

Project Report:



CS3219 Project 2023

Produced by Hannah Chia, Ren Weilin, Ng Jing Xuan, Ryan Peh, Markus Yeo

Table of Contents

Table of Contents.....	2
Chapter 1. Background and Project Context.....	4
Project Goals.....	4
Target Audience.....	4
Chapter 2. Requirements Specification.....	5
Project Scope.....	5
Feature Summary.....	5
Must-Have Functional Requirements.....	5
Nice-To-Have Functional Requirements.....	7
Non-Functional Requirements.....	8
Non-Technical Contributions.....	10
Chapter 3. Development Process.....	11
Waterfall Model and Timelines.....	11
Project Management.....	12
Formatting and Linting.....	13
GPT Development.....	14
Chapter 4. Application Design.....	15
4.1 Technology Stack.....	15
TypeScript.....	15
React.....	15
PostgreSQL.....	15
Prisma ORM.....	16
Redis.....	16
Node.js.....	16
WebSockets.....	17
4.2 Architecture.....	17
Frontend.....	17
Design Decision: Routing.....	19
Backend: Express Microservices.....	20
Matching Service.....	20
Interview Service.....	21
Question Service.....	21
User Service.....	21
Supabase service.....	21
Backend: Persistence.....	22
Redis (Data Store).....	22
PostgreSQL with Supabase.....	22

Activity Diagram.....	24
4.3 Deployment.....	25
Overview.....	25
Containerisation with Docker.....	26
Deployment with AWS ECS.....	27
Scalability.....	27
Service Discovery/Registry.....	28
4.4 Design Deep Dive: Matching and Interview Design.....	28
Matching Service.....	29
Interview Service.....	30
Code Collaboration.....	30
Chat Communication.....	31
4.5 Meeting the Non-Functional Requirements by Design.....	31
Performance.....	32
Scalability.....	32
Usability.....	32
Maintainability.....	33
Security.....	33
Reliability.....	33
Chapter 5. Future Enhancements.....	34
5.1 Enhanced Question Repository.....	34
5.2 Enhanced Collaboration Tools.....	34
5.3 User Interface and Experience.....	34
Chapter 6. Reflections and Learning Points.....	35
6.1 Project Learning Points.....	35
Microservice Architecture.....	35
Deployment Process.....	35
6.2 Teamwork Dynamics.....	35
6.3 Personal Reflection.....	36
Appendix 1.....	37
Question Service.....	37
User Service.....	37
Matching Service.....	39
Interview Service.....	39
UI.....	40
Deployment, Scalability.....	42

Chapter I. Background and Project Context

PeerPrep 🍬 is a fresh technical interview preparation platform designed to assist students in practicing whiteboard-style interview questions through a peer matching system. This platform facilitates a scenario where students can log in, choose their preferred question difficulty and topic, and be matched with peers for collaborative problem-solving.

Themed around candy, the app aims to inject a bit of levity and joy into the stressful interview prep process. You can find the application [at this link](#) to test it out for yourself.

Project Goals

The primary purpose of PeerPrep is to create a dynamic and collaborative environment for computing students to practice technical interview questions. Our first step in developing this project was deriving key objectives. We decided on:

- **Peer Collaboration:** Facilitate real-time matching of students with similar proficiency levels and interests, enabling them to practice interview questions together.
- **Skill Enhancement:** To provide a platform where students can enhance their coding and problem-solving skills through collaborative and competitive exercises.

Target Audience

The primary target audience for PeerPrep is computing students who are actively preparing for technical interviews. This includes undergraduates, postgraduates, and even self-learners who are seeking opportunities in the technology sector.

Chapter 2. Requirements Specification

In this section of the report, we delve into the foundational aspects of our application, focusing on the must-have requirements, non-functional requirements, and a detailed breakdown of functional requirements for additional features that, while not essential, would significantly enhance the user experience.

We also cover the contributions of our team members alongside the features and assess the impact of the features implemented and the requirements chosen. **Overall, all team members were essential to the completion of our project.**

Project Scope

PeerPrep is designed to encompass several key features, such as a user-friendly interface for account creation and login, a matching system for connecting peers, a question repository categorized by difficulty and topic, and a collaborative space for real-time problem-solving. The platform also incorporates modern technologies and microservice architecture to ensure a robust, scalable, and user-friendly experience. See [Appendix 1](#) for a full list of functional requirements developed and claims. Note that this project is tested on Chrome browsers and functionality is not guaranteed on other browsers.

Feature Summary

Must-Have Functional Requirements

We have implemented these must-have requirements for this project effectively and robustly.

Must-Have Requirement	Summary of key features requirements implemented for each requirement	Work allocation
M1: User Service (Application Design , FRs)	<ul style="list-style-type: none">• The ability for users to create, edit, and delete their accounts.• Secure authentication and authorization mechanisms.<ul style="list-style-type: none">◦ Access to certain functionality is restricted based on the user's role• Password recovery and reset functionalities.• Session management - handles page refresh, log out, log in	Hannah, Markus, Jing Xuan
M2: Matching	<ul style="list-style-type: none">• Algorithm to match users based on selected criteria	Markus,

Service (Application Design, FRs, Design Deep Dive)	(difficulty level, topic of interest). <ul style="list-style-type: none"> • Time-based timeout feature for unmatched requests, handling scenarios when no match is found • Option to re-enter the matching queue after a timeout. 	Ryan, Weilin
M3: Question Service (Application Design, FRs)	<ul style="list-style-type: none"> • A comprehensive database of interview questions is categorized by difficulty (easy, medium, hard) and topic. • Support the addition of new questions to the repository. <ul style="list-style-type: none"> ◦ Ability to import questions en masse • Scalable to thousands of questions; modular design 	Jing Xuan
M4: Collaboration Service (Design Deep Dive, Interview Microservice)	<ul style="list-style-type: none"> • Real-time code editor with collaborative features enabling simultaneous editing. 	Hannah, Markus, Ryan
M5: Basic UI (UI FRs)	<ul style="list-style-type: none"> • Routing and user view depend on role • Multiple pages for different purposes in the application <ul style="list-style-type: none"> ◦ Question viewing ◦ Matching ◦ Interview service ◦ User page ◦ Log-in page • Clean “candy” theme that supports both light and dark mode 	Markus, Weilin, Ryan
M6: Deployment (Application Design, FRs)	<ul style="list-style-type: none"> • Deploy the application in a local staging environment with Docker <ul style="list-style-type: none"> ◦ Services are containerised • `docker-compose up -d` to run all containers locally 	Weilin

Nice-To-Have Functional Requirements

For clarity on the nice-to-have features implemented, we have included this short table. For a more extensive breakdown of the feature requirements, see [Appendix 1](#).

Team Chocolate: Markus, Ryan, Jing Xuan

Team Toffee: Hannah, Weilin

Feature	Description and Requirements	Subteam
N1: Communication	<ul style="list-style-type: none">Real-time text-based chat in the collaborative spaceTransient for less pressure when chatting during the interview	Toffee
N2: History	<ul style="list-style-type: none">Maintain a record of questions attempted by users, including details like the date-time of the attempt and the attempt itselfAbility to view profile of users you worked with on your user page	Chocolate
N4: Question Service	<ul style="list-style-type: none">Enhance the question service to manage questions more effectively with categories like taggingMechanism to randomize question selection for matched peers.	Chocolate
N5: Interview Service	<ul style="list-style-type: none">Code formatting and syntax highlighting for an extensive number of languages	Chocolate
N9: Deployment	<ul style="list-style-type: none">Services deployed on AWS in containers	Toffee
N10: Scalability	<ul style="list-style-type: none">Kubernetes horizontal pod auto-scaler to scale up the number of application pods during high loadInclude load balancers & health checks	Toffee
N12: Service Registry	<ul style="list-style-type: none">Built-in DNS and service networking capabilities of Kubernetes within ECS	Toffee

Non-Functional Requirements

In crafting "PeerPrep," a collaborative platform for technical interview preparation, we carefully selected nonfunctional requirements considering tradeoffs. We arrived at the following choices that align with PeerPrep's goal: to offer an efficient and seamless platform for mastering technical interviews.

In the below table, we elaborate on the nonfunctional requirements and crosslink the titles to sections where we explain implementing these NFRs more clearly

NFR	Goal	Potential Measures
Performance	<ol style="list-style-type: none">1. Fast and responsive matching.2. Low latency in the collaborative coding environment.3. Efficient loading and rendering of user interface elements.	<ol style="list-style-type: none">1. Messages received within 1s2. Pages should load within 2s
Scalability <i>also see</i> (with Supabase) (with Kubernetes)	<ol style="list-style-type: none">1. Ability to handle a growing number of users and data without performance degradation.2. Scalable microservice architecture to facilitate future enhancements.	<ol style="list-style-type: none">1. Capacity for autoscaling
Usability	<ol style="list-style-type: none">1. User-friendly interface accessible to users with varying levels of technical proficiency.2. Light and dark mode3. Ability to set user preferences	<ol style="list-style-type: none">1. User satisfaction surveys2. Assess the average time users take to complete specific tasks
Maintainability	<ol style="list-style-type: none">1. Microservice architecture2. Typescript, Prisma ORM3. Code review process4. Automatic formatting and linting	<ol style="list-style-type: none">1. Use consistent code style2. Code review process so all developers stay up to date with the project

****Click on links for fuller elaboration***

As with all applications, it is not possible to focus on every functional requirement. We have decided to deprioritise the common requirement of security. While security is of course still important, as the subject matter of PeerPrep is interview prep for students, little confidential information is handled at all. Even the questions we have gathered are no secret. As a result,

there is little risk of the data being compromised, so security was not a major focus of the app. However, we have still taken care to implement basic security features, as can be seen in the section on [security](#).

We also similarly tried to implement [reliability](#).

Non-Technical Contributions

We would also like to highlight a few nontechnical contributions from members of our group related to this project.

Task	Description and significance	Contributors
Summarizing project Requirements	<ul style="list-style-type: none">Broke down extensive list of functional requirements and nonfunctional requirements by cross-referencing against assignment materials provided, helping to gauge total amount of work	Hannah
Project visual design	<ul style="list-style-type: none">Developed cohesive vision for frontend, including the color scheme and layout, for our first cut	Markus
Researching Technical Stack	<ul style="list-style-type: none">Debated differences between database providers for the needs of our projectResearched various messaging frameworksResearching deployment frameworks	Ryan, Weilin, Jing Xuan
Upskilling team with AI and other resources	<ul style="list-style-type: none">Taught team via medium articlesContributed to teams understanding and comfort level in using AI tools in development	Weilin
Project Planning	<ul style="list-style-type: none">Creating todo-lists, listing out deadlines helped to keep us on track with the project and plan accordingly	Markus, Hannah
Mentor Point of Contact	<ul style="list-style-type: none">Followed up with mentor regarding feedback and contributions on behalf of the group	Hannah
Pair programming	<ul style="list-style-type: none">Guide members with aspects of programming less familiar to them and provide detailed feedback	Ryan
QA Testing	<ul style="list-style-type: none">Stress test the app and walk through all user flows to identify bugsList different bugs	Ryan, Hannah

Chapter 3. Development Process

Waterfall Model and Timelines

The following is a timeline of how we organized and arranged the work for the project. To aid us in this timeline, our prioritization of requirements (see [Appendix 1](#)) was particularly relevant.

We adopted the **Waterfall Model** as it is excellent for projects with well-defined and stable requirements, where changes are less likely to occur. Its rigid structure also made it easy to measure progress and assess project completion. The waterfall model's systematic approach was advantageous for us as the overall project requirements were given to us and the development path was relatively predictable. This model helped us to remain disciplined throughout the development process and balance the work from CS3219 with the rest of our workload.

Feature	Timeline
Define requirements and explore tech stack (Summary , Full Appendix)	Week 5
M1: User Service	Week 6
M2: Matching Service	Week 7
M3: Question Service	Week 6
M4: Interview Service	Week 7
M5: Basic UI	First draft by end of Week 7, Revisions through Week 12
M6: Local Deployment	Week 9
N1: Communication	Week 8
N2: History	Week 9
N4: Question Service Enhancements	Week 9
N5: Interview Service Enhancements	Week 10
N9: Cloud Deployment	Week 12
N10: Scalability	Week 12
N12: Service Registry	Week 12

Alongside the feature timeline, we also guided ourselves through interim deadlines, including milestone meetings with our mentor. During these meetings with our mentor, Gaurav Gandhi, we were able to gain a sense of our project and refine our requirements and plans to meet our goals.

Milestones	Date	Details
Milestone 1	8 September	Mentor meeting: Review preliminary plan of functional requirements
Complete Requirements	15 September	Initial list of functional requirements submitted to mentor for review
Milestone 1.5	8 October	Complete all must-have features
Milestone 2	26th October 1-2 pm	Mentor meeting: Demo of all must-have features and the nice to have features communication, history, and question service enhancements
Hackathon	11th November	Full day planned to focus on ironing out tricky problems with the project, implement and test features
Feature Freeze	11th November	All new feature development is halted. All subsequent code changes involve UI improvements and bug fixing
Project Submission	15th November 5 pm	Conduct a final review of the project, ensuring all requirements are met, and perform a last round of testing to catch and resolve any lingering issues.

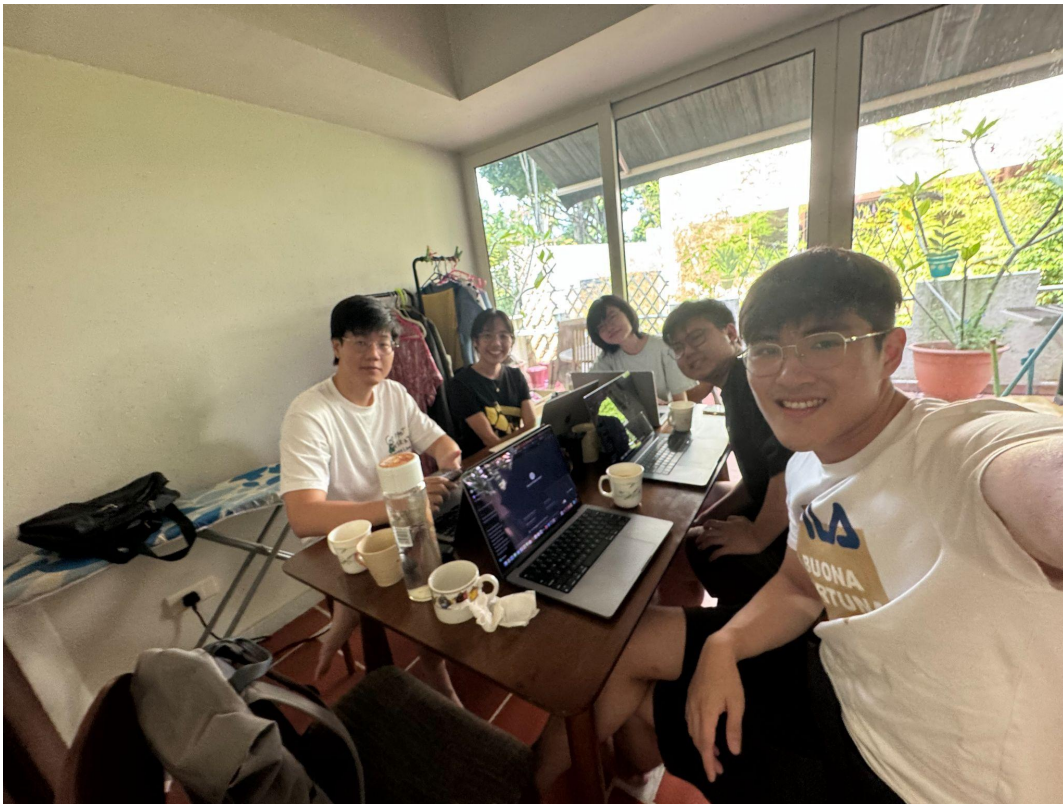
Project Management

Project management was done over Git. Git streamlines collaboration by allowing multiple developers to work on a project simultaneously with branching and merging. It provides a clear history of changes, aiding tracking and debugging, and facilitates easy rollback to previous versions

For our project, we worked independently on separate branches for different features, merging them into a central branch when the feature or main change was complete. Branches provide a clear and organized structure for development, making it easier to manage different tasks and track progress. Furthermore, the isolation reduced merge conflicts between members.

We established a group policy requiring code review from at least one other member. This was to reduce the chance that problematic and buggy code would be merged into main. It also encouraged us to maintain an overall understanding of the code base, where we understand the reasoning and rationale behind key technical decisions even if we were not the ones to implement it.

The five of us communicated asynchronously over Telegram for much of the project, followed by Zoom meetings at key milestones. We also strived to meet in person once a week to touch base, usually before tutorials or lectures. We also met up to work on the project for a “Hackathon” where we ironed out problems, implemented features, and tested the application. As seen below in the photograph, the collegial atmosphere and planning this time into our schedule benefited our project significantly and the overall experience of working on the project. We are pleased that, in addition to completing our project, we have also formed meaningful friendships. For more of our reflections, see [Section 6.2 Teamwork Dynamics](#).



Formatting and Linting

Our team strategically utilized automatic formatting tools, namely Prettier, and linting tools like ESLint, to enhance our code development project. Prettier helped us ensure a consistent and standardized coding style, reducing errors related to formatting discrepancies. It also made our code more readable. Simultaneously, ESLint in our development pipeline facilitated real-time analysis, enabling prompt identification and rectification of errors, stylistic incongruities, and deviations from coding standards. This approach cultivated a culture of cleaner and more maintainable code throughout the development process. These integrations were crucial in maintaining high-quality code standards and efficiency.

GPT Development

Incorporating GPT (Generative Pre-trained Transformer) technology into our development process marked a significant leap in our project's capabilities. The two tools we leveraged here were Github Copilot and OpenAI's ChatGPT.

We leveraged GPT for various tasks, including automated code generation, debugging natural language processing, and even drafting preliminary documentation. This not only sped up the development process but also reduced human errors, ensuring a higher code quality. Integrating AI-augmented our coding capabilities and opened up new avenues for innovation and efficiency in our project execution.

Chapter 4. Application Design

4.1 Technology Stack

In the development of PeerPrep, a careful selection of technologies was paramount to ensure **efficiency**, **scalability**, and a **seamless user experience**. The chosen stack includes [TypeScript](#), [React](#), [PostgreSQL](#), [Prisma ORM](#), [Redis](#), [Node.js](#), and [WebSockets](#). Each of these technologies plays a vital role in the project's success.

TypeScript

Rationale: TypeScript was chosen for its strong typing features, which enhance the [maintainability](#) of the code.

Advantages:

- **Error Detection:** Early detection of potential errors at compile-time, reducing runtime errors.
- **Improved Code Quality:** Enhanced readability and predictability of code, making it easier for team collaboration.
- **Integration with JavaScript:** TypeScript is a superset of JavaScript, allowing for seamless integration with existing JavaScript libraries.

React

Rationale: React's component-based architecture is ideal for building the dynamic and interactive user interfaces required for PeerPrep. Hence, we used this tech stack in our [frontend](#).

Advantages:

- **Reusable Components:** Facilitates the development of a modular UI, speeding up the development process.
- **Efficient Rendering:** React's virtual DOM improves application [performance](#), crucial for real-time collaboration features.
- **Rich Ecosystem:** Access to a wide range of libraries and tools, enhancing the application's capabilities.

PostgreSQL

Rationale: PostgreSQL is a robust and [scalable](#) database management system, suitable for handling complex data relationships in PeerPrep.

Advantages:

- **Data Integrity:** PostgreSQL enforces data integrity constraints, ensuring that data remains consistent and accurate. It supports various constraints like unique, primary key, foreign key, and check constraints, helping maintain data quality. There is also strong emphasis on ACID compliance ensuring reliable data management.
- **Advanced Features:** Supports advanced data types and efficient querying, beneficial for managing diverse data sets (like user profiles, questions, and session logs).

In this case, we used [Supabase](#), an open-source cloud-based platform which uses PostgreSQL for its real-time databases.

Prisma ORM

Rationale: Prisma ORM simplifies database workflows, enhancing the development speed and database maintainability. It allows us to generate performed SQL queries, and the additional abstraction aids [maintainability](#).

Advantages:

- **Easy Query Building:** Intuitive API for constructing queries, making database operations more manageable.
- **Type Safety:** Seamlessly integrates with TypeScript, ensuring type safety throughout the application.
- **Migration Management:** Simplifies database schema migrations, aiding in database evolution and maintenance.

Redis

Rationale: Redis is used as a high-[performance](#) data store, primarily for managing the user matching queue. Using Redis helps us achieve our NFR of performance.

Advantages:

- **Speed:** Exceptionally fast read/write operations, crucial for the real-time matching of users.
- **Scalability:** Easily scales to accommodate a growing number of users and matching requests.
- **Flexibility:** Supports a variety of data structures, allowing for versatile usage scenarios.

Node.js

Rationale: Node.js is suited for building [scalable](#) and high-[performance](#) backend services. Using Node.js helps to achieve our NFR of performance and [maintainability](#).

Advantages:

- **Non-blocking I/O:** Enhances the application's ability to handle concurrent requests, a necessity for the real-time aspects of PeerPrep.
- **Community and Ecosystem:** Large community support and a plethora of available packages.
- **Unified Language:** Using JavaScript on both frontend and backend streamlines development and helps make the application more maintainable

WebSockets

Rationale: WebSockets provide a full-duplex communication channel, essential for real-time collaboration and interaction in PeerPrep.

Advantages:

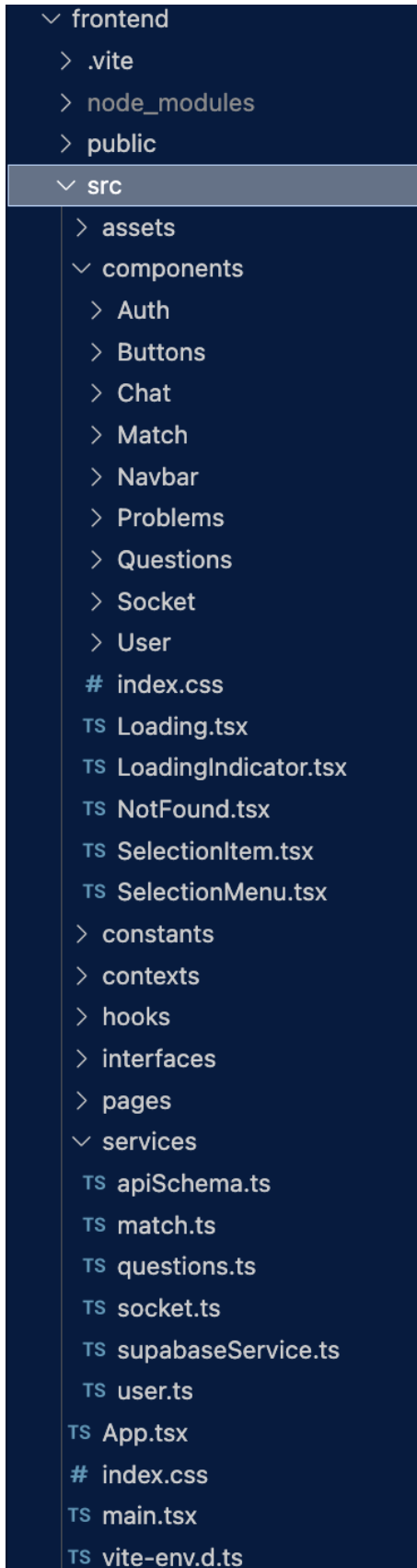
- **Real-Time Interaction:** Enables live code editing and instant messaging features, enhancing user engagement.
- **Persistent Connection:** Unlike HTTP, WebSockets maintain a continuous connection, reducing latency and overhead.
- **Bi-directional Communication:** Allows for simultaneous data exchange between clients and servers, crucial for synchronous collaborative activities.

4.2 Architecture

The overall architecture of PeerPrep is a thoughtfully designed hybrid model, combining the strengths of both microservices and a monolithic front end. This approach is chosen as it provides a strategic blend of the microservices' benefits with the cohesiveness of a monolithic front end, ensuring a scalable, maintainable, and efficient application infrastructure.

Frontend

The front end, while monolithic, employs React Components to ensure modularity and maintainability, focusing on user experience and responsive design. It acts as the primary interface for user interaction, integrating seamlessly with backend services for data retrieval and feature execution.



On the left, you can see the file tree of the frontend. To target our NFR of [maintainability](#), we made sure to have a clear folder structure and to separate our components out into manageable and readable pieces instead of having God classes.

Functionality: Provides the user interface for PeerPrep, handling user interactions, session creation, and information display. It is the primary interface for user engagement and interaction with the application

Design Considerations:

- **User Experience:** Focus on intuitive navigation and responsive design, as well as user personalisation.
- **Compatibility:** Ensure cross-browser and cross-device functionality.
- **Performance:** Optimize for fast load times and smooth interactions.

Interactions: Communicates with the User, Matching, Interview, Question, Interview, and Supabase Services for data and feature integration.

See [Usability](#) section for more on how our frontend achieves the Usability NFR.

Design Decision: Routing

Routing was implemented with `react-router-dom`. The following code comes from `App.tsx`, our parent component for the frontend of the app. The following code demonstrates two things, components and isolating routed routes

```
80      <MainNavigationBar />
81
82      <Box
83 >    flexGrow={1} ...
88    >
89      {!session && (
90        <Routes>
91          <Route path="/" element={<HomePage />} />
92          <Route path="/auth" element={<AuthPage />} />
93          <Route
94            path="*"
95            element={<Navigate to="/auth" replace />}
96          />
97        </Routes>
98      )}
99      {session && (
100        <Routes>
101          <Route path="/" element={<HomePage />} />
102          <Route path="/user" element={<UserPageMain />} />
103          <Route path="/user/:id" element={<UserProfilesPage />} />
104          <Route path="/questions" element={<QuestionsPage />} />
105          <Route path="/match" element={<MatchPage />} />
106          <Route path="/interview" element={<InterviewPage />} />
107          <Route path="*" element={<NotFound />} />
108        </Routes>
109      )}
110    </Box>
```

Firstly, the separate `<MainNavigationBar>` is a design decision that allows us to reuse the navigation header across all pages, giving users the functionality to easily navigate to certain pages on all pages while having the single source of truth for that component isolated in one place. When changes need to be made to update the navigation bar, only one component must be updated. We can see this modular design as each page is also a separate component: `HomePage`, `UserPageMain`, etc.

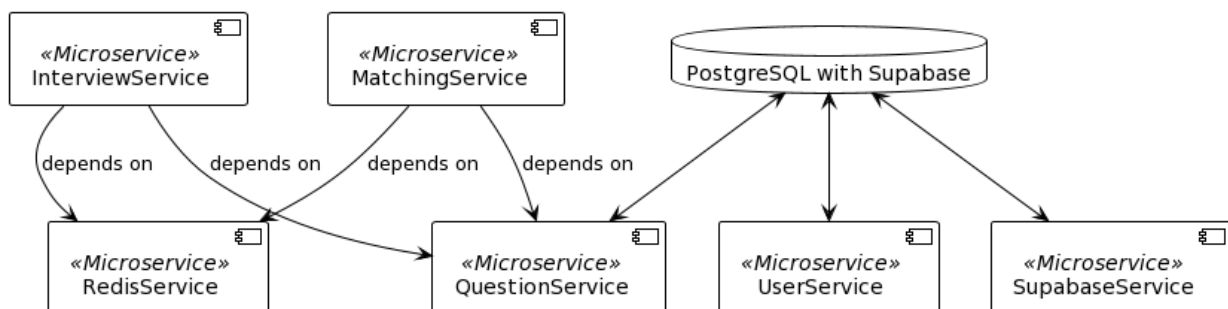
Secondly, this routing also helps us to achieve our goals of [usability](#) and the additional NFR of [security](#). Usability is achieved as navigating to the wrong route or a not available route either shows an informative `NotFound` page when logged in or redirects to the sign-in page to log in. Security is also achieved as without a logged-in session, anonymous users are not able to access the pages in the app as the routes are simply not available in the not logged-in rendering of the application.

Backend: Express Microservices

In developing PeerPrep, we adopted a microservice architecture to address several key requirements and objectives:

- **Scalability:** Microservices allow different parts of the application to scale independently, accommodating varying load patterns efficiently.
- **Flexibility in Development and Deployment:** This architecture enables the use of different technology stacks and independent deployment cycles for each service, enhancing development flexibility.
 - We deployed each service separately on AWS
- **Enhanced Maintainability:** Smaller, well-defined services are easier to understand, modify, and maintain.
- **Resilience:** Isolated services reduce the risk of a single point of failure, enhancing overall system reliability.

The services, (User, Matching, Interview, Question, and Supabase Services) are designed to interact cohesively. The frontend serves as the main user interaction point, while backend services handle specific functions. [Redis](#) assists in rapid data processing, crucial for features like user matching. [PostgreSQL with Supabase](#) underpins the entire architecture by providing robust and scalable data management.



The above diagram describes the dependency tree of the backend microservices. To understand how they interact with each other, refer to the section [Activity Diagram](#).

Matching Service

- **Functionality:** Pairs users for interview sessions based on selected criteria.
- **Design Considerations:**
 - **Algorithm Efficiency:** Develop a robust matching algorithm.
 - **Scalability:** Handle high volumes of match requests.
 - **Flexibility:** Adapt to evolving matching criteria and algorithms.
- **Role:** Essential for effective user pairing and session creation for interview service to retrieve.
- **Interactions:**

- Utilizes the QuestionService to retrieve a random question for the matched session based on difficulty.
- Utilizes Redis for queuing and communicating match results to the Frontend Service.

Interview Service

- **Functionality:** Manages the lifecycle of interview sessions.
- **Design Considerations:**
 - **Session Management:** Handle multiple concurrent sessions.
 - **Reliability:** Ensure stable and uninterrupted session performance.
 - **Integration:** Coordinate with the Interview Service for real-time features.
- **Role:** Core to facilitating the interview practice sessions.
- **Interactions:**
 - Coordinates session details with the Frontend
 - Coordinates real-time updates with the Interview Feature.

Question Service

- **Functionality:** Maintains a repository of interview questions.
- **Design Considerations:**
 - **Database Management:** Efficient storage and retrieval of a large question pool.
 - **Categorization:** Robust system for question categorization.
 - **Retrieval Speed:** Fast update and retrieval mechanisms.
- **Role:** Provides the content for interview practice.
- **Interactions:**
 - Delivers questions to the Frontend and Interview Services.
 - Supplies random questions to the Matching Service.

User Service

- **Functionality:** Manages user profiles, authentication, and data storage.
- **Design Considerations:**
 - **Security and Privacy:** Strong authentication and data encryption.
 - **Scalability:** Efficiently manage an increasing number of user profiles.
 - **Data Integrity:** Maintain consistent and accurate user data.
- **Role:** Central to managing user-related data for personalization.
- **Interactions:** Supports user operations in the Frontend Service.

Supabase service

- **Functionality:** Manages user deletion and password modification.
- **Design Considerations:**
 - **Security and Privacy:** Strong authentication, data encryption, and row-level access control.

- **Data Integrity:** Maintain
- **Role:** Allow users to delete their own accounts and modify their passwords. Also ensures that users cannot delete or modify other users' accounts.
- **Interactions:** Verify user identity and enable secure user operations.

Backend: Persistence

Data management is handled through Redis, used for *rapid, in-memory data processing*, and PostgreSQL with Supabase for *robust, scalable data storage*. This setup ensures optimal [performance](#) and [reliability](#), essential for the real-time functionalities of PeerPrep.

Redis (Data Store)

- **Functionality:** Acts as an in-memory data store for the user matching queue.
- **Design Considerations:**
 - **Performance:** High-speed data access for real-time matching.
 - **Reliability:** Data persistence and robustness.
 - **Scalability:** Accommodate growing data and user base.
- **Role:** Boosts the performance and efficiency of the Matching Service.
- **Interactions:** Mainly interacts with the Matching Service for rapid match request processing.

PostgreSQL with Supabase

- **Functionality:** Serves as the primary database, hosted on Supabase for robust data management, *scalability and performance*.
 - Supabase uses PostgreSQL, a powerful and well-established open-source relational database. PostgreSQL is known for its ability to handle large volumes of data and complex queries quickly. Hence, we are confident that Supabase can support our use-case
- **Design Considerations:**
 - **Data Management:** Efficient handling of diverse data types and structures.
 - **Scalability and Reliability:** Leverage Supabase's cloud hosting for scalable and reliable database services.
 - **Integration:** Seamless integration with all services for data storage and retrieval needs.
 - **Ease of Use:** Simplifies database management tasks, such as backups, scaling, and security, allowing the team to focus on application development.
- **Role:** Underpins the entire architecture with robust and scalable data management.
- **Interactions:** Interacts with *all* services, especially the User, Question, and Interview Services, for data storage and retrieval, providing a centralized data management solution.

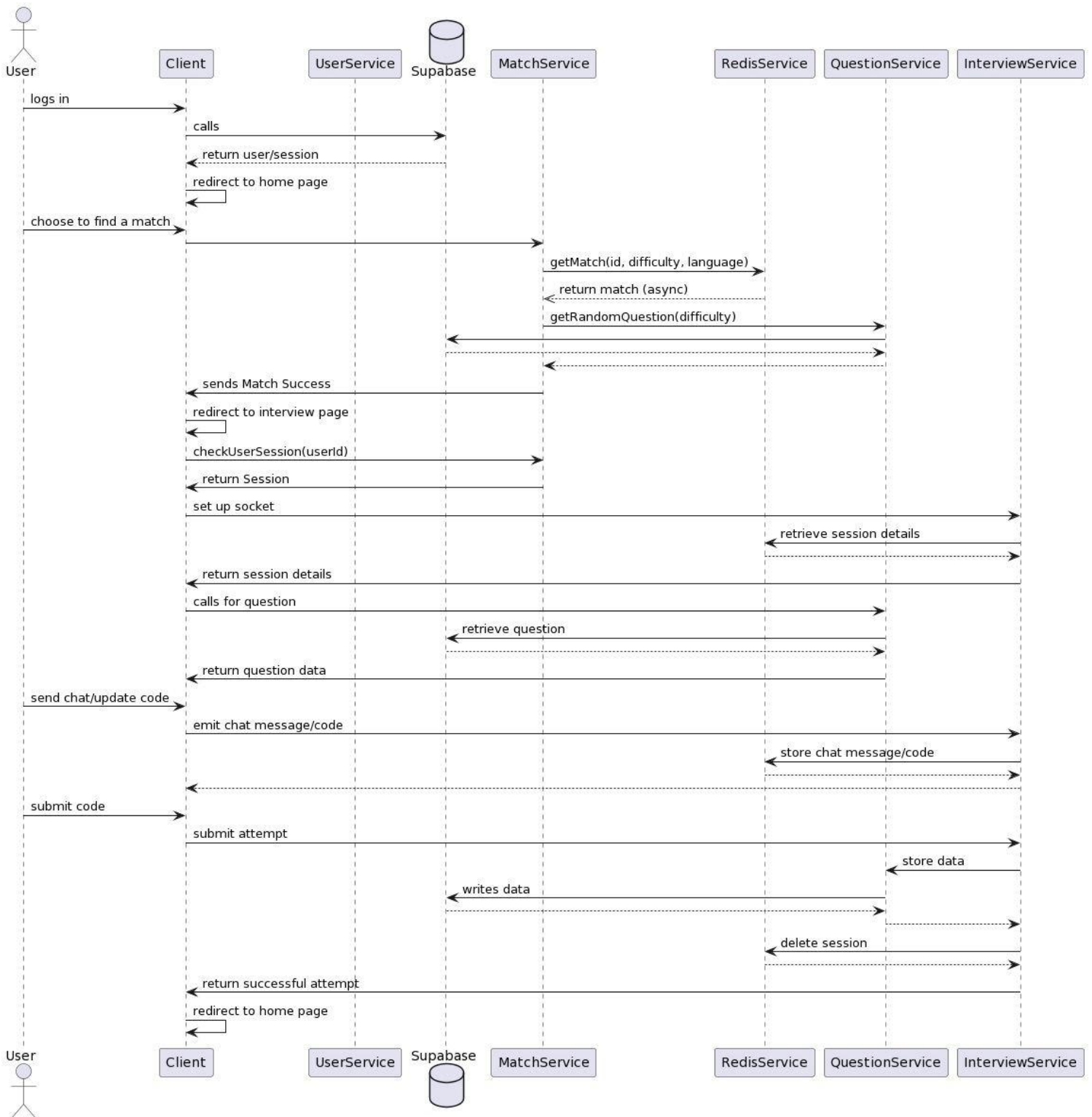
Aside from being easy to use, Supabase is particularly essential due to our goal of *scalability*. Supabase supports many enterprise uses with millions of connections and updates, so the cloud system is sure to be sufficient for our needs. Supabase is also reliable - [over the past 30 days, it has had 99.97 % uptime](#).

The below diagram demonstrates our relational schema in supabase. Using a relational database helps us maintain data integrity, enhancing the *maintainability* of the application. It also helps us avoid bugs regarding corrupted data, allowing our application to be more *reliable*.



Activity Diagram

To understand our happy flow works in more detail, we have drawn out this activity diagram:



4.3 Deployment

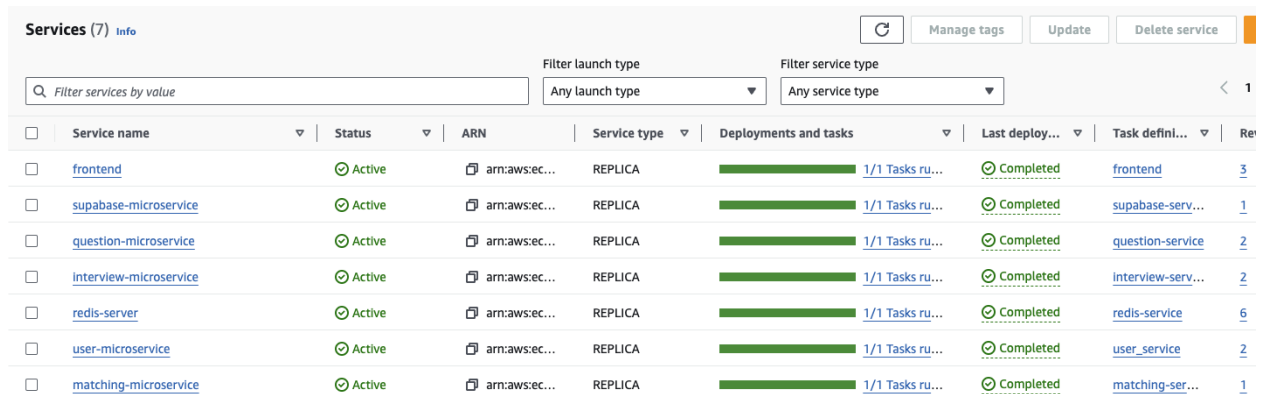
Overview

We have containerized all the components of our application and deployed it via Amazon Elastic Container Service.

AWS ECS (Elastic Container Service) was chosen as the deployment platform for our web application, which features a monolithic front end and a microservice backend, due to its robust capabilities and alignment with our architectural needs.

As one of the most familiar and popular cloud providers, AWS offers a reliable and widely recognised infrastructure, ensuring high availability and [scalability](#) for our application. ECS, being a fully managed container orchestration service, simplifies the process of deploying, managing, and scaling containerized applications. Its seamless integration with other AWS services enhances our application's functionality and allows us to leverage the extensive AWS ecosystem for monitoring, security, and automated deployments.

Moreover, ECS supports the microservices architecture natively, enabling us to manage each service independently while maintaining the cohesive operation of the entire application. This choice of deployment platform ensures that we can focus on developing and optimizing our application, while AWS handles the complexities of infrastructure management and scaling.



The screenshot displays the AWS ECS 'Services' page. At the top, there are buttons for 'Manage tags', 'Update', and 'Delete service'. Below these are filters for 'Filter launch type' (set to 'Any launch type') and 'Filter service type' (set to 'Any service type'). A search bar is also present. The main table lists seven services, all with a status of 'Active' and 'REPLICA' service type. Each service has a 'Deployments and tasks' column showing a progress bar and '1/1 Tasks running'. The 'Last deploy...' column shows a 'Completed' status with a green checkmark. The 'Task defini...' column provides a link to the task definition for each service.

Service name	Status	ARN	Service type	Deployments and tasks	Last deploy...	Task defini...	Re
frontend	Active	arn:aws:ecs...	REPLICA	1/1 Tasks ru...	Completed	frontend	3
supabase-microservice	Active	arn:aws:ecs...	REPLICA	1/1 Tasks ru...	Completed	supabase-serv...	1
question-microservice	Active	arn:aws:ecs...	REPLICA	1/1 Tasks ru...	Completed	question-service	2
interview-microservice	Active	arn:aws:ecs...	REPLICA	1/1 Tasks ru...	Completed	interview-serv...	2
redis-server	Active	arn:aws:ecs...	REPLICA	1/1 Tasks ru...	Completed	redis-service	6
user-microservice	Active	arn:aws:ecs...	REPLICA	1/1 Tasks ru...	Completed	user_service	2
matching-microservice	Active	arn:aws:ecs...	REPLICA	1/1 Tasks ru...	Completed	matching-ser...	1

Screenshot of deployed services on the ECS cluster

Containerisation with Docker

We will briefly elaborate on the process of containerisation with Docker.

- **Dockerization:** Each component of the PeerPrep application, including the Frontend, User, Matching, Question, Collaboration and Supabase services, was containerized using Docker. This involved creating Dockerfiles to define the container images.
- **Image Creation:** Docker images were created encapsulating the application code, dependencies, and environment settings, ensuring consistency across development, testing, and production environments.
- **Docker Registry:** Images were stored in a Docker registry for version control with tags and easy access for deployment. AWS Elastic Container Registry (ECR) was used for this purpose.

Private repositories (6)		
<input type="text" value="Find repositories"/>		
<input type="checkbox"/>	Repository name ▲	URI
<input type="checkbox"/>	frontend	954720214844.dkr.ecr.ap-southeast-1.amazonaws.com/frontend
<input type="checkbox"/>	interview-service	954720214844.dkr.ecr.ap-southeast-1.amazonaws.com/interview-service
<input type="checkbox"/>	matching-service	954720214844.dkr.ecr.ap-southeast-1.amazonaws.com/matching-service
<input type="checkbox"/>	question-service	954720214844.dkr.ecr.ap-southeast-1.amazonaws.com/question-service
<input type="checkbox"/>	supabase-service	954720214844.dkr.ecr.ap-southeast-1.amazonaws.com/supabase-service
<input type="checkbox"/>	user-service	954720214844.dkr.ecr.ap-southeast-1.amazonaws.com/user-service

Screenshot of Docker images pushed to AWS ECR

Deployment with AWS ECS

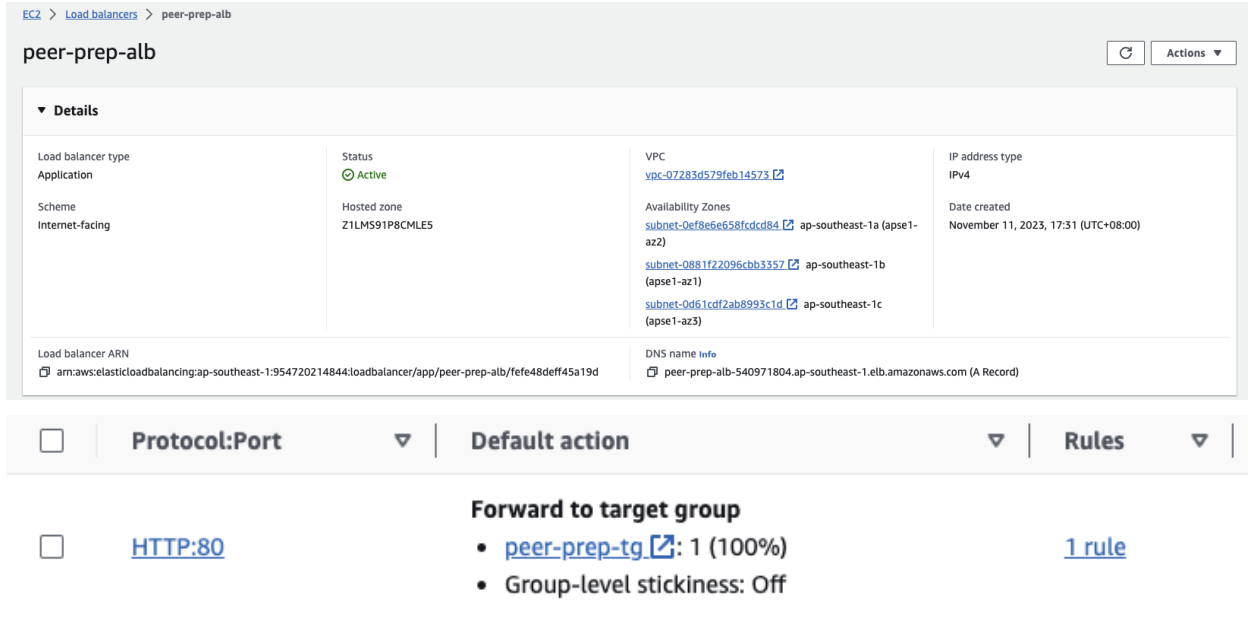
In this section we briefly describe the steps we took to deploy the services to the cloud.

1. **ECS Clusters with Kubernetes:** The application is deployed on AWS ECS clusters configured to use Kubernetes as the orchestrator. This integration leverages Kubernetes' robust orchestration capabilities, such as automated deployment, scaling, and management of containerized applications.
2. **Task Definitions:** ECS task definitions are used to specify the Kubernetes-managed Docker container images and configurations. These include resource requirements, environment variables, and logging configurations, ensuring that each service is optimally deployed and managed.
3. **Service Deployment:** Within these Kubernetes-enabled ECS clusters, services manage the ongoing operation of the application's containers, including the critical deployment of the backend microservices, redis and the front end. We have chosen to deploy each container as their individual service since none of them are tightly coupled and we want to be able to quickly iterate and redeploy microservices as individual components.

Scalability

As mentioned earlier, scalability is one of our key NFRs. While deploying, we specifically considered features of Kubernetes and ways to balance load to ensure scalability.

- **Kubernetes Auto-Scaling on ECS:** The integration of Kubernetes within AWS ECS facilitates auto-scaling capabilities, dynamically adjusting the number of container instances based on the application's load. This feature is crucial in handling varying usage levels, ensuring that the application scales efficiently during peak periods as well as conserving resources during low usage times.
- **Application Load Balancer Integration:** To further enhance scalability and availability, we employed an AWS Application Load Balancer (ALB) that points to the ECS cluster as a target group. The ALB plays a pivotal role in distributing incoming application traffic across multiple ECS container instances. This not only balances the load effectively but also ensures high availability and fault tolerance. The ALB's advanced routing capabilities allow it to respond to incoming traffic smartly and efficiently, significantly improving the overall performance and reliability of the application.



Screenshots of AWS Application Load Balancer Configuration

- Load Balancing with Kubernetes:** Kubernetes within ECS uses its internal load balancing capabilities, which efficiently distribute incoming traffic across container pods. This ensures high availability and consistent performance, crucial during high-traffic conditions.

The combination of Kubernetes' load balancing capabilities within ECS and the external ALB ensures a comprehensive and robust load management system. This dual-layer approach guarantees that the application can handle high traffic demands seamlessly, maintaining a consistent and responsive user experience.

Service Discovery/Registry

- Built-in DNS and Service Networking:** Our architecture utilizes the built-in DNS and service networking capabilities of Kubernetes within ECS. This feature allows for efficient service discovery and seamless inter-pod communication, ensuring that each component of our application can dynamically discover and communicate with others. It aligns with task [N12](#), demonstrating effective service discovery and implementing a service registry through Kubernetes' native features.

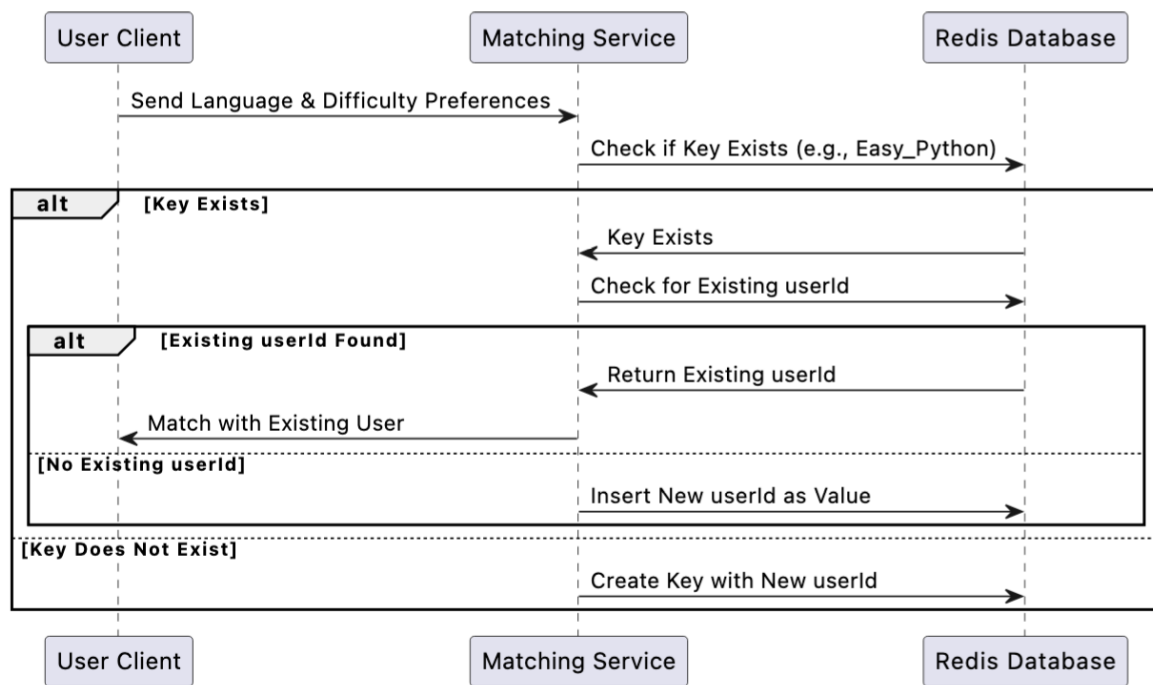
4.4 Design Deep Dive: Matching and Interview Design

As technical interview preparation is a key part of the application, we want to highlight the technical decisions relating to this key feature.

Matching Service

Our Matching Service leverages Redis as a message queue to pair users for interview sessions. This system is designed to match users based on shared programming language preferences and desired difficulty levels.

In our design, we employ the Polling Pattern, where the frontend would regularly poll the matching service to check for updates in user preferences and match them accordingly.



Sequence Diagram of how a user is matched

key	value
Easy_Python	{"d351b3fc-e27a-41ed-b2ad-cb2178f5fd56":{"partner":null,"id":19}}

Screenshot of Redis while a user is waiting for a match

key	value
Easy_Python	{}
session:3d169c97-2b16-4f98-a8...	{"status":true,"id1":19,"id2":20,"userId1":"d351b3fc-e27a-41ed-b2...
user:d351b3fc-e27a-41ed-b2ad-...	3d169c97-2b16-4f98-a8d7-4ef1d942af45
user:3a2ec9d1-d1ce-4b09-841b-...	3d169c97-2b16-4f98-a8d7-4ef1d942af45

Screenshot of Redis once the user is matched

When a user selects the 'Match' option, our matching service promptly adds their profile to Redis, categorizing them under a unique `{difficulty}_{language}` key, reflecting their chosen language and difficulty level. Following this, the user's client initiates a regular polling process with the matching service, spanning over the next 30 seconds, to determine if a suitable match has been found.

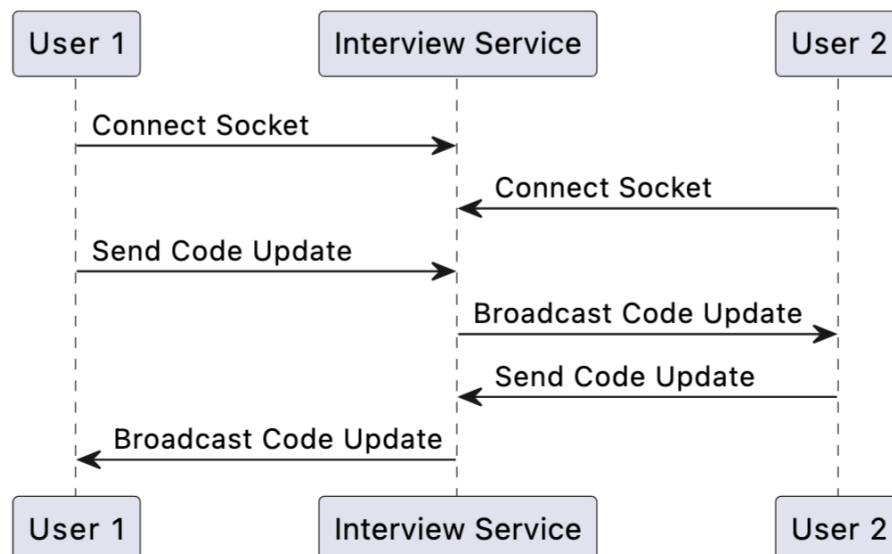
Should a match occur within this timeframe, The matching service would return an interview session id to the user client, and the user would be redirected to the interview page to commence their session.

In the event that a match is not established, the matching attempt is automatically canceled, and their id is removed from the Redis store. The user is then given the option to retry the process, possibly selecting different language or difficulty preferences, to enhance their chances of finding a compatible match.

Interview Service

The Interview Service is a vital component of the application, designed to facilitate interactive, real-time technical interviews. It utilizes WebSockets to enable real-time code collaboration as well as text chat communication, ensuring a seamless and interactive interview experience.

Code Collaboration



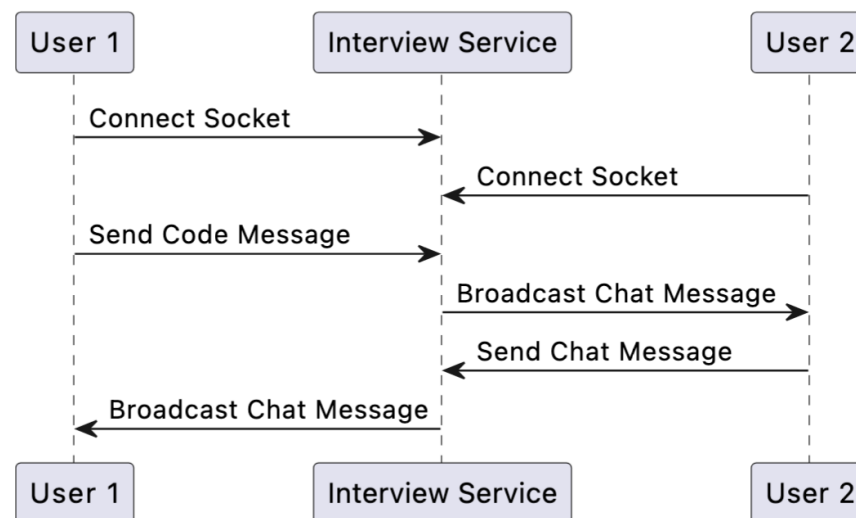
Sequence diagram of code syncing between users

Chat Communication

The chat service in the application is designed as a real-time communication platform using WebSockets, allowing for instantaneous and interactive messaging between users.

The WebSocket server plays a crucial role in this architecture, handling incoming messages and broadcasting them to the appropriate recipients, thus maintaining an efficient and responsive chat system. By establishing persistent connections between the clients and the server, the service ensures low latency and continuous data flow, essential for a seamless chat experience.

Users can connect to the chat service, send messages, and receive responses in real-time, fostering an environment conducive to dynamic conversations. This setup is ideal for facilitating direct user-to-user interaction, making it a vital component in applications that prioritize real-time communication and collaboration.



Sequence diagram of the chat communication between users

4.5 Meeting the Non-Functional Requirements by Design

As summarized earlier, our key prioritized NFRs are *performance*, *scalability*, *usability*, and *maintainability*. We do however have a few additional NFRs that we have considered, such as *security*, which still play into our app.

While some of our NFRs were met through architecture choices as explained above, we wish to concisely and specifically elaborate on which architecture choices contributed to the NFRs, and overall how we feel we have achieved them.

Performance

[React](#) and [Node.js](#) are highly performant and quick tech stacks, and [Redis](#) is also extremely fast. Relying on these help to make our project responsive. Furthermore, the use of [Websockets](#) via the library [socket.io](#)

Scalability

Overall, our main source of enabling scalability comes down to the architecture choices we made. The [microservice](#) architecture allows us to deploy each container separately on cloud and orchestrated with Kubernetes, as elaborated on in the sections on [deployment choices](#) (particularly [Scalability](#), which speaks about the load balancers implemented).

Furthermore, via database choice [Supabase](#), we safeguard ourselves against growing amounts of data. Supabase is a cloud-based backend as a service which relies on PostgreSQL, reliable and able to handle large volumes of data; as it is managed, the Supabase also takes care of much of the difficulty with scaling. We have no issue with thousands of questions in our database

To support large numbers of users, [Redis](#) is a particularly good choice, as it easily supports thousands of users while remaining lightning-fast.

Usability

Our group effectively addressed the non-functional requirement (NFR) of usability in our project by strategically implementing several key features. See [Appendix 1: UI](#) for more details on the functional requirements chosen to implement.

We chose [React](#) Material-UI (MUI) components for their known ease of use and adaptability, ensuring a smooth, intuitive user interface. This choice also facilitated a consistent look and feel across the application, contributing to a coherent user experience. Our commitment to maintaining a consistent “candy” theme throughout the application enhances the visual appeal and improves usability by adding elements of fun and delight to navigating around our app. Using React’s context API, we set a theme context across our frontend to ensure a consistent color scheme and a single source of truth for where the theme would be set. We also adapted components from libraries such as the Monaco library to also use similar theme colors. Despite the fun theme, we also ensured that colors were strongly contrasting so as to benefit accessibility.

To accommodate diverse user preferences, we incorporated both light and dark themes, allowing users to select the mode that is most comfortable for their eyes, a feature particularly appreciated in different lighting conditions or during extended use. Additionally, by saving user preferences, our application provides a personalized experience each time a user returns, remembering their preferred difficulty and language and pre-populating these when matching. These features have an outsized effect on making the user feel like the app cares about what they like.

Maintainability

Certain aspects of our code stack, particularly [Typescript](#) and [Prisma ORM](#) helped to keep our codebase clean and avoid silly bugs like type-based errors. We also enabled [automatic formatting and linting](#). However, a larger portion of the code base's maintainability can be attributed to the [project processes](#) we set in place, including code review.

Our [microservice architecture](#) and [component based react frontend](#) also help to isolate bugs and allow us to modify parts of the application without huge changes needing to be made for the other parts.

Security

While security was not high on our list of NFR priorities, we still took pains to implement certain security features. We will cover two key examples regarding user authorisation.

Firstly, users must log into the application to view pages associated with logged-in users. We use `UserContext` and a separate router for logged-in and non-logged-in routes help to guarantee that unauthorized access to pages is not allowed. For example, if you are not logged in, navigating to a logged-in route such as `/user` will cause a redirect back to the auth page as the not logged-in router does not have that route. Checking the current status is also easily seen in the nav bar, which uses the authorisation context to show the current user.

Secondly, we mandate authentication to do certain activities, particularly those which involve persistent data storage in Supabase. For example, only administrators can add or delete questions; otherwise, these actions are disabled. We use row-based authentication with [Supabase](#) to prevent unauthorized access and modification of credential-related rows. Additionally, the [Supabase Service](#) keeps the logic that ensures users are only able to delete their own user and modify their own password

Reliability

Similarly, reliability was not a key priority. However, our app can maintain high uptime due to the way the [deployment](#) process was; high scalability will allow the app not to crash even under a high volume of users. Furthermore, [Amazon ECS SLA guarantees a 99.99% monthly average uptime](#), so we can infer that the cloud service used for hosting is usually very reliable.

Chapter 5. Future Enhancements

This section proposes potential improvements and enhancements for the PeerPrep platform. These suggestions are aimed at extending the platform's capabilities and enhancing user experience.

5.1 Enhanced Question Repository

- **Diverse Question Types:** Incorporate a wider range of question types, including system design and scenario-based problems, to cater to a broader spectrum of technical interviews. At present, we only include leet-code style questions due to the ready availability of these types of questions
- **Community Contributions:** Allow users to contribute questions to the repository, subject to review and approval, to continually expand the question pool. At present, only administrators can add questions.

5.2 Enhanced Collaboration Tools

- **Integrated Video Calling:** Add a video calling feature within the collaborative space to facilitate face-to-face interaction, making practice sessions more realistic and engaging.
- **Interactive Whiteboard:** Implement an interactive whiteboard tool for diagramming and brainstorming during sessions, especially beneficial for system design questions.

5.3 User Interface and Experience

- **Accessibility Features:** Incorporate accessibility features to make the platform more inclusive, such as screen reader compatibility and alternative image text.
- **Gamification:** Originally, we were hoping to implement the nice feature of badges or being able to rank and review questions in the community. These badges can play into the candy theme and add more delight to using our application. Due to time constraints, we did not implement these features.

Chapter 6. Reflections and Learning Points

6.1 Project Learning Points

Microservice Architecture

One of the key learnings was about [microservice architecture](#). Although our application was relatively small, it served as an excellent project to understand the potential and application of microservices in real-world scenarios. This architecture style, characterized by its division into small, independently deployable services is highly beneficial for larger, more complex applications. We saw the usefulness of relatively independent services firsthand as we were able to work on sections of the application concurrently if we were developing different microservices while limiting merge conflicts.

Deployment Process

[The deployment process](#) was another significant learning area. Our decision to deploy each service separately, where each required its own set of task definitions, made the redeployment process quite tedious. Each deployment required roughly half an hour.

While this was manageable for a small-scale project like ours, it highlighted the importance of infrastructure as code and robust continuous integration/continuous deployment (CI/CD) practices. These practices are crucial for automating deployment processes, thereby saving time and reducing the likelihood of errors, especially in larger projects. However, the tradeoff is the time taken to set up such a system. While this was not a good tradeoff for our team, we would like to explore this more in the future.

6.2 Teamwork Dynamics

Communication played a pivotal role in the project's success. The team dynamics were excellent, with each member respecting others' time and accommodating different schedules. This mutual respect and flexibility allowed us to explore various parts of the project that interested us individually.

[We supported each other](#) through good project management, implementing feature reviews, bug hunting, and quality assurance. These fostered a collaborative and productive work environment. Additionally, our [initial planning](#) paid off; we were able to stick closely to our plans and milestones. Allocating buffer time and organizing in-person syncs, like our week 12 hackathon, proved invaluable for quickly resolving issues and making team decisions. The blend of synchronous and asynchronous work allowed us to accommodate everyone's schedules effectively.

While there were some changes, such as the evolution of our 'nice-to-have' features, our adherence to the original plan was surprising and something we are proud of. We adequately

projected the amount of work that was required through good communication, and this led to a manageable amount of workload and stress without any of us becoming overwhelmed.

6.3 Personal Reflection

The project was a gateway to new technologies for many of us. For instance, using websockets was new to us. Implementing the real time communication offered hands-on learning about real-time messaging systems. Moreover, our team included a math major who was relatively new to full-stack development; this project gave him a valuable opportunity to delve into frontend development. Overall, the project was a rewarding experience for all team members, as we explored and learned more about current technologies. Each member walked away with a deeper understanding and appreciation of the different facets of software development.

Appendix I

Question Service

F1: Question service – responsible for maintaining a question repository indexed by difficulty level (and any other indexing criteria – e.g., specific topics).	Priority
F1.1 Create and maintain a database of questions	H
F1.1.1 Have index on searchable criteria like difficulty level, topic	H
F1.1.2 Able to retrieve a specific question	H

N4: Question service: Enhance question service to enable managing questions, for example, tagging (by topic, popularity, etc.,)	Priority
N4.1 Allow tagging	
N4.1.1 Questions can have global tags	M
N4.1.2 These tags can be filtered on	M
N4.2 Allow random retrieval of question-based on difficulty and language	H
NFR: Scalability and Performance: Question service should remain performant even with 1000 questions	H

User Service

F2: User Service – responsible for user profile management. NFRS: Security	Priority
F2.1 User creation	
F2.1.1 On arriving on the user website, check if user is signed in, and redirect to login/signup page if not	M
F2.1.2 Signup page asks for email and password	

F2.1.3 User is asked to confirm password	L
F2.1.4 Password must meet certain best practices: eg more than 8 characters	L
F2.2 Email confirmation	
F2.1.2 User must confirm their account through their email	M
F2.3 Manage account	
F2.3.1 Provide a way for them to delete their account	H
F2.3.2 Provide a way for them to change the password of their account	H
F2.3.3 Provide a way for them to change the name of their account	M
F2.3.4 Provide a way for them to change the preferences of their account (language, difficulty)	M

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.	Priority
N2.1 Save questions attempted	
N2.1.1 After attempting a question, the question, the date-time of attempt, and the attempt itself are saved to the history	M
N2.1.2 User can view their history	M
N2.1.3 User can re-access the question	M
N2.1.4 User can reassess their attempt of the question for review	M
N2.1.5 User can view the profile of the user they did the question with	M
N2.1.6 User can sort their history to find the question they want	L
NFR: Usability: The viewing and managing of history is intuitive to users	M
NFR: Scalability and Performance: History should remain performant even with 1000 questions saved in history	M

Matching Service

F3: Matching Service – responsible for matching users based on some criteria	Priority
F3.1 Implement a matching queue	
F3.1.1 Allow users to enter a waiting queue if there is no immediate match	H
F3.1.2 Implement a fixed wait time (e.g. 30s) to find a match	H
F3.1.2.1 Provide feedback to the user during wait (e.g., animation to indicate the user to wait until match is obtained)	M
F3.1.3 Allow users to exit the queue if the wait is too long	H
NFR: Usability: Users understand the current state of their match and find the process reasonable	M
F3.2 Match based on preferences	
F3.2.1 Allow users to filter matches based on question difficulty and language	M
NFR: Usability: User preferences are saved and pre-populated for convenience	M
F3.3 Matched users should be sent to the same question (see F4)	H
F3.4 Question chosen should fulfill preferences and be randomized (see N4.2)	M

Interview Service

F4: Interview service – provides the mechanism for real-time collaboration (e.g., concurrent code editing) between the authenticated and matched users in the collaborative space. NFR: Performance, Reliability, Interoperability	Priority
F4.1 Implement Concurrent Code-Editing	
F4.1.1 Allow user to view matched user's code editing in real-time	L
F4.1.2 Allow user to edit on the same text editor as matched user	L
NFR: Performance: Changes should be visible within 1s	M

NFR: Reliability: Code edited should not be lost even if the connection is intermittent	L
--	---

N3: Communication: Implement a mechanism to facilitate communication among the participants in the collaborative space (other than the shared workspace)	Priority
N3.1 Allow users to communicate during collaboration	
N3.1.1 Provide text-based chat service	M
N3.1.1 Clearly show which messages are from which user	L
NFR: Usability: Text-based chat should not take up much space when not in use	M
NFR: Performance: Messages should be received in under 1s	M
NFR: Usability: Messages should be received and retained for the session even if the other user loses connection or navigates away	L

N5: Enhance interview service by providing an improved code editor with code formatting, syntax highlighting for one language, syntax highlighting for multiple languages.	Priority
N5.1 Enhance code editor	
N5.1.1 Allow code formatting	M
N5.1.2 Syntax highlighting for one language	M
N5.1.3 Syntax highlighting for more than one language	L
N5.1.4 Code editor has a light and dark theme that user can choose	L
NFR: Usability: Themes of code editor should contribute to the cohesive style of the application	L

UI

F5: Basic UI for user interaction – to access the app that you develop. NFR: Usability	Priority
---	----------

F5.1: Login / Signup pages for users	
F5.1.1 Login and Signup modal that shows up if users are not signed in	H
F5.1.2 In the case of wrong entry into fields, an error message or notification appears on the screen and guides the user	H
NFR: Usability: Login and signup pages should look visibly similar to make things readable for the user	M
NFR: Usability: Navigating to a page you cannot access when not logged in should redirect you back to authorisation screen	M
F5.2: Question Service	
F5.2.1 Allows the adding, editing, and updating of questions for administrators	M
F5.2.2 Allows the sorting and filtering of questions for all users	M
F5.2.1 Allows an in-depth view of question details	M
F5.3: Matching	
F5.3.1 On the matching screen, users can specify some basic criteria to match on, perhaps in a pop-up modal	M
F5.3.2 While waiting for a match, a loading screen or similar shows that the app has not hung	M
F5.4 User page	
F5.4.1 Allow users to choose a preferred language and difficulty	L
F5.4.2 Provide on the user page the user history where they can see and sort questions attempted so far	L
F5.4.2 Allow users to edit their name and password	M
F5.4.2 Allow user to delete their account	M
F5.4.2 Allow users to view other users pages	L
NFR: Security: Only allow editing of user information and deleting of user on users own page	M
F5.4 Implement a text editor for users	
F5.4.1 A separate component that allows users to type in code	H
F5.4.2 A component for users to chat and send messages	M

F5.4.3 A button for users to submit code and end the session, saving the question to the history	M
NFR: Usability: In the case of an inconsistent or intermittent connection, what has been typed should not be lost	M
F5.5 Night/Light mode	
F5.5.1 A toggle for night/light theme for users which is also saved as a user preference	L
F5.6 Other	
F5.6.1 Delightful / Surprising features: randomized candy emoji bullet points and decoration, home icon changes emoji when clicked	L
NFR: Performance: App should feel responsive and never take more 1s to show that it is responding	M
NFR: Usability: When app must wait, loading screens or similar should be visible to show that the app has not hung	M
NFR: Scalability: Performance should not degrade even if a 1000 users are using the application	M

Deployment, Scalability

For a deeper dive into the design decisions behind this, see [4.3 Deployment](#).

F6: Deploy the app on a (local) staging environment (e.g., Docker-based, Docker + Kubernetes). NFR: Performance, Reliability, Scalability	Priority
F6.1 Local deployment	
F6.1.1 Project should have easy-to-follow set-up instructions (docker-compose up -d)	H
F6.1.2 Project should clear env.example set up files & package-lock.json to make it easier when setting up the project on a new machine that has a consistent development environment (maintainability)	H
F6.1.3 Project should have set up scripts to populate the databases (if local database) for testing	H

N9: N9.1 Deployment of the app on the production system <i>NFR: Performance, Scalability</i>	Priority
N9.1 Deploy on AWS	H
N9.1.1 Use AWS services to ensure auto-recovery and self-healing capabilities	H
N9.2 The deployment architecture must be optimized for low-latency response times under typical load conditions.	H
N9.3 The application should have health checks in place	M

N10: Scalability – the deployed application should demonstrate easy scalability of some form.	Priority
N10.1 The application must support the automatic scaling of resources, employing Kubernetes horizontal pod auto-scaler to manage the scaling of application pods efficiently in response to fluctuating loads.	M
N10.2 Implement AWS Application Load Balancer Integration	M

N12: Implement service discovery or implement a service registry of some kind.	Priority
N12.1: Deployment via AWS ECS for inbuilt DNS and service registry via Kubernetes	M