, distorted layouts, or compromised user experiences. By prioritizing bug-free performance across these major browsers, we can guarantee a seamless and reliable experience for most users.
NFR 4.2 The user interface should allow users to choose between light mode and dark mode.	M5	Implementing both light mode and dark mode in a user interface caters to diverse user preferences, contributing to a more personalized and comfortable experience. Dark mode reduces eye strain, particularly in low-light conditions, and can save battery life on certain screens. It enhances accessibility for users with visual impairments, aligns with design trends, and allows users to adapt the interface to different contexts.
NFR 4.3 Errors should be shown as pop-ups		This is to ensure implementation details are not leaked and also to inform users who are not tech-savvy, by displaying an error message that they can understand
NFR 5 CI/CD		
NFR5.1 The production environment should be isolated from the development and staging	N9	Production environment is different from the development and staging environments. Unintended changes and data leaks can be prevented through this

environments		requirement
NFR5.2 Secrets should be well hidden from users in production environment	N9	When deployed to production, malicious users will try to figure out environment variables. Thus it should be protected to prevent leaks.
NFR5.3 Testing should be included in the pipeline	N8	Testing is crucial as the development environment may be different from the production environment. Furthermore, it is to prevent regression.
NFR 6 Kubernetes Cluster		
NFR 6.1 The Kubernetes cluster must ensure 99% uptime, with the ability to automatically replace failed pods to maintain service continuity.	N9	99% uptime and automatic pod replacement is to ensure that the services hosted on the Kubernetes cluster are almost always available to users, with minimal disruption.
NFR 6.2 The Kubernetes cluster should automatically scale the number of pods up or down based on CPU or memory utilization thresholds, with the ability to scale to a minimum of 1 pod and maximum of 10 pods per service.	N10	Automatic scaling of pods based on resource utilization ensures that the cluster can handle varying loads without human intervention, allowing for cost-effective resource usage.
NFR 6.3 The application	N11	By having an API gateway, we create a

should have an API gateway to redirect requests to relevant microservices, preventing direct external access to services within the cluster		single entry point into our system, preventing unauthorized access and potential security breaches by ensuring that direct external access to the microservices is restricted.
NFR 6.3 The cluster must provide a service registry and discovery mechanism that allows services to discover and communicate with each other using service names instead of IP addresses.	N12	This allows services to locate and communicate with each other in a dynamic and scalable way by decoupling services from their network locations.
NFR 6.4 The system must include comprehensive monitoring and alerting capabilities to track the health and performance of all services.		Comprehensive monitoring and logging are necessary for proactively identifying and resolving issues, maintaining operational health, and ensuring the accountability of all actions within the cluster. Alerts based on predefined metrics allow for quick response to potential issues before they affect users.

Developer Documentations

This documentation encapsulates the team's developmental journey, chronicling the methodologies adopted, decision-making processes, and project management practices.

Project Management

Our team embraced an Agile framework with a flat hierarchy, given the small size of our team. This iterative approach, coupled with a non-hierarchical structure, fostered flexibility and responsiveness to change. We engaged in weekly sync-ups every Thursday evening. These sessions served as a platform for active dialogue, ensuring alignment on project objectives and progress.

To promote accountability and project momentum, we established realistic milestones aligned with our Agile Sprints. We added branch protection to our main branch on github where every Pull Request (PR) required endorsement from at least one other team member before merging. This practice not only upheld code quality but also facilitated knowledge sharing and collective code ownership.

Overview of Development Process

Initial Design and Planning Phase

Our developmental journey commenced with an intensive planning sprint, wherein the entire team congregated to delineate the project's foundational aspects. In this formative phase, we articulated the project's scope, established our objectives, and identified our target user demographic. A pivotal outcome of this sprint was the creation of a comprehensive Figma design for the frontend, which crystallized our collective vision for the user interface. This collaborative design process ensured unanimity in our approach, setting a solid groundwork for the subsequent development phases.

Agile Development and Team Dynamics

With a cohesive design blueprint in place, we transitioned into our core development phase, adopting an Agile methodology characterized by biweekly sprint cycles. We bifurcated into two sub-teams, each with distinct yet complementary responsibilities, operating in parallel to maximize efficiency and productivity. This strategic division allowed for focused development on different fronts of the project, fostering specialization while maintaining a synchrony towards a shared destination.

Project Log

A project log is maintained and updated at the end of every sprint to keep us aware of what we have accomplished.

PROJECT LOG				
Sprint 1: Week 4 - Week 5				
S/N	**Task Overview**	**FR**	**NFR**	**Remarks**
1	First meeting and agenda setting	-	-	Met up and learnt about the project requirement
2	Design meeting	-	-	1. Completed rough sketch of frontend 2. Set the purpose and target audience of the project
3	Figma design	-	-	Completed figma design of all major frontend components
4	Discussed FRs and NFRs	-	-	Tabulated the FRs and NFRs to be used along the way

5	Met up with mentor to verify our FRs and NFRs	-	-	Refined our FRs and NFRs and added reasoning for them
6	Backend Design	-	-	Draw out overview of backend design and delineated services needed.
Sprint 2: Week 6 - Week 7				
S/N	**Task Overview**	**FR**	**NFR**	**Remarks**
1	Create User and Question services using Firebase and Supabase	FR1, FR4	NFR1, NFR3	Dockerised on creation
2	Setup frontend client	-	-	NextJS used
3	Complete frontend for /home	FR4	-	Displays all questions
4	Add user management page	FR1	-	-

5	Implement Authentication with Firebase	FR1.1, FR1.2	NFR1.2	Setup secure login and session management
6	Initialize Question Database in Supabase	FR4	NFR3.3	Configured to handle 2000 questions
7	Develop Question Filtering and Sorting	FR4.2	-	Implemented on frontend /home
8	Backend API for user profile management	FR1.3	NFR1.4	Developed endpoints for profile CRUD operations
9	Frontend Integration with Backend Services	FR1, FR4	NFR4.1	Ensured cross-browser compatibility
10	Set up kubernetes cluster on Minikube			Local testing to prepare for deploying on GCP

11	Mentor Review and Feedback Session	-	-	Discussed progress and received guidance
Sprint 3: Week 8 - Week 9				
S/N	**Task Overview**	**FR**	**NFR**	**Remarks**
1	Implement Matching Algorithm	FR2.1, FR2.2	NFR1.2	Setup to handle 1000 concurrent users
2	Real-Time Collaboration Setup	FR3.2	NFR2.3	Used Y-WebRTC for code editor
3	Deploy Video Call Functionality	FR2.3	NFR2.2	Integrated Agora for low-latency communication
4	Design Database Schema for Collaborations	FR3	NFR3.4	Stored in Supabase

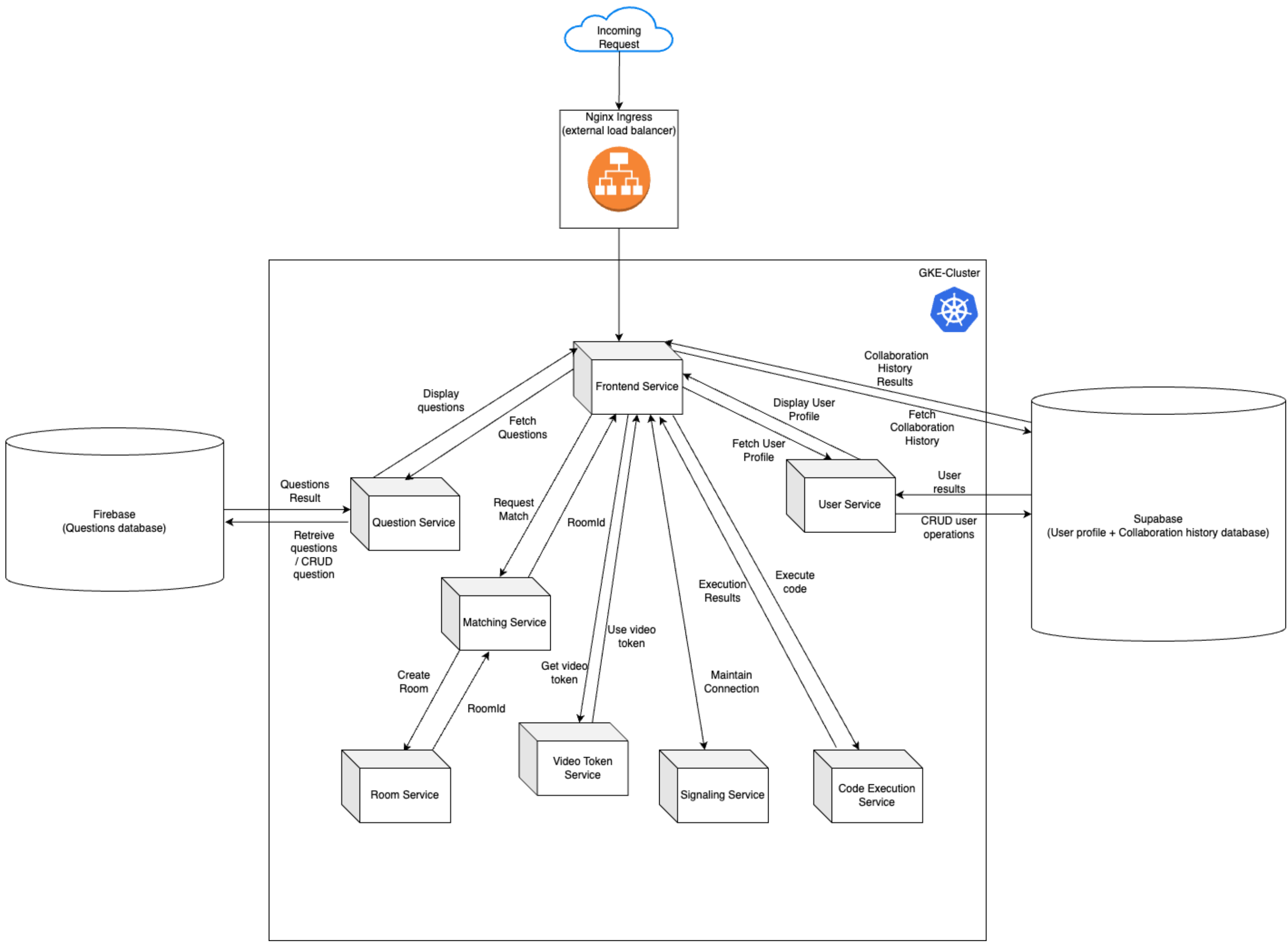
5	Add Authenticated Question Update Service	FR4.1, FR4.2	-	-
6	Frontend Implementation for Matching Service	FR2	-	Developed UI for difficulty selection
7	Backend Integration for Code Execution	FR3.4	NFR2.1	Linked with Judge0 API
8	Testing and Bug Fixes	-	NFR4.1	Conducted cross-browser tests by using 2 browsers
9	Set up CI/CD Pipeline	-	NFR5	Establish CI/CD Pipeline using triggers in Cloud Build to build images and push them to Artifact Registry.

10	Prepare for Mid-Project Review	-	-	Compiled all features for mentor review
Sprint 4: Week 10 - Week 11				
S/N	**Task Overview**	**FR**	**NFR**	**Remarks**
1	Add onboarding page	FR1	NFR1	Ensures first-time users properly set up their account details and language preferences
2	Add ability to add display pictures for users	FR1	NFR1.2	Supported file formats and size limit implemented
3	Optimize Matching Efficiency	FR2.2	NFR1.2	-
4	Enhance Collaborative Editor Experience	FR3.2	NFR2.3	CodeMirror optimizations for real-time collaboration

5	Implement Light/Dark Mode Toggle	-	NFR4.2	Frontend feature for user preference
6	Develop Error Handling Mechanisms	-	NFR4.3	Implemented user-friendly pop-ups for errors
7	Conduct Integration Tests	-	NFR5.1	Ensured proper functioning across development and production environments
9	Integrate Secret Manager into CI/CD Pipeline	-	NFR5.3	Environment variables are added as secrets and used for testing
8	Integrate Google Kubernetes Engine into CI/CD Pipeline	-	NFR5	Use triggers in Cloud Build to update Google Kubernetes Engine(GKE) deployments when images are pushed

Sprint 5: Week 12 - Week 13				
S/N	**Task Overview**	**FR**	**NFR**	**Remarks**
1	Finalize UI/UX Enhancements	-	NFR4.1, NFR4.2	Refined user interfaces for better accessibility
2	Implement Collaborative Session History	FR3, FR4.3	NFR2, NFR3	
3	Finalise CI/CD	-	NFR5	
4	Finalise Deployment		NFR5, NFR 6	

System Overview



Our microservices architecture hosted on Google Kubernetes Platform. For details of each service, please read the relevant sections.

Explanation on Choice of Core Technology

Here is an overview of core technologies that we used. For specific technologies / detailed explanation, they will be included in the relevant components and explained where necessary.

Next JS

Next.js is an open-source React front-end development web framework that enables functionalities such as server-side rendering and generating static websites for React-based web applications. This enhances performance and improves Server-Side Rendering (SSR), aligning with our non-functional requirement (NFR) of ensuring a responsive and swiftly loading interface. It also offers built-in routing and an easy-to-use page-based structure, which simplifies the development of the Home View and Settings View, directly contributing to fulfilling the requirement (NFR4.1) of providing a bug-free user interface across various browsers.

Typescript

TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale. It offers static typing, which is an essential feature that helps prevent many potential runtime errors. This is crucial for maintaining the system's robustness and reliability , especially when managing the complex state and props within the Matching View and Collaboration Room components. TypeScript's type system is particularly beneficial in achieving scalability requirements and enhancing maintainability.

Y-WebRTC

Y-WebRTC is a framework that facilitates collaborative features in web applications, particularly real-time bidirectional event-based communication. In our Peerprep platform, Y-WebRTC underpins the interactive code editor, allowing paired users to engage in simultaneous code editing—crucial for achieving Functional Requirement FR3.2 (Both users can edit the code simultaneously with changes reflected in real-time). By employing Y-WebRTC, we also meet Non-functional Requirement NFR2.3, ensuring that both users can interact within the same code space concurrently, which is vital for emulating the collaborative nuances of a technical interview.

Supabase

Supabase acts as a backend-as-a-service, offering an alternative to Firebase with a Postgres database foundation. It enables us to manage our user profiles and authentication with ease, addressing Functional Requirements FR1.1 to FR1.3, which cover account creation, login, and profile updates. Furthermore, Supabase's time stamp capabilities align with Non-functional Requirement NFR3.4, ensuring timestamps are accurately stored and synchronized with Singapore Time (GMT+8). Its robustness and scalability also contribute to meeting the system's usability standards, specifically NFR1.2.

Firebase

Firebase provides a comprehensive suite of tools that are instrumental in managing user data and interactions, aligning with the Functional Requirements related to user management and collaboration spaces. By leveraging Firebase's NoSQL database solutions, we ensure that the system adheres to Non-functional Requirements such as NFR3.3, where the database must handle a substantial number of questions.

Agora

Agora is strategically selected to empower our web application with advanced real-time communication capabilities. It is vital to our objective of creating authentic interview simulations (FR2.3), allowing users to engage in interactive sessions that closely mimic the dynamics of a live technical interview. With Agora's service, Peerprep can offer a robust video and voice call experience that aligns with our Non-functional Requirement NFR2.2, which mandates less than 200ms latency to ensure smooth user interactions. Agora's impressive uptime of 99.99% and global latency of around 400ms contribute to a reliable platform that facilitates uninterrupted communication, thereby supporting the collaborative nature of technical interviews and enhancing the overall user experience.

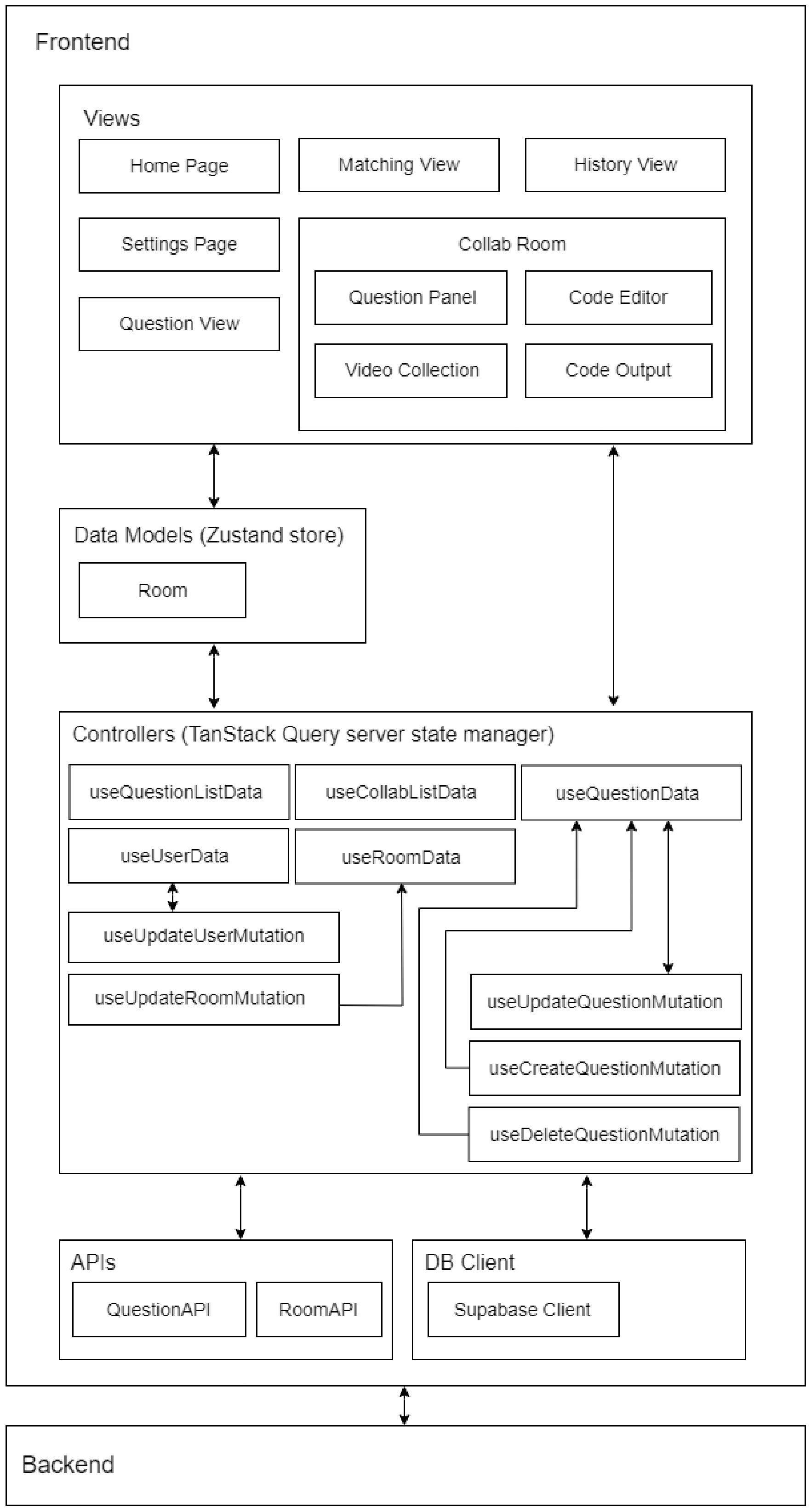
Code Mirror

CodeMirror is integrated within Peerprep to fulfill our commitment to providing a collaborative coding environment (FR3.2). This versatile code editor enriches the collaborative space with features such as code formatting and syntax highlighting for multiple programming languages, addressing the Nice-to-Have Feature N5. By utilizing CodeMirror, we ensure that our users have access to an intuitive and responsive interface for writing and reviewing code, which is essential for real-time collaboration (NFR2.3) and supports the preparation of users to handle actual coding interviews effectively.

Judge0

Incorporating Judge0 into Peerprep serves a dual purpose: it not only allows users to execute code within our platform (FR3.4) but also scales efficiently to handle a high volume of submissions, which directly addresses Non-functional Requirement NFR1.2 that supports up to 1000 concurrent users. Judge0's REST API facilitates the seamless execution of users' code, providing an isolated environment that aligns with security standards. This integration is critical for offering an end-to-end coding experience where users can write, execute, and evaluate code—all within the collaborative space of Peerprep, thereby enhancing the platform's educational value and user engagement.

Frontend Components



The provided diagram offers a comprehensive view of our frontend (client) architecture. While our overall frontend architecture adheres to the Model-View-Controller (MVC) design pattern, the specific technology stack we've opted for, involving React, Zustand and TanStack Query, introduces a nuanced relationship between the view and controller aspects. For example, a React component may contain both the view as well as some logic that goes behind it (e.g. clicking buttons and inputting text).

Auth, middleware

The authentication (Auth) and middleware components play a crucial role in managing user authentication and handling middleware functionalities. These elements ensure secure and authenticated access to the application, safeguarding user data and interactions.

Feature	Description	FR/NFR
User Authentication	Ensures secure login and session management	FR1
Middleware Security	Manages API calls and token validation	NFR1

Matching View

The Matching View is designed to facilitate the matching process within the application. It uses a range slider in the user interface to pair users based on difficulty range selected, providing a streamlined and efficient way for users to match with other users and begin the collaboration process.

Feature	Description	FR/NFR
Difficulty Level Matching	Matches users based on selected difficulty levels	FR2

History View

The History View is responsible for presenting a user's historical sessions within the application. This includes the user that they matched with, the time matched, the duration of the session as well as the programming language used. This feature allows users to review their past collaboration sessions so that they can improve their technical skills.

Feature	Description	FR/NFR
Session History Display	Lists past collaboration sessions with details	FR5

Collaboration Room

The Collaboration Room is a central element that fosters real-time collaboration among 2 users. It involves features like video call, collaborative code editing, output from running the code, and interviewer notes. This allows 2 users to take turns being the interviewee and interviewer, which allows them to practice their technical interview skills.

Feature	Description	FR/NFR
Real-Time Collaboration	Enables interactive coding and video calling	FR3.2, FR3.5
Live Code Execution	Displays output from running the code	FR3.4

Home Page

The Home Page serves as the main landing page for users. It provides an overview of all the questions in the question bank.

Settings Page

The Settings Page is dedicated to user preferences and settings. Users can customize their experience through their avatar, preferred programming language and also customize their profile (e.g. name and username). The design ensures clarity and ease of navigation for users managing their preferences.

Question View

The Question View focuses on presenting and editing questions within the application. The design aims to optimize the user experience for engaging with the content effectively as we use Markdown and KaTex renderer to allow maintainers to style the question description effectively with code blocks, headings, lists, and emphasis, enhancing readability. Additionally, incorporating KaTeX syntax enables the inclusion of mathematical equations which can be necessary for some types of questions.

Feature	Description	FR/NFR
Content Formatting	Offers Markdown and KaTeX support for question curation	NFR3.1

Backend Components

Matching Service

Overview

The Matching Service is in charge of matching users based on the difficulty level they have chosen. There are 3 difficulty levels Easy, Medium, Hard which correspond to the difficulty of the questions. Users can choose to work on questions over a range of difficulties and match with others who want to do the same difficulty of questions as them, for example, User 1 {Easy, Medium, Hard} can match with User 2 {Medium}.

Design

Unlike the other microservices that uses Axios requests, we chose to use websockets due to the following benefits it provides.

1. Persistent Connection: WebSockets provide a persistent connection between the client and the server, which is essential for real-time applications like a user matching service. This persistent connection allows the server to push updates to the client immediately without the client having to make an Axios request.

2. Real-Time Communication: In a user matching system, we want to notify users of a match as soon as it happens. WebSockets excel at this because they enable real-time bi-directional communication. Axios, being based on HTTP, is designed for a request-response model, which is not ideal for scenarios where the server needs to initiate communication.
3. Efficiency: A WebSocket connection is more efficient for real-time applications that require frequent messages. With Axios, every HTTP request involves a handshake and header metadata, which can introduce latency and overhead, especially if you need to check repeatedly for timeouts or updates.
4. Stateful Interaction: Matching users based on difficulty level and managing timeouts inherently requires maintaining state over multiple interactions. WebSockets are well-suited for stateful interactions because the connection remains open, and the state can be tracked easily. In contrast, HTTP requests via Axios are stateless, so additional mechanisms would be needed to maintain state over multiple requests.
5. Timeout Management: Managing timeouts is more straightforward with WebSockets because the server can continuously monitor active connections and enforce timeout rules directly. With Axios, this would require the client to send periodic requests to the server to check the status, which is less efficient and could lead to delays in detecting timeouts.

We also used a doubly linked list so we can remove users in O(1) time with the reference to the Node they are stored in without having to traverse the list. This is particularly useful as our matching system has a timeout feature, allowing us to efficiently remove users who have timed-out. This also accounts for when users choose to cancel their match request, as it allows for quick removal of users from anywhere in the list.

FR/NFR fulfilled	
F2.1 Users can select the difficulty of the question(Easy, Medium, Hard).	We allow users to choose from Easy, Medium, Hard. They can choose a single difficulty or a range.
F2.2 Users should be matched with other	By having 3 queues (Easy, Medium, Hard), we ensure

users who select the same difficulty of the question.	that they match with others in the same queue who chose the same difficulty.
NFR1.1 A timeout period of 30 seconds should be placed in the system.	If a user stays in the matching queue for 30s which is the timeout threshold, their corresponding socket sends a timeout signal to the server, which then triggers the removal of the user from the matchmaking queue.
NFR1.2 It should be able to handle 200 concurrent users queuing.	Our system utilizes a doubly linked list for efficient user queue management, with WebSocket connections ensuring immediate updates. GKE automatically scales up pod deployment to maintain performance during up to 200 simultaneous user connections. This number can easily be increased with more budget to deploy more pods on more nodes across multiple regions.

User Service

Overview

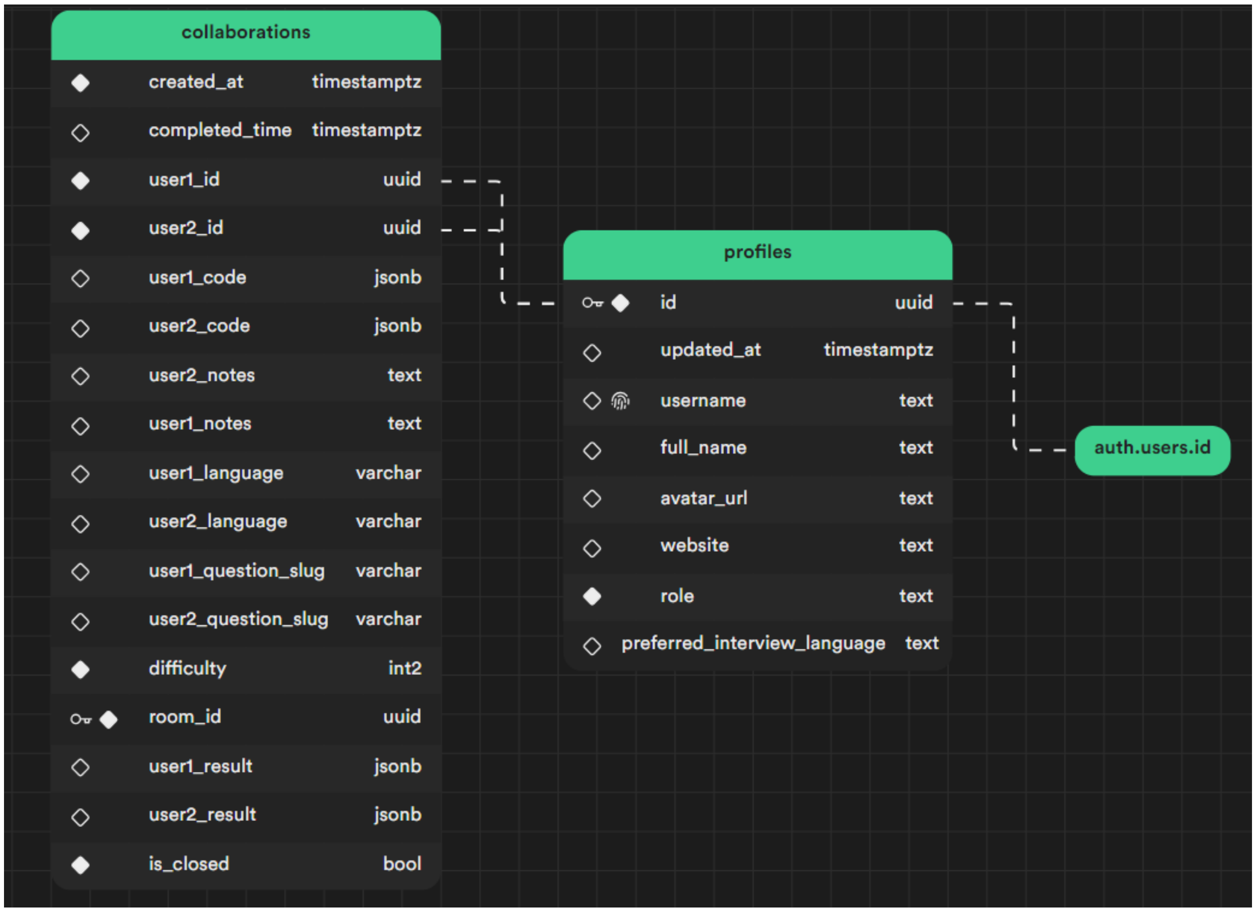
The User Service primarily handles user profile management and stores user collaboration records. It utilizes PostgreSQL (PSQL), a relational database system, to efficiently manage the relationships between user profiles and collaborations. This relational model significantly enhances the performance of joint operations and contributes to the system's scalability.

Database Design and Schema

The user service is responsible for user profiles management as well as storing user collaborations. The data base used is PSQL since the data stored (user profiles and collaborations between users) are relational in nature, using a SQL DB would therefore increase performance of joint operations between collaborations and profiles table, hence increasing scalability.

Referring to the Schema diagram below:

1. UUIDs are used for identifying users, which enhances the security and uniqueness of user profiles.
2. The profiles table includes essential attributes like username, full name, avatar URL, and roles, which are critical for profile management functionality.
3. The collaborations table holds data pertinent to user interactions, such as code contributions, notes, and the difficulty level of questions tackled together.



Database Schema

In addition, triggers are also added to the "collaborations" table to ensure that the database will store the latest edited datetime for the attempt on the question by the user. **This fulfills FR 1.4** .

<ins>APIs</ins>

The User Service supports the following RESTful API endpoints:

FR / NFR fulfilled	API
F1.1 Account can be created by users.	POST /user
F1.2 Users access the service by logging into the account.	GET /user
F1.3 Users can update their profile details.	DELETE /user UPDATE /user

The APIs have specific CORS policies that prevent unintended access by malicious parties. This ensures that only the services that are supposed to access the APIs can access them.

Question Service

<ins>Overview</ins>

The Question Service primarily handles maintaining a question repository indexed by difficulty level and specific topics. It utilizes a NoSQL database from Firebase, document-oriented database. This use of unstructured data enables the Question Service to handle large volumes of data at high speed with a scale-out architecture that contributes to the system's scalability.

<ins>Collection Design</ins>

The database used is a NOSQL database since the data stored (question data) are not relational in nature. Hence using a NOSQL without any join operations will allow better performance in retireval and storage.

The rough structure of the collection is shown below:

Categories	Stores an array of topics that the question falls under
Description	Stores a string containing the markdown of the question description
Difficulty	Stores a number (1-3) that corresponds to the questions difficulty
Link	Stores the link to the original question where this is scrapped from
Slug	Stores the question slug, to be used by other services
Title	Stores the title of the question

The existence of difficulty helps to **fulfill FR4.2** as they support the filtering of questions based on difficulty.

APIs:

The Question Service supports the following RESTful API endpoints that are protected via a middleware:

FR / NFR fulfilled	API	Admin Only
F4.1 Users of higher role level can add, delete and update questions.	PUT /:uuid POST / DELETE /:uuid	Yes
FR 4.3 All users can view the questions	GET / GET /:slug	No

The middleware requires an access token which prevent unintended access by malicious parties.

This ensures that only authorized users that are supposed to access the APIs can access them.

Further the middleware checks if the user is an admin before authorizing them to call admin only API routes.

Video Token Service (and signaling server)

<u>Overview</u>

The Video Token Service is a critical component that facilitates secure and authenticated access to video collaboration sessions. It plays two roles: it manages the authentication of users entering a video call within a collaboration room, and it serves as a checkpoint to ensure the signaling server's availability, which is vital for initiating peer-to-peer connections.

<u>Authentication of users</u>

Upon finding a match and joining a room, the frontend makes an API call to the Video Token Service to authenticate the user, which generates and returns a unique token specific to the current session. The frontend uses the generated token to connect to the Agora Server, which is a third party provider, to join the video call.

<u>Signaling server</u>

The signaling server manages peer-to-peer connections vital for PeerPrep's collaborative environment. As a hub for WebRTC's signaling data, it manages crucial information for direct communication setup. Imagine it as a digital operator, ensuring users can connect seamlessly.

For state persistence, Yjs's decentralized model syncs data in real-time across browsers, crucial for our document state continuity. Our system sends snapshots to an API for persistence, ensuring document availability even if one of the users' session ends. If a user is first to access a document, our hybrid approach tries fetching the state from peers or, if needed, from the API, preserving the document's state reliably.

FR/NFR	Description	Fulfilled by
FR3.5	Users should be able to communicate with one another	Video Token Service, Signaling Server
NFR2.2	The latency should be less than 200 ms	Signaling Server
NFR5	CI/CD isolation of production from development	Video Token Service, Signaling Server

Code executor

The code executor serves as the backbone of PeerPrep's interactive coding environment, offering users the capability to run their solutions in a controlled, scalable setting. Our integration with Judge0 allows users to validate their answers to interview questions by executing code directly within the application.

Supported Languages: The code executor proficiently handles Python 3, JavaScript, and Java, which are languages a year 1 computing student would have been exposed to.

Execution Trigger: Within the collaborative code editor in the frontend client, users can test their code by clicking "Run Code," which sends their script to Judge0.
Scalability: Judge0 runs within a Docker container on our server, ensuring resource-efficient scalability. It dynamically allocates runners according to the number of concurrent requests, guaranteeing prompt feedback even during high usage periods.

Execution Feedback: The service returns comprehensive results, including output, execution time, memory usage, and status codes. This detailed feedback helps users optimize their solutions and understand their code's performance under test conditions.

Feature	**Description**	**FR/NFR**
Code Execution	Allows execution of code within the app	FR3.4
Language Support	Supports Python 3, JavaScript, and Java	NFR2.4
Detailed Feedback	Provides output, time, memory, and status for each execution	FR3.4

Room service

The room service provides services regarding the collaboration room.

FR / NFR fulfilled	API	Admin Only
F3.1 Both users can see the question.	POST /create GET /:roomId	No
N2: History: Maintain a record of the questions attempted by the user e.g., maintain a list of questions attempted along with the date-time of attempt, the attempt itself and/or suggested solutions.	GET /	No

The middleware requires an access token which protects the above endpoints and prevents unintended access by malicious parties. This ensures that only authorized users that are supposed to access the APIs can access them.

Upon creation of a room after a match has been found, a post request with the initial details of the room is sent to the room service. A random question was also selected for each user based on the difficulty chosen and sent back to the frontend, which would be displayed in the collaboration room.

The /:roomId endpoint is to get the details of both the users that are assigned to collaborate in the room with that particular roomId. This would help the frontend display the correct details in the collaboration room.

The GET / endpoint would fetch all the past collaborations that the user did and display them in a table. The user can also click in to view details of the past collaboration if the user wishes to review the interview process as well as read the interviewer comments.

Testing, CI/CD, Deployment

Testing

Jest is used for testing the backend services. For local testing, “npm test” is run, while for testing for production is done in the Cloud Build scripts. The environment variables required for testing are formed using Secret Manager. Afterwards the environment file originally created for testing will be removed before the image is pushed to the Artifact Registry. This fulfills NFR5.3 and NFR5.2.

FR / NFR Fulfilled		
NFR5.2 Secrets should be well hidden from users in production environment	N9	When deployed to production, malicious users will try to figure out environment variables. Thus it should be protected to prevent leaks.
NFR5.3 Testing should be included in the pipeline	N8	Testing is crucial as the development environment may be different from the production environment.

		Furthermore, it is to prevent regression.

The following are the test cases for the backend services:

- ❖ Question service:
  - ➢ Interaction with Firebase
    1. GET /questions should return a list of questions
    2. POST /questions should add a question with title “Test Question 1”
    3. POST /questions with duplicate title should return an error
    4. POST /questions should add a question with title “Test Question 2”
    5. PUT /questions should fail to update a question into one where the title already exists
    6. PUT /questions should update a question as long as no other question with the same title exists
    7. DELETE /questions should delete questionData
    8. DELETE /questions should delete secondQuestionData
  - ➢ Middleware:
    1. All API request should be rejected due to no authorization header
    2. POST /questions should receive 401 response due to no authorization header
    3. PUT /questions should receive 401 response due to no authorization header
    4. DELETE /questions should receive 401 response due to no authorization header
- ❖ Matching service:
  - ➢ Matching Function:
    1. Queue 2 users that match difficulty. Second user joins after 29 seconds. Expect the message "Match found, timeouts cleared, room created."

ii. Queue 1 user and timeout before 30 seconds[Extra time(2 more extra seconds) allocated for test to complete for messages to appear after dequeuing]
iii. Queue 2 users but different difficulty. Expect "Removing User" message”
iv. Queue user but DISCONNECT after 10 seconds. Expect "Socket socketId has disconnected."
v. Queue user but REQ_STOP_FINDING_PAIR after 10 seconds. Expect "Socket socketId has disconnected."
vi. Queue the same user twice within 10 seconds after the initial queue. Expect "User already in queue."
vii. Queue the same user twice. Time to requeue is after the timeout time of 30 seconds. Expect "Removing user from difficulty queue"

❖ User service:
  ➢ CORS middleware
    i. should not allow CORS with an incorrect origin
    ii. should allow CORS with the correct origin

Deployment on Google Cloud Platform (GCP)

We deployed our microservices on Google Kubernetes Engine (GKE) cluster to ensure high availability and scalability. For the purpose of our demonstration and to manage costs, we initially configured our cluster within a single zone, operating on a three-node setup. However, to enhance our system's scalability and accommodate a higher volume of user requests, we have the flexibility to expand our node pools and extend our deployment across multiple zones.

We also set up an Nginx API gateway as an external load balancer to manage and route incoming traffic, providing a secure point of entry for the services and preventing direct external access to our services in our cluster. We then used Kubernetes' built-in DNS service for intra-cluster service-to-service communication by leveraging its automatic service discovery capabilities. The use of Nginx also enabled us to use peerprep.com to access our services hosted on the cloud.

Our GKE cluster is designed for high availability and is capable of deploying across multiple regions. Currently, we only deploy our services in a single zone to save costs and as we are simply demonstrating our application. However, the cluster can easily be deployed across multiple zones, ensuring service replication across several nodes to provide redundancy and quick recovery from node failures.

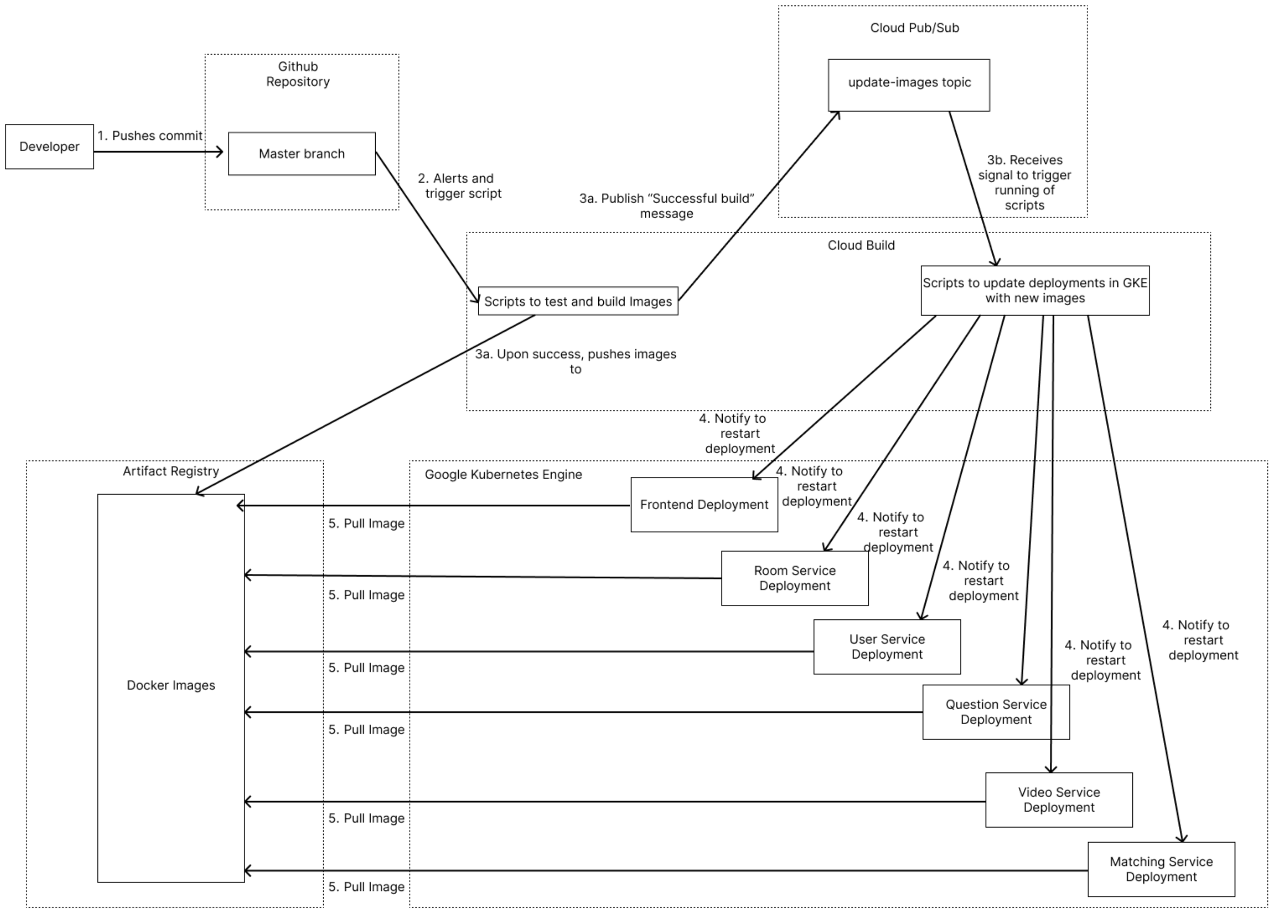
In terms of scalability, we employed a Horizontal Pod Autoscaler for each microservice, enabling automatic scaling based on CPU usage to efficiently manage load variations. Our use of Kubernetes' built-in service discovery and nginx ingress load balancer supports seamless inter-service communication. For monitoring and logging, we've integrated with Google's operations suite to capture pod logs and set up alerts for quick anomaly detection and resolution, ensuring operational health and service reliability.

FR / NFR Fulfilled		
NFR 6.1 The Kubernetes	N9	By deploying on GKE, we can leverage its global

cluster must ensure 99% uptime, with the ability to automatically replace failed pods to maintain service continuity.		infrastructure to replicate our pods across multiple regions ensuring high availability. This approach ensures that if one region experiences downtime, traffic can be redirected to the healthy pods in another region.
NFR 6.2 The Kubernetes cluster should automatically scale the number of pods up or down based on CPU or memory utilization thresholds, with the ability to scale to a minimum of 1 pod and maximum of 10 pods per service.	N10	We implemented Kubernetes' Horizontal Pod Autoscaler (HPA) to dynamically scale the number of pods in our service deployments. This scaling is triggered when the CPU utilization per pod surpasses the 70% threshold, allowing our system to efficiently manage increased traffic by provisioning additional resources. These pods will also be automatically scaled down when traffic reduces.
NFR 6.3 The application should have an API gateway to redirect requests to relevant microservices, preventing direct external access to services within the cluster	N11	We configured Nginx as an external load balancer within Google Cloud Platform (GCP) to orchestrate incoming traffic. This setup ensures that requests are routed to the relevant microservices, while simultaneously preventing any direct external access to the individual services in the cluster..
NFR 6.4 The cluster must provide a service registry and discovery mechanism that allows services to discover and communicate with each other using service names instead of IP addresses.	N12	We leverage Kubernetes' built-in DNS and service networking capabilities to facilitate secure intra-cluster communication. This allows our services to discover and interact with each other via service names rather than relying on IP addresses. Such a configuration ensures that the services within our cluster can seamlessly communicate in a secure environment, shielded from direct external access.

NFR 6.5 The system must include comprehensive monitoring and alerting capabilities to track the health and performance of all services.		We utilize Google Cloud Platform's pod logging features to integrate comprehensive monitoring and alerting capabilities within our system. This enables us to closely track the health and performance metrics of all services, ensuring proactive oversight and maintenance of our service infrastructure.

CI/CD Pipeline



Developer
1. Pushes commit
Github Repository
Master branch
2. Alerts and trigger script
3a. Publish "Successful build" message
Cloud Pub/Sub
update-images topic
3b. Receives signal to trigger running of scripts
Cloud Build
Scripts to test and build Images
Scripts to update deployments in GKE with new images
3a. Upon success, pushes images to
4. Notify to restart deployment
Artifact Registry
Docker Images
Google Kubernetes Engine
Frontend Deployment
Room Service Deployment
User Service Deployment
Question Service Deployment
Video Service Deployment
Matching Service Deployment
5. Pull Image

Fig 2. Control Flow of CI/CD Pipeline

When a developer pushes a commit to the master branch, it will cause the trigger in Cloud Build to run the scripts to test and build the images from the updated source code in the master branch. The production image is built in the build scripts. Once a build is successful, the images will be pushed to the Artifact Registry. Simultaneously, a message will be published to the “update-images” topic in Cloud Pub/Sub. This will trigger Cloud Build to run the script to update the images of the deployments in Google Kubernetes Engine. Afterwards, the deployments will pull the updated images from Artifact Registry and run a restart.

FR / NFR Fulfilled		
NFR5.1 The production environment should be isolated from the development and staging environments	N9	Production environment is different from the development and staging environments. Unintended changes and data leaks can be prevented through this requirement

Possible Improvements and Enhancements

1. Incorporate test cases and check whether output of code execution passes the test case
2. Add suggested solutions for the questions. Users will be shown the solutions after they leave the room and show the comparisons between their solution and model solution (i.e. How long it took to run the model solution vs their solution)
3. Avoid using magic links as a login system. It causes difficulty in automated integration testing as users need to go to their email and click the link to be authenticated to use the services
4. Future enhancements could include support for additional programming languages, further broadening the platform's appeal.

5. Integrating a broader set of test cases for each question would provide users with more thorough validation of their code's correctness and efficiency.

Reflections

Being new to this technology stack, I see the similarity in the concepts used in the development process as compared to using Django which is what I am familiar with. However, I do also see the complexity in building the application and this translates to longer periods of time required to develop.One thing to note is that although more effort is required, the end product looks very appealing and furbished. Next, I learned that applying SoC when designing the application is crucial as it allows for scalability and independent development which means that the development process can be parallelised.This makes me further appreciate the microservice architecture, allowing new services to be hooked up without affecting the other services. Lastly, I realized the importance of implementing soft deadlines, as delays usually crop up and this results in rushed work which is a problem due to the abundance of bugs that require fixing.
This project has conferred me a deeper appreciation of the microservice architecture. The microservice architecture is not only great in ensuring scalability, it also enforces a clear delineation of functional domain. This helps to make the design process clearer and allows development of these services to be done in parallel. Developing collaborative services also allowed me to better appreciate Server Side Events and sockets more.
As I had no prior knowledge on Kubernetes, NGINX, and Google Cloud Platform (GCP), learning about cloud computing in this project has been challenging but rewarding. I learnt how to orchestrate microservices communication within a cloud-based architecture, ensuring cohesive operation. I also learned how to use WebSockets for live, two-way communication and Axios for sending stateless HTTP requests, allowing our services to communicate with

one another. During my journey of learning Kubernetes, I began with setting up a local cluster using Minikube, which laid the groundwork for understanding Kubernetes operations. This knowledge was crucial when I transitioned to deploying services on Google Kubernetes Engine (GKE), a platform that underscores high availability and robust container management. GKE's integration with cloud resources enabled us to deploy scalable applications. Configuring Nginx API gateway as an external load balancer on GCP also taught me how to secure our microservices by providing a single entry point for external traffic. It also facilitated the resolution of IP addresses to our domain name peerprep.com. Additionally, employing Kubernetes' in-built DNS service was pivotal for internal service discovery, enabling intra-cluster communication among our microservices without exposing them directly to the external network. Developing a serverless function to scrape APIs and populate our database with questions has taught me the value of automating data collection and integration within a cloud environment. I also set-up the Horizontal Pod Autoscaler (HPA) to enable responsive scaling, ensuring our infrastructure could dynamically adjust to the workload demands. Overall, these experiences have significantly honed my capabilities in creating scalable, resilient, and secure cloud-based infrastructures.
Throughout the project, we were given a lot of liberty to choose the tech stack to develop the project from ideation to deployment. From ideation, there was enough leeway given to create a product that we felt was important and this motivated us to develop something that we were passionate about. As the project was more open-ended, we were not bound by exactly what to accomplish but we could choose features depending on their suitability to the project's target audience. Moreover, this opportunity to choose technologies that were modern as well as apply software engineering concepts was amazing. For example, it was the first time I used a microservice architecture and this allowed me to better appreciate the strengths and

extensibility of a microservice architecture. In addition, this experience made me realize that while the product is important, creating a conducive developer experience is equally important. Having a good CI/CD is extremely helpful for developers so that we can focus more on delivering the best experience for the users.
Through this project, I've learned about the importance of microservices. Breaking down the application into smaller, independently deployable services not only made development and maintenance smoother but also allowed for targeted scalability of specific functionalities. The modular nature of microservices improved fault isolation, making it easier to identify and fix issues without affecting the entire system. This experience highlighted the strategic significance of microservices in promoting agility, rapid development, and effective responses to evolving business requirements. Despite its benefits, the project faced challenges and delays due to unforeseen complexities in integrating certain microservices and varying levels of tech familiarity among team members. Managing competing priorities, like urgent bug fixes and unexpected feature requests, required a shift in attention focus, emphasizing the need for flexibility and continuous communication. In conclusion, the project offered valuable insights into microservices' technical intricacies and the dynamics of managing a team with diverse technological backgrounds, emphasizing the importance of adaptability in software development projects.