PeerPrep Project Report

CS3219

Software Engineering Principle and Patterns

Group 25

Group members:
Wang Xinyi (A0243320X)
Yan Xiaoying (A0237873R)
Lin Leyi
Fang Yiye
Tan Yong Feng Deon (A0240793B)

Table of Content

Color Code in this report

- Must to haves
- Nice to haves implemented by Group 1
- Nice to haves implemented by Group 2

Background and purpose of the project

Our project is aimed to address a growing need within the student community for a collaborative platform focused on coding and programming practice. In an era where computer science and coding skills are increasingly essential, we recognized the importance of fostering a supportive environment for students to enhance their programming abilities. PeerPrep serves as a digital bridge, connecting like-minded students who aspire to improve their coding skills by providing them with a platform to match and collaborate on coding questions. Our purpose is to facilitate peer-to-peer learning and create a vibrant community where students can share knowledge, tackle challenging coding problems together, and, ultimately, hone their coding proficiency. PeerPrep aims to empower students on their coding journey and make the process of skill development more engaging and efficient.

Specification of Requirements: Functional Requirement

FR1 - User Service(M1):

F1.1 Implement user authentication for authorized access.

- Users can create accounts with unique usernames and passwords.
- Implement role-based access control to distinguish between maintainers and users.

F1.2 User profile management.

- Users can log in by Credentials such as GitHub or Discord accounts.
- Users information such as user name, email address, avatar will be maintained.
- Provide the option for users to delete their accounts.

F1.3 Role Management (on CRUD of question repository).

- Maintainer: have full CRUD ability to the question repository.
- User: can only add questions into the question repository.

FR2 - Matching Service(M2):

F2.1 Student to be matched with another student who has chosen the same difficulty level.
- Allow users to select the difficulty level and specific topic they want to practice.
- Ensure that questions are randomly assigned to pairs to prevent predictability

F2.2 Time out if matching was not completed in 30 seconds.
F2.3 A retry button allows users to retry matching after failing matched.
F2.4 Each room has a unique room ID and uses API to access.
- Rooms can be created, deleted, listed and updated.

F2.5 Match successfully will link to the collaboration room.

FR3 - Question Service(M3):

F3.1 Maintainers are able to create, read, update, and delete the question in question repository .
- Organize questions by difficulty level and specific topics.
- Allow for easy addition, modification, and removal of questions.

F3.2 Registered users are able to read questions from the question repository.
F3.4 History Service:
- Each user will have a personalized section within their profile that maintains a record of all the interview questions they have attempted.(N2)
- The history will include details such as the question difficulty level (easy, medium, or hard), the topic of the question, and the timestamp of when the question was completed.

F3.5 Enhance question (Retrieve questions on the fly) (N4)
- When the room is created, any one of two users may select the questions from the question repository of their chosen difficulty level.
- The other user will receive a notification stating that their partner has picked a question.
- This question will be stored in the database such that users will be able to access the same question if they resume the session after leaving.

FR4 - Collaboration Service(M4):

F4.1 Students are able to view questions of their selected difficulty.
F4.2 Students are able to develop their solutions simultaneously on the provided question.
- A collaborative coding space that supports real-time code editing, and code sharing.
- The collaboration service is accessible and intuitive for both users

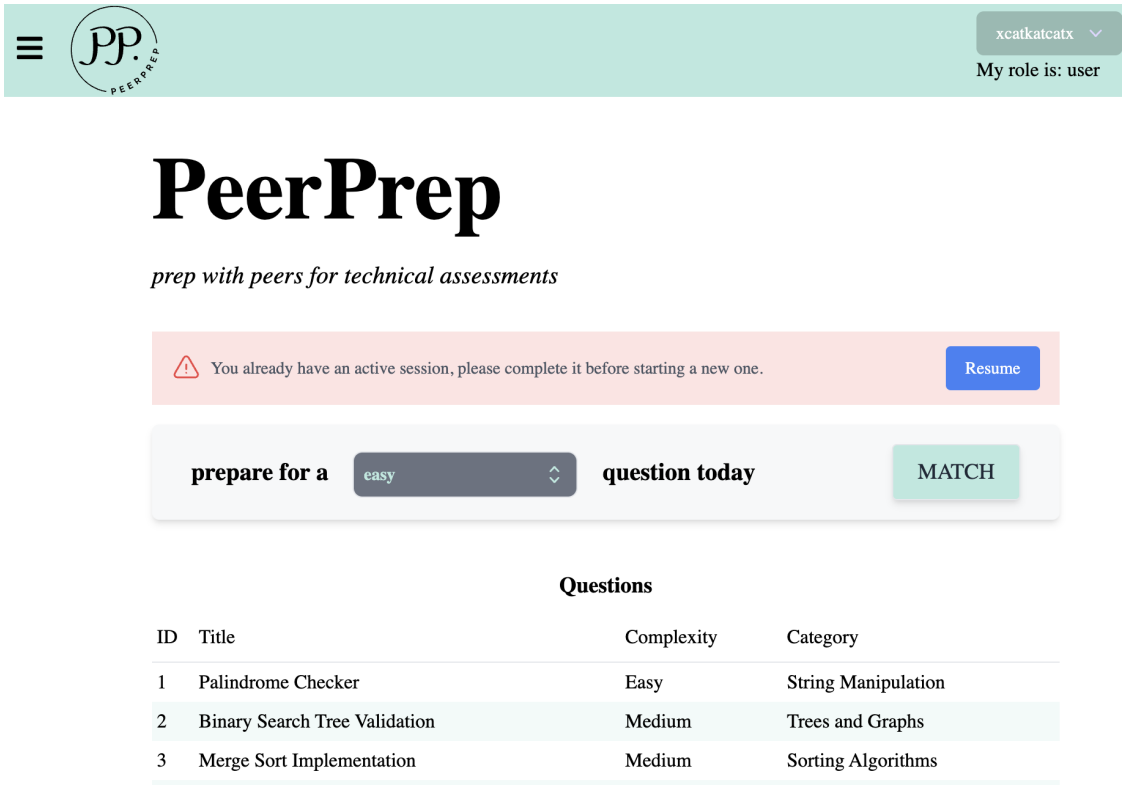
F4.3 Enhance code editor(N5)
- Enhance collaboration service by providing an improved code editor with code formatting, syntax highlighting for one language, syntax highlighting for multiple languages.
- Support four kinds of programming language as well as their syntax and auto-filling, including Java, JavaScript, Python, C++.

F4.4 Students are able to end the session gracefully.

- Provide users with the option to gracefully terminate a collaboration session.
- End session button is provided for exit or terminate the session.

F4.5 Save the progress made during the session for later review (Abrupt end session)

- In the case when one user suddenly closes the tab or loses connection during a session.
- Allow re-enter the room: send a message to the partner to notify that partner disconnected.
- When the partner is disconnected, the user can end the session by themselves.
- However, if the session is not terminated, the user cannot match again with others.



FR5 - Incorporate generative AI (N7):

F5.1 Incorporate generative AI to assist during the preparation.

- Users can utilize ChatGPT to seek in-depth explanations and insights into code written by their peers, enhancing their understanding and learning experience.
- Our generative AI system can assist users in solving coding puzzles by accepting prompts provided through the interface. Users have the option to request open-ended prompts, tailoring the assistance to their specific needs.

FR6 - Continuous Integration / Deployment (N8):

F6.1 Extensive (and automated) unit, integration, and system testing using CI.

- We make use of GitHub hosted virtual machines (runners) to run automated build scripts and test scripts through GitHub Actions and workflow files defined in YAML.
- These files enable building and testing on all 3 major operating systems, Windows, Mac, Linux.
- Automated the release of a good build to the production environment. (GCP CloudRun & CloudBuild)

Status	Build	Source	Ref	Commit	Created	Duration	Security Insights
✓	65cb2645	CS3219-AY2324S1/ay2324s1-course-as...	master	adc6b0c	11/14/23, 3:26 PM	6 min 38 sec	VIEW
✓	22e7cc1e	CS3219-AY2324S1/ay2324s1-course-as...	master	0203c86	11/14/23, 1:14 AM	4 min 50 sec	VIEW
✓	70ebd0ad	CS3219-AY2324S1/ay2324s1-course-as...	master	903a3d4	11/13/23, 11:35 PM	7 min 24 sec	VIEW
✓	e323660e	CS3219-AY2324S1/ay2324s1-course-as...	master	5f66065	11/11/23, 11:43 PM	7 min 4 sec	VIEW
✓	11b23260	CS3219-AY2324S1/ay2324s1-course-as...	master	6d957b1	11/11/23, 10:26 PM	6 min 21 sec	VIEW
✓	6e02f99b	CS3219-AY2324S1/ay2324s1-course-as...	master	29b31af	11/11/23, 9:36 PM	5 min 41 sec	VIEW

Specification of Requirements: Non-functional Requirement

NFR1 - User Interface and Usability(M5):

NF1.1 Layout and structure of web pages to be clear and easily navigable.

- Lowers cognitive load on the user to understand the interface at a glance.
- User interface has a consistent design and layout throughout the application.
- Design a user-friendly interface with a clean and intuitive layout.
- Ensure that users can easily navigate the application and understand its features.

NFR1.2 The homepage is accessible even when users are not logged in

- We use the middleware file to achieve the not-logged-in page.
- The sign up button leads to signing up as a new user.

NFR2 - Deployment (M6):

NF2.1 Deployment on Docker-based staging environment

- The application supports deployment on both local development environments (individual machines) and Docker-based staging environments.
- Docker-based support deployment in different computer environments.
- Docker based on microservices, including:
    - Frontend-service
    - Question-service
    - User-service

- Matching-service
- Collaboration-service
- Frontend-service
- History-service

NFR3 - Scalability (N10):

NF3.1 The application is designed to scale horizontally to accommodate increasing data and user loads.

- Use Kubernetes as Orchestrator, support heterogeneous platforms, for example, a variety of OS/Browser/Platform combinations.
- Design the application to be horizontally scalable, allowing it to handle increased user traffic.
- Implement load balancing to distribute requests evenly across multiple server instances.
- Monitor and auto-scale resources based on usage patterns.

NFR4 - Security:

NF4.1 User authentication using OAuth with popular online services like GitHub, Discord, without exposing user account credentials.

- Store user information from online services into our user database.

NF4.2 Access control mechanisms prevent unauthorized access to questions repository and collaboration space.

NF4.3 Prevent unauthorized resource access from a different domain.

- Use of CORS(Cross Origin Resource Sharing) protection to prevent any random website from using authenticated cookies to send an API request.

NF4.4 Session management using JWT tokens and database.

NFR5 - Performance:

NF5.1 The application can respond to user requests quickly.

- Ensure that the application performs well under expected user loads, with minimal latency.

NFR6 - Reliability and Maintainability:

NF6.1 The application is easy to maintain, update and debug.

- Establish a routine maintenance schedule to apply updates and patches to the application and its dependencies.

- Each service is built under MVC patterns, and there is detailed README to instruct with developing for each service.
- Ensure backward compatibility when making changes to existing features.

NF6.2 Reliability: Minimize system failures and errors by thoroughly testing the application.

- Maintain a system uptime log to track and address any recurring issues.

NFR7 - Compatibility and Interoperability:

NF7.1 Website is able to work under different browser environments

- Users will use browsers on Windows, MacOS, or Linux . We ensure the website works on those browsers.

N7.2 By using Containerization, we support dependency and change management.

- Caters to heterogeneous platforms
- Improved interoperability and portability and maintainability.

Sub-groups: Small Team Ownership

Group 1: Lin Leyi, Fang Yiye, Tan Yong Feng Deon

Nice-to-haves Responsibility:

- N2 : History service(view attempted questions)
- N4 : Question service enhancement (Retrieve questions on the fly)
- N8 : CI/CD workflow for Integration and Deployment on GitHub

Group2: Wang Xinyi, Yan Xiaoying

Nice-to-haves Responsibility:

- N5 : Real time coding space (support 4 languages syntax styling)
- N7: AI chat prompt service (with OpenAI API)
- N10: Scalability (using Kubernetes HPA)

Each group/group member will only need to focus on the feature they need to develop, which helps with the decentralized development and governance.

Individual / Sub-group Contribution and Backlog of Project:

Time	Task	Contributor(s)	Technical /Non-technical
Sep 7	Update README.md with instructions to run the app. Install dependencies for API development. Setup node start and dev scripts.	Deon	Non-Technical

Sep 8	Setup jest testing suite. Create CI workflow for Frontend and Backend. Setup backend API. Modify and Troubleshoot CI workflows.	Deon	Technical
Sep 9	Setup MongoDB cloud cluster for questions API. Design database schema for questions. Develop API routes for CRUD questions.	Deon	Technical
Sep 10	Decide to work on each person's fork repo, and PR.	All	Non-technical
Sep 11	Add pagination for GET questions.	Deon	Technical
Sep 12	Finish Assignment 1: Single Page Application of CRUD questions; Use browser's cache to store question; Deploy the SPA in GitHub.	Xinyi, Xiaoying	Technical
Sep 13	Start Assignment 2 to include users API; Create database for users and Express create route for user profiles; Implement REST API for the user database.	Deon, Leyi, Yiye	Technical
Sep 14	Create and deploy Frontend using Next.js; Choose Next.js with the concern of containerization; Use a CI script to deploy the React app to GitHub pages(N8).	Deon (Group 1)	Technical
Sep 16	Basic UI for Sign in, Homepage with Next.js; Routing between pages; Add an API to get users by email, last time we used ID only.	Leyi	Technical
	Made the Sign up and Profile page; Enhanced current Homepage; Routing between pages; Integrated CRUD operations for user profile from Frontend with Backend API.	Yiye	Technical
Sep 17	Create workflow to deploy next.js app on GitHub page. Optimize existing workflow files. Add status badges in README.	Deon	Technical
	Improve error handling and fix small random mistakes.	Leyi, Deon	Technical
Sep 18	Setup user authentication using next-auth. Design PoC of new homepage using next.js.	Deon	Technical
Oct 1	Migrate Assignment 1 SPA to use Next.js framework;	Xinyi	Technical
Oct 2	Migrate Frontend to use TypeScript. Modify next-auth authentication providers. Decide not to deploy on GitHub.	Deon	Technical

Oct 6	Add API route for questions to frontend of GET POST PUT DELETE.	Deon	Technical
Oct 12	Draft Dockerfile for frontend.	Deon	Technical
Oct 13	Frontend Design: redid the vercel landing pages with our SideBar.	Leyi	Technical
Oct 16	Dockerize the Frontend and Backend.	Xinyi	Technical
Oct 20	Matching Service added for Backend; Integrated the matching service with the Frontend.	Yiye	Technical
	Finish refactoring and merging Assignment 2.	Xinyi, Leyi, Deon, Yiye	Technical
Oct 25	Add owner parameter to question schema. Use prisma adapter for next-auth-database integration. Add exclusion to root and api route in middleware session authentication. Use Prisma adapter to integrate session and user management using database; Modify existing postgres schema to fit usage of Prisma adapter. Allow users to login with any password as long as the email is found for dev purposes.	Deon	Technical
Oct 29	Implement Manage Question Page; Users can add questions, edit and delete questions that they created.	Leyi	Technical
Nov 1	Use vercels live blocks to construct a real time code editor. Collaboration service is under localhost 3001 temporarily(N5). Add the logo for this web application. Add detailed README for this web application.	Xinyi (Group 2)	Technical
Nov 2	Matching service: Using a regular queue with the web socket, and integrated with the frontend. Able to match two users at the same time.	Yiye	Technical
Nov 3	Clean up the code base. Implement rooms API, for two users having the same unique room ID.	Deon	Technical
Nov 4	Refactor the whiteboard and frontend whiteboard; In the name of modularity, refactored the frontend portion from the collaboration-service	Leyi	Technical

	Style for the code editing page (whiteboard); Include the language selection and AI chat bot.		
	Design and add logo in main page and Update README accordingly.	Xinyi	Non-technical
Nov 5	Add the language switch in to the code editor; Enable different programming language syntax and auto filling.	Xiaoying	Technical
	Containerize all the services; Made the docker images for each service.	Xinyi	Technical
Nov 8	Handle the matching timer run out; UI to display when the timer run out; useContext to establish connection between wrappers.	Leyi	Technical
	Edit the matching service logic and added the history service(N2).	Yiye (Group 1)	Technical
Nov 9	Add implementation for user role authorization; Added Discord login functionality. Added role based authorization as well as example. Currently all new users will be "user" role.	Deon	Technical
	Kubernete HPA for each service(N10); Fix minor styling bugs in the code-editor.	Xinyi (Group 2)	Technical
Nov 10	Add ChatGPT AI chatbot(N7). Implemented OpenAI API into the collaboration and frontend service. Users can access the AI chatbot in the collaboration service page, and ask ChatGPT questions about the coding problem.	Xiaoying (Group2)	Technical
Nov 11	Fix Match retry and add selection questions(N4).	Leyi (Group 1)	Technical
Nov 12	Add a dynamic route for the collaboration room.	Deon	Technical
Nov 13	Write the report	All	Non-technical
	Only Admin (Maintainers) can modify questions. Redirection to unauthorized page based on role. Passing of role parameter to SideBarWrapper.	Deon	Technical
	Get GPT AI hint prompts	Xiaoying	Technical
	Implement web sockets in Whiteboard	Yiye, Leyi	Technical
* More detailed commits can be found in our team repo PR history.			

Developer Document (Technical)

Tech Stack we used:

- Frontend: Next.js (TypeScript), React.js (JavaScript)
- Backend: Node.js
- Database: MongoDB, PostgreSQL
- Authentication: Next-Auth
- CI/CD: GitHub
- Conterization: Docker and Kubernetes
- Matching Service: Queueing
- Collaborative Service: Vercel Liveblocks (web socket)

Development Process:

1. **Project Inception**

- Initial idea and concept behind PeerPrep.
- Set goals and objectives for the project.
- Discuss the team composition and roles.

2. **Requirements Gathering and Analysis**

- Detail the process of gathering and analyzing the functional and non-functional requirements for the platform.

3. **Design Phase**

- Discuss the architectural decisions, including the choice of technologies, databases, and frameworks.
- Outline the system's high-level design, emphasizing how it incorporates principles such as modularity and scalability.
- Design user interface and any user experience considerations.

4. **Development and Implementation**

- Our development process was organized into iterative cycles, typically following a one-week sprint model. During each sprint, we set specific development goals and priorities, which were aligned with the project's overall objectives.
- We followed a modular development approach, creating independent, self-contained components for various functionalities. Each module was developed with well-defined boundaries and interfaces.
- Containerized various components of the platform using Docker.

- Orchestrating the deployment and scaling of containers within our infrastructure with Kubernetes and HPA.

5. **Quality Assurance and Testing**

- Extensive (and automated) unit, integration, and system testing using CI.

6. **Deployment and Scaling**

- Deployment on Docker-based staging environment.

7. **Documentation**

- Record everything by this report.

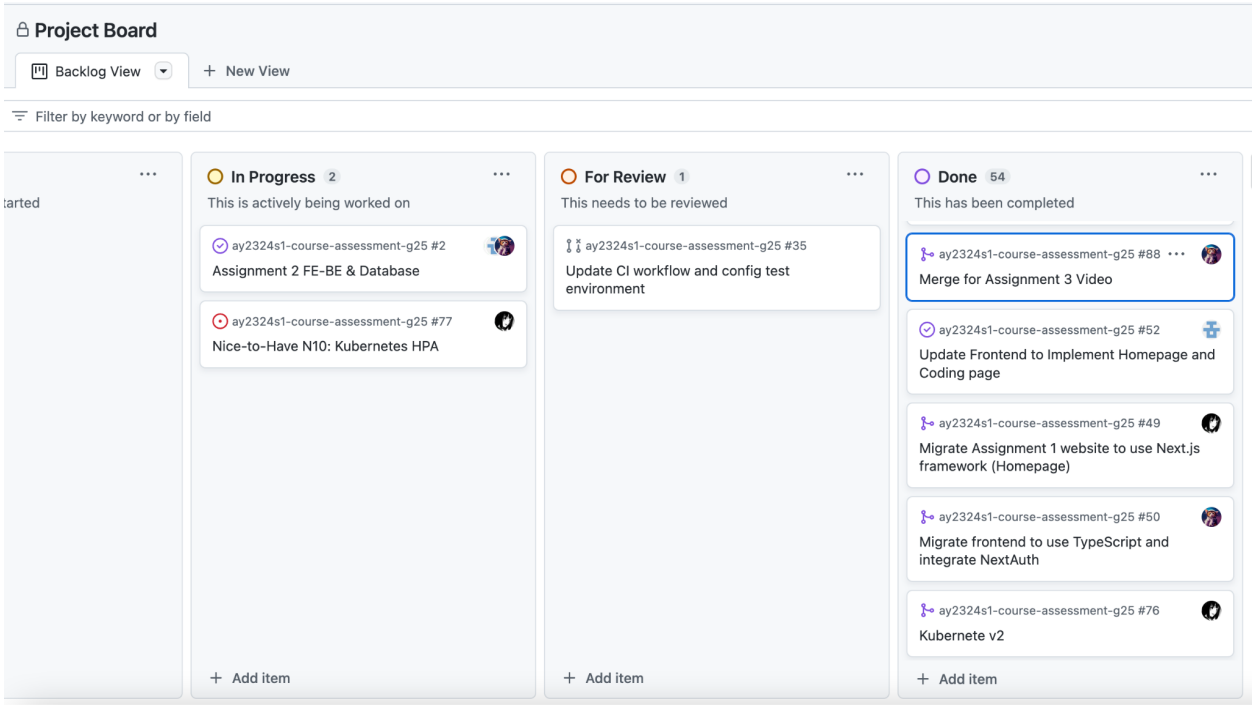
Project Management:

1. Agile Scrum : Sprint meeting for the next task

In adherence to the Agile Scrum methodology, during the Sprint Planning Meeting, our team collaboratively identifies and prioritizes tasks for the upcoming sprint. Each team member discusses their potential contributions, and the team collectively commits to a set of sprint goals. This iterative planning process ensures that the team stays focused on delivering incremental value to the project.

2. Project Kanban Board

To enhance transparency and provide a visual representation of the project's progress, we utilize a Project Board.

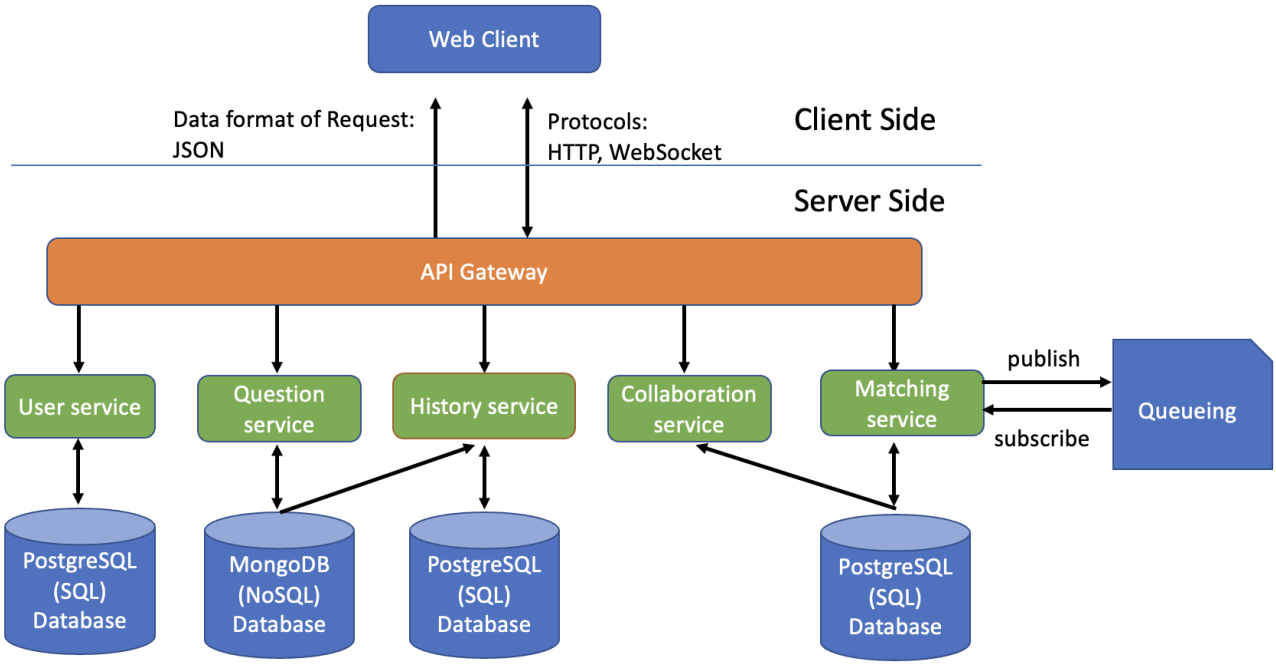


3. Individual Repositories and Pull Requests

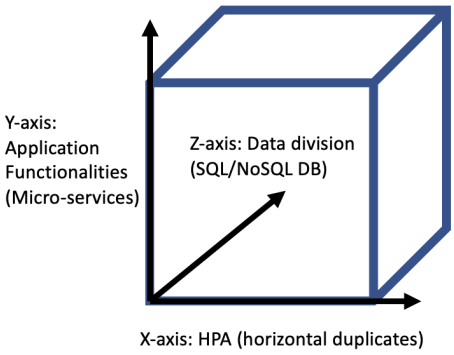
Each team member maintains an individual fork of the team repository. This decentralized approach allows team members to work on features or bug fixes independently, reducing conflicts and streamlining the development process.
After completing a sprint task, team members initiate a Pull Request (PR) to merge their changes into the main repository.

Design of the application

Microservices Architecture Diagram



1. Frontend (Web-client)
2. User-service
3. Question-service
4. History-service
5. Collaboration-service
6. Matching-service

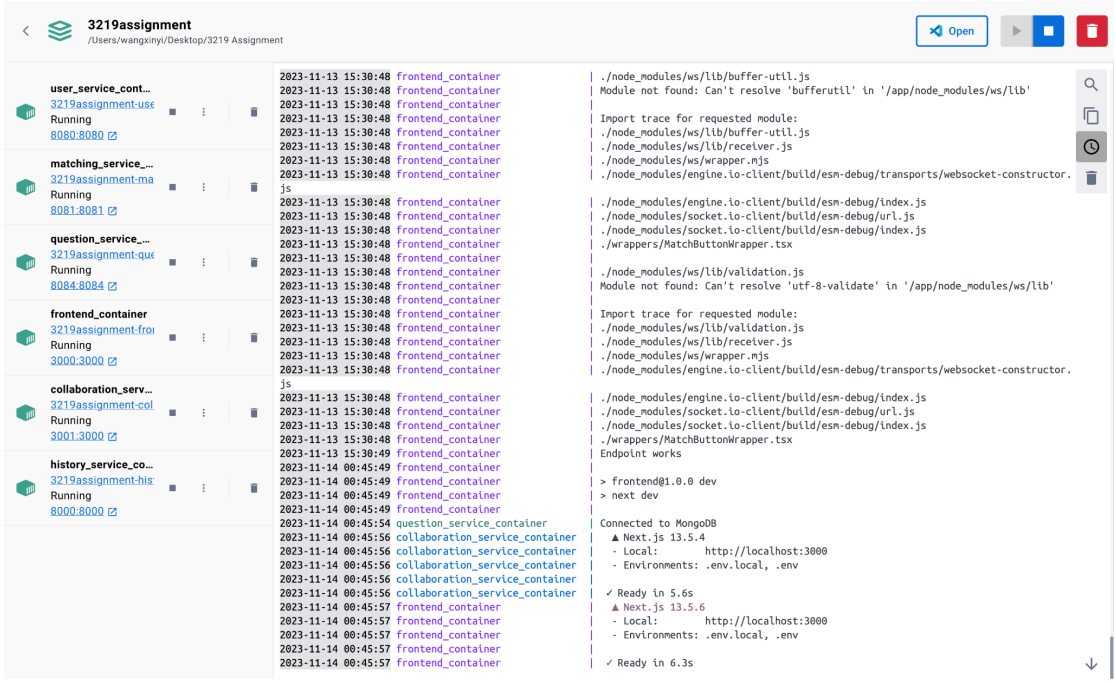


Container -Docker & Orchestrator - Kubernetes(HPA)

We are using Containerization that allows us to provision containers to run the applications. Apart from the request that this project needs to be under microservice architecture, containerization is ideal for CI/CD and Agile practice in our team. Here are some of the advantages or benefits:

- Granular and controllable: deployments can be of the whole system or just an element within the system; And the deployment can be monitored, so that any rollback or redeploy can be done easily.
- Reproducible: guaranteed to be identical on any system that run containers.
- Isolation and security: avoid conflicting dependencies.
- Quick and easy to launch.

Docker Desktop was used to build and run the images and containers.



By using Docker, it helps hold the code and the data, and it can be run in order for the overall software system to work. At the same time, Docker provides separate deployable things for each container.

Lens is used to monitor the Kubernete clusters.

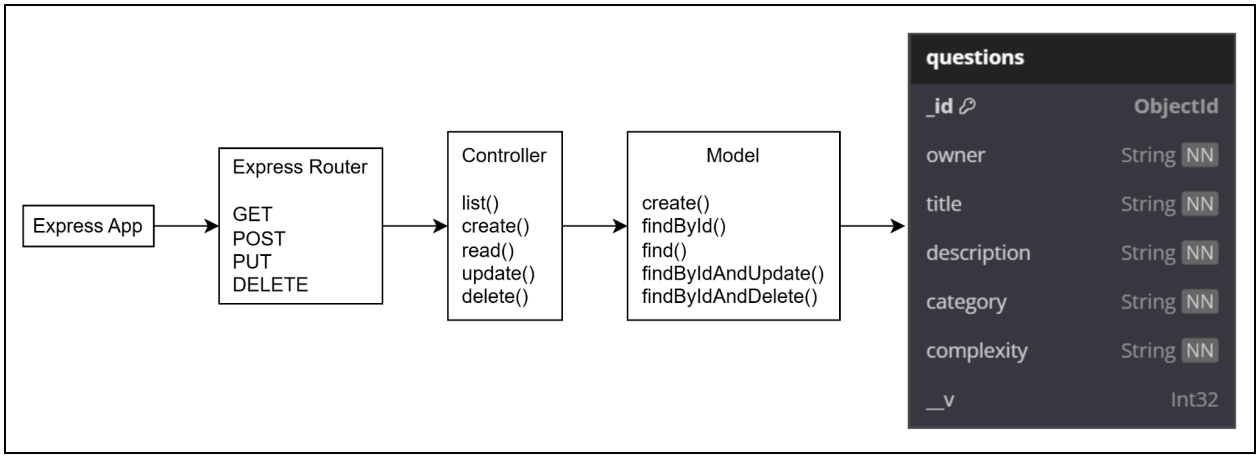
By horizontally scaling the pods, if pods' CPU grows up more than 60% , then HPA creates another pod to keep each pod resource utilization within 60%.

This increases the resilience or fault tolerance, but adds cost and complexity. As our micro-services are designed to be small, focused and independently deployable, we chose horizontal pod scaling.
And we have been monitoring the resource usage within our Kubernetes cluster and fine-tune resource allocation for containers to avoid underutilization or overloading.

Per-service Database Design

We designed our question repository using a NoSQL database like MongoDB. Leveraging MongoDB deployed in the cloud, our database schema encompasses fields such as "_id", "owner", "title", "description", "category", "complexity", and "__v". The "_id" field represents a unique alphanumeric identifier using ObjectId built into MongoDB. The "__v" field denotes the version number of the database deployed, for version control during any data migration.

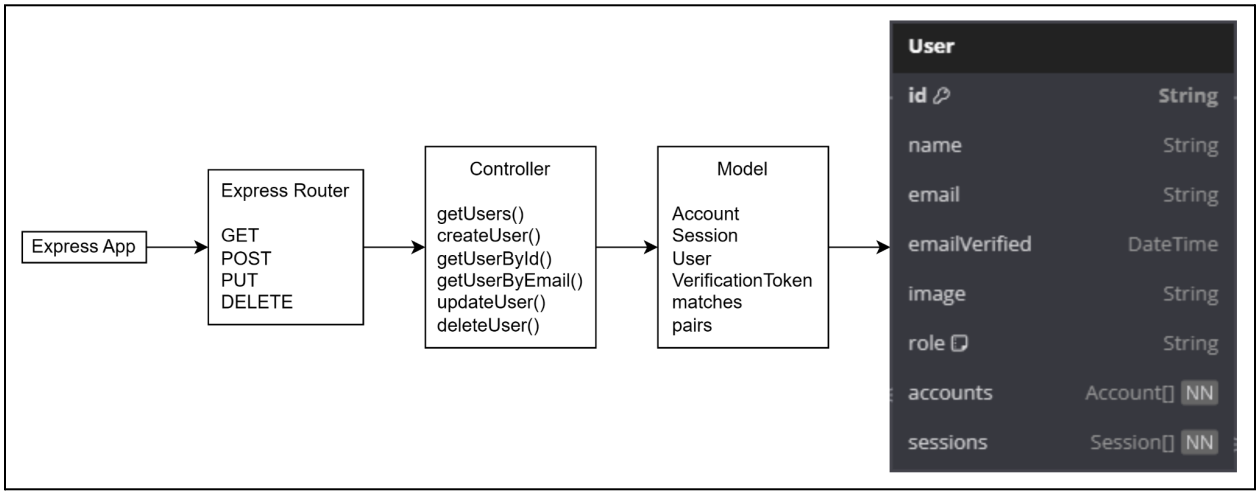
In alignment with our microservices architecture, we interact with this database by implementing a backend service that exposes an API using Express. This approach ensures a modular and scalable solution for both the question API service as well as the question database service. Below is an image of a request flow as well as the schema.

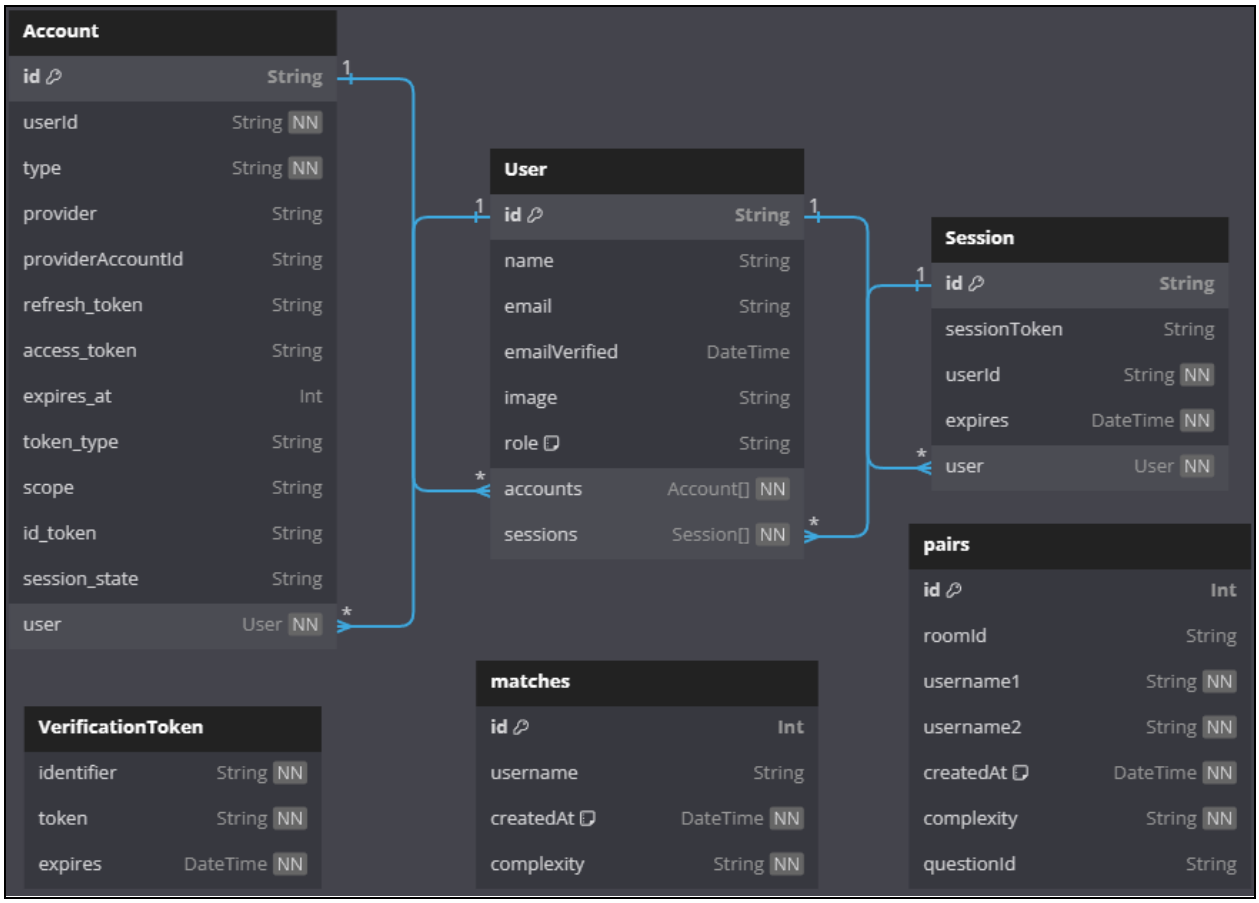


Our current MongoDB deployment utilizes a free-tier shared compute system for development. There is an option to scale as the user base or data volume grows. This strategic choice in deployment allows us to keep development costs to a minimum.

In addition to our NoSQL design, we also designed our user repository using a SQL database like PostgreSQL. For this we used PostgreSQL 15 deployed on the Google Cloud Platform. Our database schema contains tables such as “Account”, “Session”, “User”, “Verification Token”, “matches”, “pairs”. The “Account” table contains details about the different accounts users might have. E.g., “type:oauth”, “provider:github”. The “Session” table contains active sessions of logged in users. The “User” table contains details like “name”, “email”, “role”.

We also interact with this database via a backend service that exposes an API. Below is an image of a request flow, as well as the schema.





For our other backend services like matching-service and history-service, we also use a similar approach as user-service.

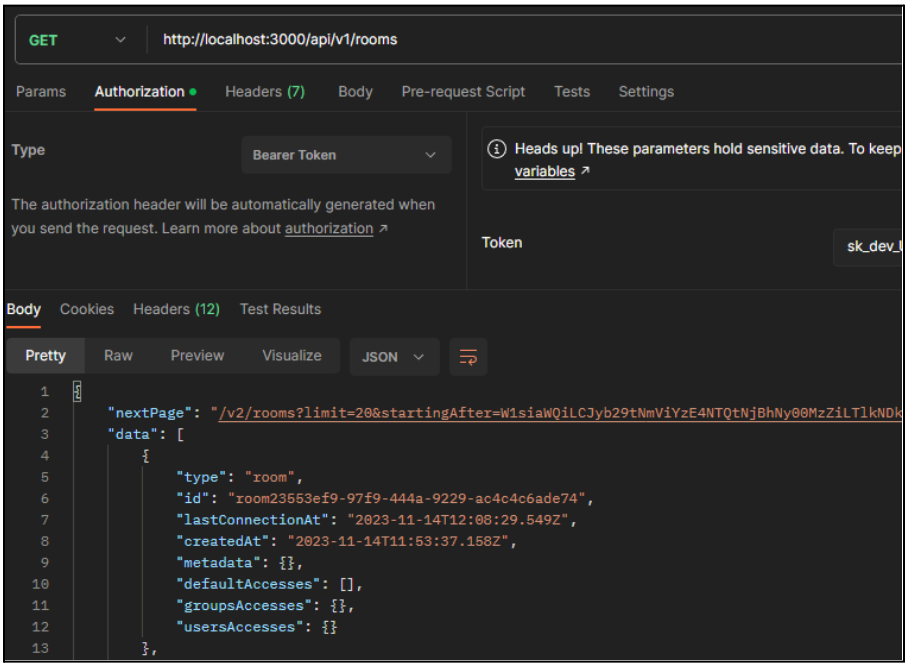
Overall, by decentralizing the data management, each service is responsible for its own data.

Rest API

Following our microservices architecture, the implementation of multiple REST APIs within our application allows us to establish a clear separation between microservices. We are able to decouple services and promote modularity, scalability, and platform independence. These APIs enable other components of our application to perform operations such as creating, reading, updating, and deleting data. We have the following services that provide APIs for other components. 1. Question-service, 2. User-service, 3. Matching-service, 4. History-service, 5. Collaboration-service.

Furthermore, security in any application is a paramount concern. Some of our REST APIs are designed to authenticate requests using the Authorization Header with a “Bearer TOKEN” for authorized access.

```
const response = await fetch(BASE_URL + `/rooms/${id}`, {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${process.env.SECRET_KEY!}`
  }
})
```



In the event of invalid requests, default parameters are utilized.

```
// Handle GET Requests to /api/v1/questions
questionRouter.get('/', async (req, res) => {
  const page = req.query.page ? parseInt(req.query.page) : 1;
  const limit = req.query.limit ? parseInt(req.query.limit) : 10;
  console.log(`GET /api/v1/questions?page=${page}&limit=${limit} from ${req.hostname}`)
  const result = await QuestionController.list(page, limit);
  res.send(result);
});
```

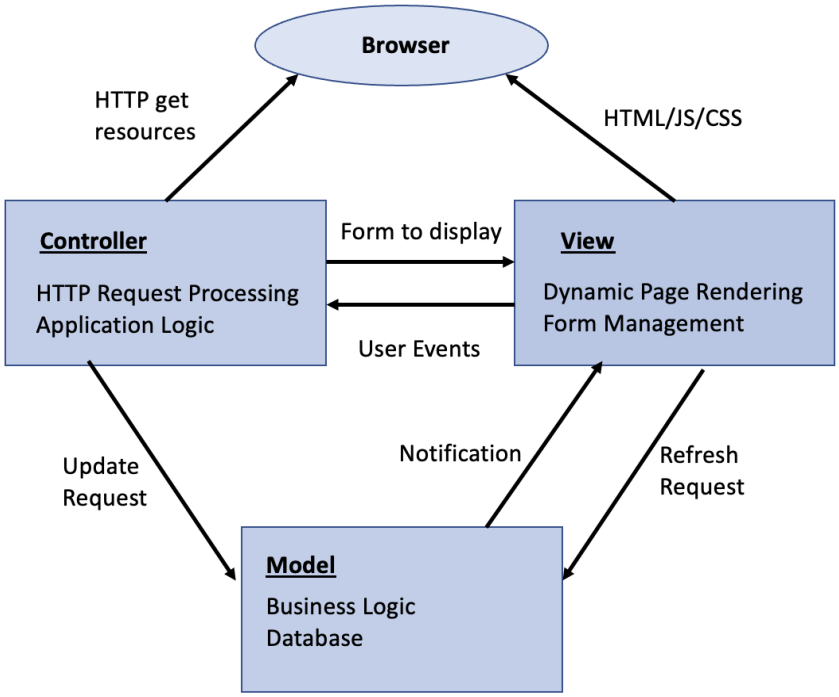
We have also fortified against potential injection attacks via parameterization.

```
pool.query('SELECT * FROM public."User" WHERE email = $1', [email], (err, results) => {
```

Principle and Patterns we used

Model-View-Controller Architecture

Each service in our project leverages the Model-View-Controller (MVC) architecture to create a structured and maintainable software framework. Here's a brief overview of each component's role:



Model

- Manages data and implements business logic.
- Handles database operations.
- Maintains the application's state.

View

- Renders the user interface.
- Presents data to users.
- Manages user interactions.

Controller

- Acts as an intermediary between Model and View.
- Listens for user input and translates it into actions.
- Controls the flow of the application, which is non-hierarchical in our application.

The MVC patterns help us to separate the concern of output or UI presentation from the user input handling. It largely helps with facilitating the extensibility in the development process. A

new feature or functionality can be added into the model more easily and independently of the other components.

Modularity and Separation of Concern (SoC)

- Modularity: separate the services of the program, into independent deployable, manageable units of software, provide a concise interface to the clients.
- Well-defined boundaries and well-defined interface for clients to access.
- Keep the abstraction high in the service level so that we can reduce coupling.
- Better for us to reuse the service structure.
- We use vertical slicing for the decomposition, since we design the web application by features.
- We use ports and adapters to connect between each service.

Modularity helps us with shorter development time and better flexibility, and by decomposing the big application into smaller chunks as in microservices with well defined APIs.

High Cohesion & Loose Coupling:

We use **Communicational Cohesion** in our project, since we keep the facilities which operate the same data together. And we use **Control Coupling**, for one module can control the other module by passing parameters/messages to tell the other module what to do. For example, if the matching service is successful, it will give in a flag for proceeding into the collaboration service.

We implement **Information Hiding** by separating the parts of the code that may change frequently (frontend/app::page.tsx) from the ones that don't usually change (Services::backend routers/controllers). And the module boundary is stable, making the maintenance safer and easier.

Pub-Sub Structure

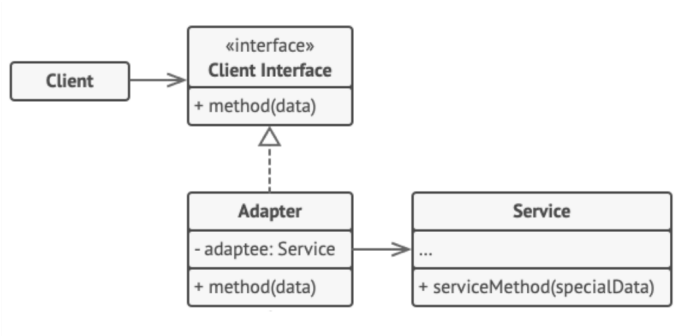
In Matching-service, all users attempting to find a match will publish their request to the shared queue. Request contains details such as username and difficulty level selected. Then, they'd also subscribe to the same queue. If a match has been found, both users would then be popped out of the queue.

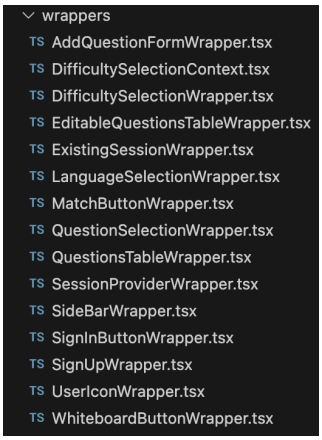
Event-driven Messaging

Upon successful match, both users subscribe to a topic named after an unique Room ID. Events, along with messages, are passed between users through a connection to a common websocket. These events mainly facilitate the communication for selecting questions and ending the session. For example, when one user wishes to end the session, "end-session-request" event will be sent to the topic,

Adapter / Wrapper

By using various Wrappers, we allow the interface of existing classes to be used by other components. Our Wrappers are mainly **Object Adaptors** that don't work with both clients and services. Instead, it implements the client interface, with wrapping the service objects.





Suggestion for Improvement and Enhancement

1. Security:

- Could strengthen our authentication system with multi-factor authentication (MFA) options
- Users should be able to update their profiles with information like username and password.

2. Testing:

- Could implement automated testing procedures to streamline the testing process and identify issues more efficiently.
- Could integrate user feedback into our testing processes. Encourage users to report issues and suggest improvements, creating a feedback loop that enhances the quality of our platform.
- Could Implement robust error handling and logging to aid in debugging and issue resolution.
- We planned to take use of some testing framework like jest to run tests in parallel for speed and generate code coverage.

3. Message Queueing and Event-driven Architecture(Pub-sub):

- Could implement load balancing techniques to distribute message processing across multiple servers to prevent overload on any single server.

- Could enhance the error-handling mechanisms and logging to provide clear insights into message queuing issues so that we can identify and address problems promptly.

4. Kubernetes Optimization:

- Leverage Kubernetes' built-in service discovery mechanisms to facilitate communication between microservices and ensure that message queuing systems are dynamically discoverable.
- Streamline the integration of the CI/CD pipeline with Kubernetes.

5. **Code Smells:**

- There are some parts of duplicate code or lazy classes, particularly in Kubernetes.

Reflection and Learning Point from the Project Process and Group work:

- **Effective Collaboration and Decentralization**

Regular team meetings, status updates, and instant messaging channels allowed us to stay synchronized on project goals and progress. Assigning specific roles and responsibilities to each team member ensured that everyone had a distinct contribution to make. This minimized overlap and ambiguity in tasks.

- **Challenges Faced**

Challenges were inevitable, and task allocation proved to be one. Balancing workloads and ensuring that every member was fully engaged required careful coordination. We also encountered technical hurdles, such as solving the socket issues, which tested our problem-solving skills. Overcoming these challenges often required a combination of research, teamwork, and seeking external expertise.

- **Time Management and Milestones(Sprints)**

The management of project timelines and milestones was crucial and intense. Staying on track and meeting deadlines ensured the project was delivered on time. Meeting milestones enabled us to build the app systematically.

- **Technical Learning and Skill Development**

Peerprep journey expanded our technical proficiency. We had to become adept at containerization with Docker and Kubernetes, integrate external AI services, and design complex user interfaces. The skills acquired during this project are not only beneficial for PeerPrep but are transferable to future projects. These newfound abilities have broadened our skill sets.