Title of Assignment

Project Report

Module

CS3219: Software Engineering Principles and Patterns

AY22/23 Semester 1

Members of Group 42

Blake Tan Ting Yu	A0218235J
Ng Jun Kang	A0218325J
Ryan Aidan Jayasuriya	A0218327E
Yusuf Bin Musa	A0218228E

Due By

9 Nov 2022

Table of Content

1. Background **4**

2. Purpose of the Project **5**

3. About the Team **6**

3.1 Team Philosophy 8

3.2 Modus Operandi 8

4. Development Process Decisions **8**

4.1 Development Process 8

4.1.1 Meeting Agenda 9

4.1.2 Developer Handbook 9

4.1.3 CI/CD Pipeline 9

4.2 Overview of Key Features Implemented 11

5. Key Design Decisions **12**

5.1 Tech Stack 12

5.1.1 Microservices Consideration 14

5.1.2 Deployment Decision 15

5.2 Architecture Diagram 17

5.2.1 Microservice Architecture Diagram 17

5.2.2 4-Tier Architecture Diagram 18

5.2.3 Design Decision for 4-Tier Architecture 19

5.3 User Flow Diagram 20

5.4 API Routes 20

5.4.1 User Service 21

5.4.2 User Authentication Service 21

5.4.3 User Attempt Service 22

5.4.4 Question Service 22

5.5 Database Schemas 23

5.6 PeerPrep Requirements 23

5.6.1 Functional Requirements 24

FR1. User Authentication 24

FR2. User Attempt 25

FR3. Matching Service 25

FR4. Question Service 26

FR5. Room Collaboration Service 27
FR6. Room Communication Service 27
5.6.2 Non-functional Requirements 27
NFR1. Security 28
NFR2. Usability 29
NFR3. Portability 30
5.7 Services Summary and Design Patterns 31
5.7.1 User Service 31
5.7.1.1 Authentication Service 31
5.7.1.2 User Attempt Service 32
5.7.2 Matching Service 32
5.7.2.1 Finding a Match 33
5.7.2.2 Join Queue Given No Initial Match 33
5.7.2.3 Match Found, Joining a Room 34
5.7.2.4 Rejoining an Assigned Room 34
5.7.3 Question Service 35
5.7.4 Room Service 36
5.7.4.1 Collaboration Service 36
5.7.4.2 Communication Service 36
5.7.5 User Interface 37
5.7.6 Design Patterns 40
6. Future Improvements 45
6.1 Online Judge 45
6.2 Viewing Attempt Details 45
6.3 CI with Comprehensive Tests 45
7. Reflections 46
8. Appendix 47
8.1 Security NFR Implementation 47
8.2 Portability NFR Implementation 49
9. Product Screenshots 50

1. Background

The Feynman technique is considered to be one of the best ways of learning and follows a simple 4-step plan.

1. Choose a concept you want to learn about,
2. Explain it to a 12-year-old,
3. Reflect, refine, and simplify, and
4. Organise and review.

To put this into the context of software engineering, one way to test your understanding of a certain concept is to be able to explain it such that a layperson would be able to understand it, and get feedback on what can be improved and iteratively improve on it and test your understanding again.

This is also similar to how technical interviews work.

It's one thing to be able to write effective code, but explaining it in a way that the interviewer can understand, regardless of their technical background, is also important. Services that can help its users improve on their ability to solve coding or algorithmic problems are commonplace. Getting an optimal solution is only part of the journey in the interviewing process of a software engineering job, the other part is the ability to communicate your ideas into code and vice versa.

PeerPrep is a platform that allows its users to match with each other and give them the opportunity to work together on an algorithmic problem. In doing so, users can also simulate interviews by assuming the role of interviewer-interviewee, or work together as a pair to solve a problem.

2. Purpose of the Project

For our users, while we had initially conceptualised a player-versus-player style LeetCode platform as we thought that it would be engaging. After giving it some more thought, however, we feel that a platform like that serves a category of users that don’t really need such a service as it already exists – competitive programmers with online judges.

This brings us to PeerPrep. We recognise that being able to solve algorithmic problems is only one part of the interview process – being able to communicate their ideas to the interviewer is another. We find value in providing this platform to users who would also like to practise communicating their ideas and solutions to other players, or even simulate a proper mock interview.

The team finds value in pursuing this project as we also see ourselves as the users. On top of that, we also believe that through this project, we can hone other skills that are equally as relevant in the field of software engineering such as web development, devops, software patterns, architecture, and design, and software project management.

3. About the Team

This section covers the roles of each member in the team and the contributions of each member, technical and non-technical, as well as the timeline of our development process.

Name	**Role**	**Contribution**
Blake Tan Ting Yu	Backend and Test Engineer	1. Wireframe UI 2. Swagger API documentation 3. Github login 4. Question mode component
Ng Jun Kang	Full-stack Engineer	1. User authentication service 2. Email verification 3. General bug fixing 4. Front-end: Question and Attempt
Ryan Aidan Jayasuriya	Lead Engineer	1. All front-end and UI matters 2. Matching service 3. Room service 4. Collaboration service 5. Communication service 6. Attempt service 7. Sprint planner 8. Linting CI 9. Front-end deployment
Yusuf Bin Musa	Backend Engineer, Project Manager	1. Project organiser 2. Sprint planner 3. Documentation 4. Backend: Question and Attempt 5. Back-end deployment 6. Back-end CD

Sprint	Accomplishments
Week 3	• Project setup
Week 4	• Prototyping • Authentication - Backend
Week 5	• Authentication - Frontend
Week 6	• Matching service - Backend, Frontend
Recess week	• Authentication - Github login • Question - Indexing, serverless deployment, API routes
Week 7	• Deployment - backend R&D • Room - Backend, Frontend • Collaboration - Backend, Frontend
Week 8	• Deployment - Backend test on GAE • Matching - Frontend improvements
Week 9	• Deployment - Backend test on DigitalOcean • CICD - CD for DigitalOcean backend
Week 10	• Collaboration - Fixes • Backlog - Bug fixes
Week 11	• Question - frontend • Room/Collaboration - Link up question service • Attempt (History) - Backend
Week 12	• Collaboration - Fixes • Attempt - Frontend
Week 13	• Communication - Backend, Frontend • Backlog - Final bug fixes

3.1 Team Philosophy

Our team's philosophy when approaching this project is simple – "Optimise for our users and cost, and play to each other's strengths". This philosophy is ultimately what we rely on when prioritising features, designing architectures, and deployment.

3.2 Modus Operandi

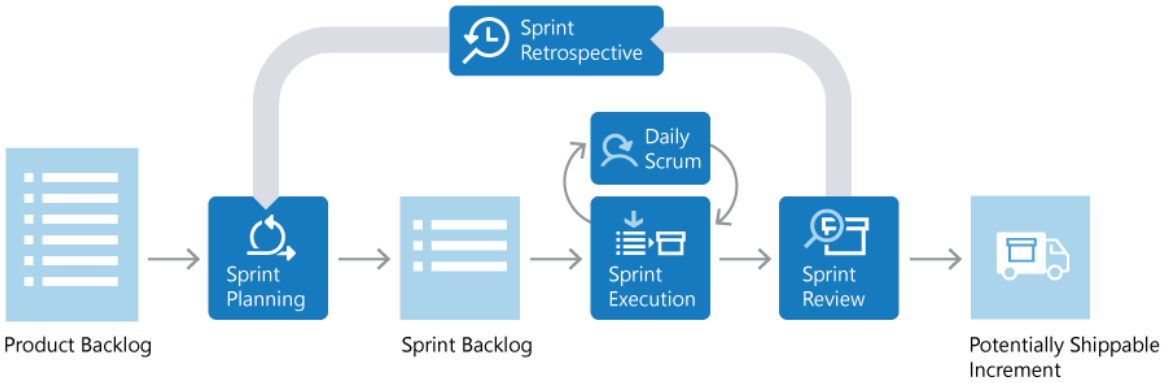
The team adopts open communication and prioritises learning from each other so that we can better hone our skills towards software engineering and what was mentioned in "Section 2".

4. Development Process Decisions

This section goes over the decisions that were taken in the development process such as the meeting agenda and how we operate as a team, onboarding through the developer handbook, and a brief overview of the features implemented.

4.1 Development Process

During the development of PeerPrep, we utilised a modified Scrum framework that would suit the member's schedule as well as combined some elements of Continuous Integration and Continuous Deployment (CI/CD). The image below depicts the scrum lifecycle.



We were able to utilise the framework by using tools like Google Meet as a conferencing platform, Google Calendar to remind the team members of weekly sprints and milestone deadlines, and Github Projects (which the organisation has set

to private) which also links to our Github Issues. In place of daily scrums, our team uses Telegram to update each other on our progress as well as discuss blockers so that we can work on something new for the next sprint. The outline of our weekly meetings can be seen in section 4.1.1.

For the CI/CD portion, we made use of Github Actions to help with simple CI such as linting in case there is an issue with the linter that was mentioned in section 4.1.2. CD was also enforced in the frontend via Vercel and Github Actions for the backend. More details are explained in section 4.1.3.

4.1.1 Meeting Agenda

1. Go over blockers, if any,
2. Refer to backlog in our Github Issues and our main project document, and
    a. Prioritise (1) before new assignments,
    b. Prioritise must-haves before nice-to-haves, and
    c. Parallel development with minimal to no overlaps to prevent merge conflicts
3. Discuss everyone's availability for the week and assign tasks based on (2) and play to everyone's strength to the best of our ability.
    a. New developers are paired with more experienced ones.
    b. New developers get a simple working version before handing over to the experienced developers, or a pair works together on an issue.

4.1.2 Developer Handbook

We've come up with a handbook of coding practices that the team follows to the best of our abilities. This provides a way for people unfamiliar with the language we chose to quickly familiarise themselves with some common practices and would shift discussions away from trivial matters such as naming conventions, and ensure that we can easily understand and work on each others' code.

The handbook is a pinned Github issue and can be viewed here.

4.1.3 CI/CD Pipeline

The purpose of our CI pipeline is to enforce coding standards in our codebase to the best of our abilities by adhering to the developer handbook, and ensuring that the

codebase builds successfully. We have also enforced branch protection in place of unit tests. This means that for the branch to merge to “main”, it requires the approval of 2 people who would test the changes locally and review the pull request for any code smells or bug reports.

For our CD, we have 2 pipelines dedicated for our backend and frontend changes respectively this is done by specifying that a certain Github Action should only trigger changes that have been made to a specific folder, which, in this case, is the backend and frontend directory respectively.

For our backend, we ensured that the Github Action only triggers if changes have been made to the backend directory and only when the pull request merges to main. This is to ensure that the service in production is updated with the latest changes. To summarise how this was achieved, we first set up our DigitalOcean Droplet instance by cloning the Github repository, creating the necessary environment files and SSL certificate. After which, Github Action resets to the current head, pulls the latest changes, and builds and runs the Dockerfile for the backend which exposes the necessary port. The full details can be seen here. We’ve also created a Github Action that we can manually trigger in the case of hotfixes that get pushed to the main branch directly.

In the case of our frontend, we use a combination of Github Actions and Vercel’s automatic deployment. To utilise Vercel’s features, we would need to use a personal Github account, and for that to do so, we would need to fork the group’s repository into a personal account so that it can trigger Vercel’s workflow. To do this, our Github Action in one of our member’s forked repository is triggered when a change to the frontend is detected and causes the forked repository to also update. This then triggers Vercel’s CD to run, and thus, ensures that the latest frontend changes are reflected on the production link, which is also provided by Vercel.

4.2 Overview of Key Features Implemented

Service	**Features**
User	● Authentication ○ Local login ○ Email verification ○ Github login ● Attempt
Matching	● Match according to: ○ Difficulty ○ Topics ○ Question of the Day (Daily LeetCode Question)
Room	● Room ○ Matched users gets connected into the same room ● Collaboration ○ Work on the same document ● Communication ○ By text and audio
Question	● Indexes the questions from LeetCode to ensure availability ● Caches commonly accessed question data

5. Key Design Decisions

This section goes over the design decisions that we took for our application and covers the rationale behind choosing our tech stack, deployment decision, architecture and the likes. We would also be going over the functional (FR) and non-functional requirements (NFR) of each service while explaining why we chose to prioritise the development of each service.

5.1 Tech Stack

We have chosen to use Typescript as the main language for our application as it provides type-safety on top of the easy-to-use Javascript. This greatly helps with debugging even if it comes at a cost of some initial setup.

	Package	**Rationale**
Backend	NestJS (User, Question)	Built on top of Express, extensive middleware support, and enforces various design patterns and functional cohesion.
	nodemailer (User)	Mailing service for email verification.
	Socket.io (Matching, Room)	Real-time bidirectional data communication utilised in the matching, collaboration, and communication services.
	yjs (Room)	Share editor state between matched users.
	zod	Validates inputs before being passed as parameters.

Data	NodeJS (Question)	Supported by Firebase Functions
	PlanetScale (User, Question)	Hosted SQL database
	Prisma (User, Question)	Type-safe NodeJS ORM
	Redis (User, Question)	Cache verification tokens and frequently-accessed question data.
Frontend	monaco-editor	Browser-based code editor
	react-query	Manage remote data in React
	simple-peer	1-1 WebRTC for video/voice and data channels
	tailwindcss	CSS framework
	vite	Frontend tooling
	y-socket.io	Socket IO connector for yjs (backend)
	zustand	React state management

5.1.1 Microservices Consideration

While we had developed PeerPrep with the microservice architecture in mind, we did not follow through with it in deployment.

The microservice architecture allows us to develop faster as microservices are decoupled and isolated from each other, and allows for granular levels of scaling each microservice where needed. However, it was not a good fit to our team's philosophy.

For starters, it does not “optimise for users and cost”.

Given the current resources that we are willing to expand and how we forecast that our application does not need to greatly scale at this stage, we feel like it is not in our best interest to try and optimise for that.

Even though the school does offer reimbursement for the project within the duration of the module, we intend on keeping a deployed version up for at least one year and a microservice architecture would easily exceed the allocated budget or the budget that we were willing to set aside.

Should the time come where we do end up getting a high demand for our service, we consider that a good problem to have and could allocate the necessary resources to scale it when the time comes.

This leads us to the second half of the team’s philosophy – such an infrastructure does not “play to each other’s strengths”.

Coming into this project some of us are still very green to web development and feel like it would be more beneficial to learn and contribute to the code base from the more experienced developers instead of allocating man hours looking into DevOps so that we can coordinate the deployment of the multiple different microservices.

To summarise, the team acknowledges the benefits of the microservice architecture and agreed that it allows for parallel development by allowing the members to work on different parts of the code base easily without much merge conflicts. However, we do not feel that it is in our best interest to expend more resources than we were willing to, to follow through with it in deployment and have chosen to deploy application as a monolithic backend.

5.1.2 Deployment Decision

In this section, we would be going over our choice of where we deployed our application. We opted to deploy our application as a monolithic backend and frontend. The rationale for this was covered in "Section 6.1.1".

For the backend, we were considering the following where "price" is the largest deciding factor and "CI/CD Difficulty" is the lowest.

	Google Cloud Run	**Google App Engine**	**DigitalOcean Droplet**
Price	Free 180000 vCPU seconds per month	Free for the first 28 instance hours of "F" instances	USD18/month for 2 vCPU, 2GB RAM, 60GB SSD
Availability	The first web request is slow as the instance would need to spin up.	Available 24/7	Available 24/7
Scaling	Abstracted out	Configurable based on utilisation metrics	Single instance only
CI/CD Difficulty	Relatively simple with Github Actions	Possible with Github Action, may require more Google services	Possible via SSH from Github Action, might not be the most secure.

During our research, even though Google offers a very tempting free tier of services, how they bill the users gets very confusing as seen highlighted here. While we initially chose to deploy on Google App Engine as it was a good compromise that offered a free tier, is available 24/7, and allows scaling, we went to deploy the backend on a DigitalOcean Droplet instead as it was cheaper than Google's offering of a similar product (Compute Engine), and has a more predictable billing. We also encountered an issue with websockets and scalable compute instances which ultimately made the switch to DigitalOcean more palatable. Additionally, the GitHub student pack offers a USD200 DigitalOcean voucher, which allows us to run the

backend for free for at least 11 months and would only need to pay for the 12th. Thus, we chose to deploy our backend on DigitalOcean Droplet.

Similarly, for the frontend, the following was our consideration.

	Vercel	**Google App Engine**
Price	Free for personal repositories.	Free for the first 28 instance hours of "F" instances
Availability	Available 24/7, also provides readable URLs when deployed	Available 24/7, default URLs are not very readable
CI/CD Difficulty	Vercel has their own CD pipeline for staging and deployment environments.	Unable to find

Scaling is not part of the consideration this time round as we feel like scaling the frontend would not affect our application. We decided to go with Vercel as having their own CD pipeline for different environments would be very useful.

For our database, we have chosen to use PlanetScale as it is a hosted SQL database that provides a generous free tier. This was chosen over a hosted NoSQL service like MongoDB Atlas as the data in our service can be relational and we would like the ability to work with that. This database would be used for the user and question service.

For our matching service, we would be utilising Redis' namespace in place of queues or PubSub topics. Redis also offers a free hosted tier.

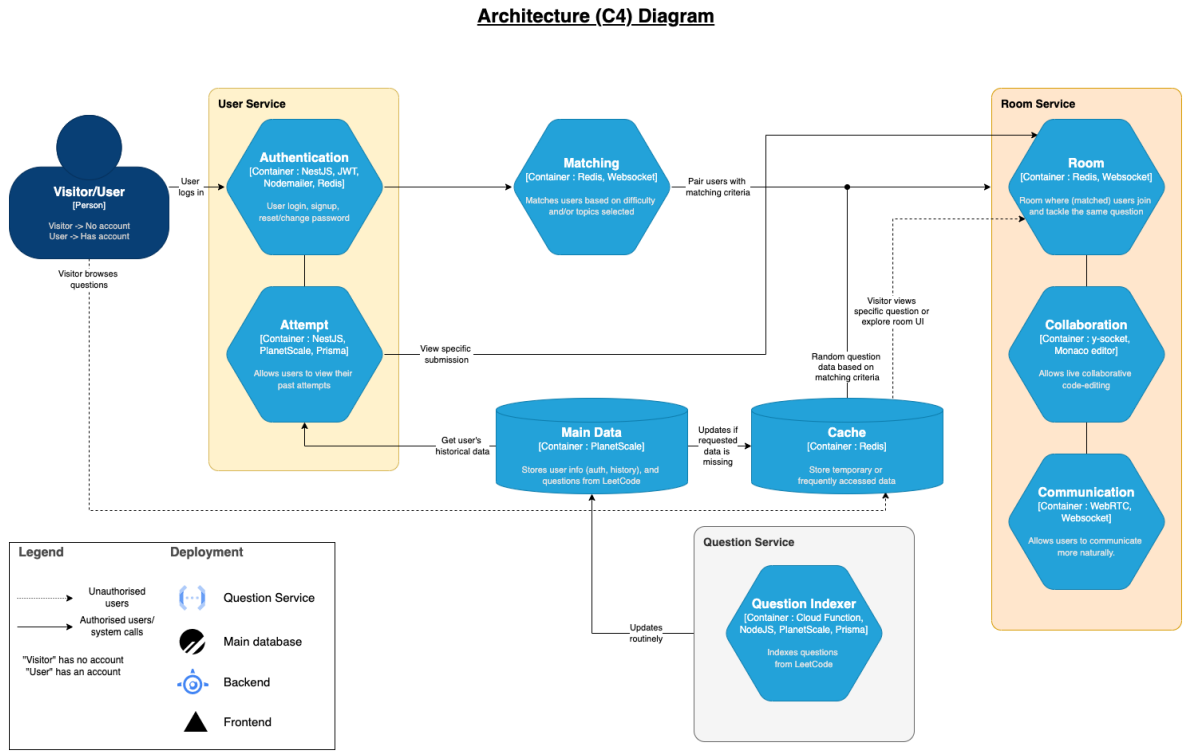
The question services utilises Google's Cloud Functions which offers a generous free tier to populate our question database and update the necessary information.

5.2 Architecture Diagram

The following diagram shows how each service is broken down into their own microservices. We've attached the initial architecture diagram (section 5.2.1) and the finalised architecture diagram (section 5.2.2).

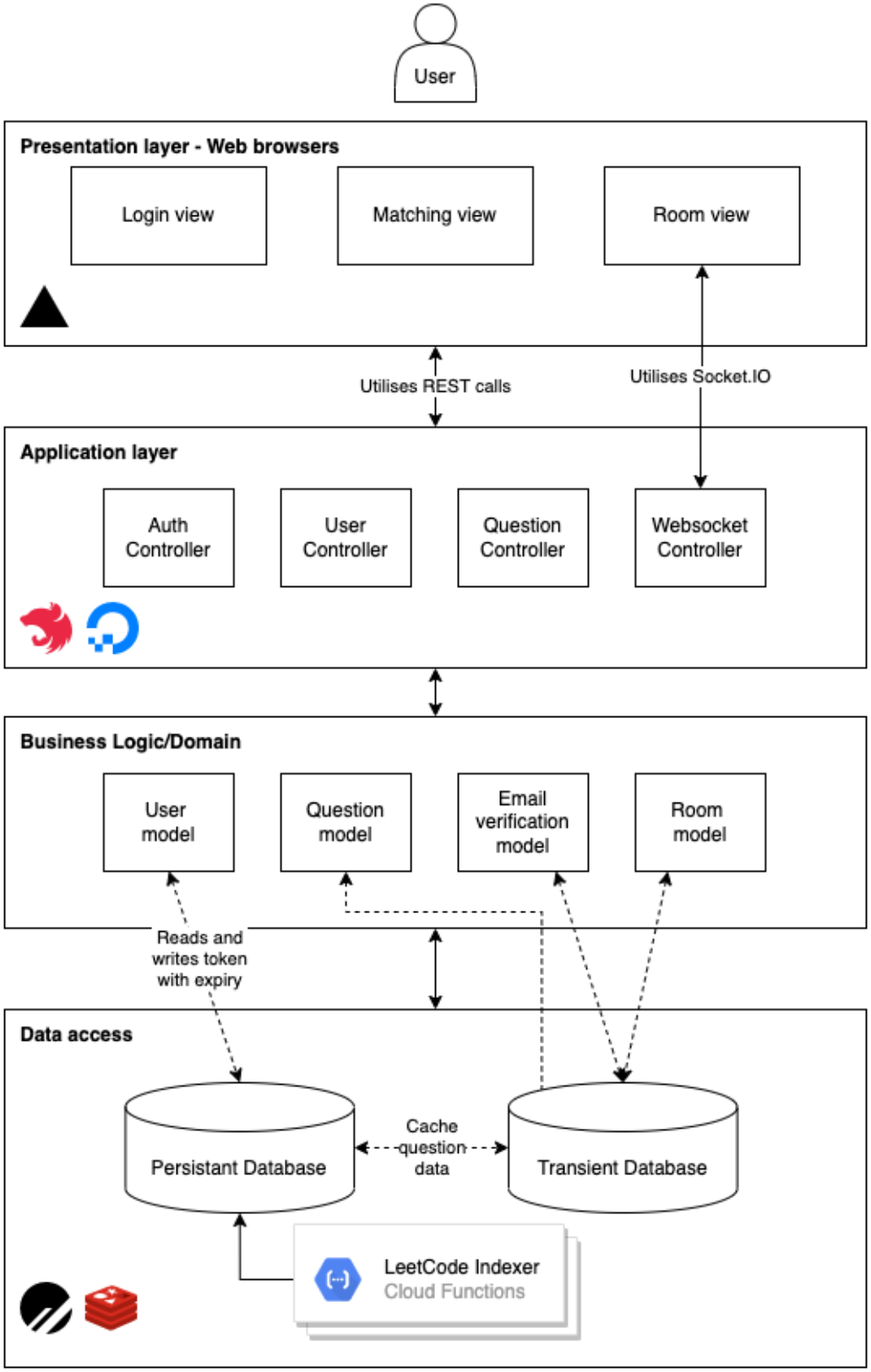
Essentially, the initial architecture diagram is our dream architecture, if there were no constraints, but as we would have it, that is not the case and we ended up making some compromises.

5.2.1 Microservice Architecture Diagram



Bigger version here.

5.2.2 4-Tier Architecture Diagram



Bigger version here.

5.2.3 Design Decision for 4-Tier Architecture

As mentioned in “Section 5.1.1” the team has initially agreed to develop PeerPrep following the microservice architecture, but due to the factors mentioned in “Section 5.1.2”, we went against the microarchitecture specifications and are using a shared database which ultimately shifted our architecture into a 4-tier design. That being said, there are still some benefits of the microservice architecture that are still applicable to the 4-tier architecture design, namely, loose coupling, efficient and quick development, maintainability and extensibility.

To reiterate, the loose coupling allows us to add, modify, or fix features easily without affecting other components, allowing it to be more extensible. This also plays nicely to maintainability as there is a distinct flow within the layers, making it easy for developers to understand and diagnose issues.

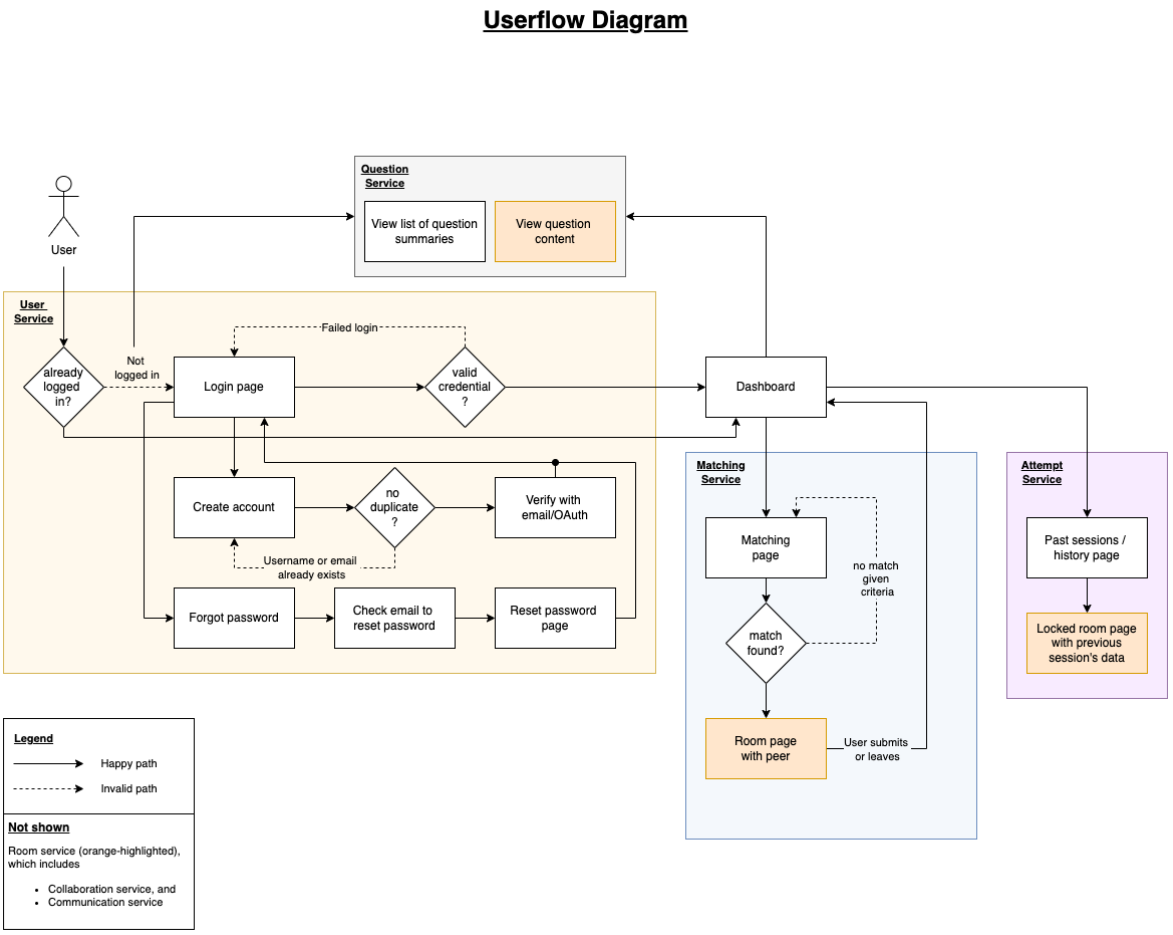
In our case of the architecture diagram, this also grants us an upgrade path with regards to the infrastructure should the demand require it and gradually upgrade to a microservice architecture.

The websocket controller in the application layer can be hosted on a standalone server as it does not play nicely with automated scaling. This means that the remaining business logic can be scaled, managed or otherwise, through Kubernetes or cloud services such as Google’s Cloud Run or App Engine.

Similarly, each domain can consist of its own database instead of sharing a common one which is currently being used.

5.3 User Flow Diagram

The following diagram illustrates how a user would use our application.



Bigger version here.

5.4 API Routes

The following table highlights the endpoints that we have created for PeerPrep, their corresponding service, and a summary of what it is used for.

For the full details of the API, you may refer to our Swagger endpoint for more details, but to test them, we do recommend using Postman via our collection instead.

5.4.1 User Service

Endpoint	Description
GET /users/me	Verifies that the JWT token passed in request is valid.
GET /users/{id}	Returns info of the user with the given ID.
PATCH /users/{id}	Edit data of specified user.
DELETE /users/{id}	Delete data of specified user.

5.4.2 User Authentication Service

Endpoint	Description
POST /auth/change-password	Changes password of the current authenticated user.
POST /auth/delete-account	Deletes user account after verifying the specified password.
POST /auth/forget-password	User requests to change existing password (not logged in).
GET /auth/local/oauth	Users log in via Github.
POST /auth/local/signin	Signs in a user with the provided credentials.
POST /auth/local/signup	Creates a new user with the provided credentials.

GET /auth/refresh	Refreshes the JWT token cookies.
POST /auth/reset-password	The user enters a new password to change their existing password.
POST /auth/signout	Signs the current user out.
POST /auth/verify/{token}	The user verifies their email to sign up.

5.4.3 User Attempt Service

Endpoint	**Description**
POST /attempt	Stores the content of the given attempt into the database and returns a success status.
GET /attempt	Returns a list of all past question attempts.
GET /attempt/{titleSlug}	Returns a list of past attempts that corresponds to the title slug.

5.4.4 Question Service

Endpoint	**Description**
GET /question/content/{titleSlug}	Returns the content of the question with the associated titleSlug.
GET /question/content/daily	Returns the content of the question of the day.
GET /question/summary	Without any parameters, it returns all the summaries of all available questions.

	Available parameters: ● titleSlug, ● difficulty, ● topicTags, ● topicMatch (defaults to AND)
GET /question/summary/daily	Returns the summary of the question of the day.

The question service can be tested using the Swagger UI that we had set up here, or via our Postman collection here.

5.5 Database Schemas

To support these APIs, we have populated our database with the necessary fields. As we've chosen to use an SQL database, PlanetScale, we had to specify our schema which can be seen here.

5.6 PeerPrep Requirements

In this section, we would be utilising the following prioritisation scheme to indicate which functional and non-functional requirements the team worked on.

Priority	**Explanation**
High	Basic functionality needed for the application to function.
Medium	These requirements are also needed for our application and either (1) requires high priority to be completed, or (2) does not require immediate attention.
Low	These features are nice-to-haves or stretch goals.

5.6.1 Functional Requirements

FR1. User Authentication

S/N	Requirement	Priority
FR1.1	The system should allow users to create an account by specifying their email, username (screen name), and password.	High
FR1.2	The system should ensure that each account created has a unique email and username.	High
FR1.3	The system should require users to verify their email on account creation to ensure that they have provided a valid email.	High
FR1.4	The system should allow users to login into their accounts by entering their email and password.	High
FR1.5	The system should allow users to log out of their account.	High
FR1.6	The system should allow users to delete their account.	Medium
FR1.7	The system should allow users to change their password.	Medium
FR1.8	The system should allow users to reset their password by specifying their registered email.	Medium

Refer to section 5.7.1.1 for this service's diagram.

FR2. User Attempt

S/N	Requirement	Priority
FR2.1	The system should allow users to view a summary of their past attempts given a question.	Low
FR2.2	The system should allow users to view a specified previous attempt of a question.	Low

Refer to section 5.7.1.2 for this service's diagram. Due to time constraints, we were not able to implement FR2.2.

FR3. Matching Service

S/N	Requirement	Priority
3.1	The system should allow users to search for a match by specifying the difficulty level of the question they wish to attempt.	High
3.2	The system should allow users to stop searching for a match.	High
3.3	The system should stop searching for a match if a user is not matched within 30 seconds.	High
3.4	The system should match 2 users with the same matching criteria.	High
3.5	The system should allow users to specify their matching criteria.	Low

In this service, we would like to elaborate that "matching criteria" as mentioned in FR3.4 can apply to question difficulty, topic, or question of the day.

Refer to section 5.6.2 for this service's diagram.

FR4. Question Service

S/N	Requirement	Priority
4.1	The system should ensure that matched users are served the same question based on their matching criteria.	High
4.2	The system should allow users to view the summary of questions.	High
4.3	The system should ensure that the summary of questions is updated.	High

To provide the latest information to our users, we utilised scheduled serverless functions that index and update question data from LeetCode directly into our database. There are several benefits in doing so.

First, as mentioned, our service is able to serve the latest questions that are added to LeetCode as well as ensure that their summary information is updated such as their topic tags and acceptance rate as these may determine whether a user would attempt the question.

The other benefit to doing this is that we reduce coupling to LeetCode's question endpoint. In the event where LeetCode changes this endpoint, our service would still be able to serve users with relatively updated questions. Fixing this issue would also be as straightforward as changing the endpoint that the serverless function queries, assuming that the shape of the data remains the same.

Lastly, this greatly improves the response time (which can be tested here) for users to obtain a question.

Refer to section 5.6.3 for this service's diagram.

FR5. Room Collaboration Service

S/N	Requirement	Priority
5.1	The system should ensure that the user can work on the same document.	High

Refer to section 5.6.4.1 for this service's diagram.

FR6. Room Communication Service

S/N	Requirement	Priority
6.1	The system should ensure that users can communicate to each other by text.	High
6.2	The system should ensure that users can communicate to each other by audio.	Low
6.3	The system should allow users to communicate to each other with video.	Low

Refer to section 5.6.4.2 for this service's diagram.

5.6.2 Non-functional Requirements

The non-functional requirements (NFRs) are categorised under the following labels from highest to lowest priority.

1. Security
2. Usability
3. Portability

We have ranked the following NFRs as such as it is in line with our team's philosophy. We've also included some code snippets and examples to show how the NFRs are met in section 8.

Security is of the highest priority as our lives have shifted towards the new normal of being online, schools, businesses, and personal lives alike. As such, the team has

agreed that it is to the best interest of all stakeholders that we should adhere to the best security practices as much as possible. This gives our users the peace of mind knowing that their credentials would not be accessible to bad actors.

We have ranked usability second as we would, again, like to provide our users with the best experience while using our application. Without a large user base, the key features of our application such as its collaboration and communication services may not be utilised as much as we had hoped. Hence, by making our service more usable, we hope that our users would recommend this service to their fellow peers, and them, theirs so as to ensure an organic growth of users on our platform.

In line with usability, we would also like to ensure that our service is portable. The goal is to ensure that our service is client-agnostic. This means that we would like to allow our users to access our service from any reasonable device.

NFR1. Security

S/N	**Requirement**	**Priority**
1.1	Passwords stored in our database should be hashed. *Rationale*: Ensure our users that their plain-text passwords are not compromised in the event of a breach.	High
1.2	User account creation should ensure passwords should be at least 8 characters long. *Rationale*: Ensure a level of password complexity.	High
1.3	Tokens should be generated with a secret key. *Rationale*: Prevent unauthorised token generation and MITM attacks.	High
1.4	Access and refresh tokens should expire after 15 minutes and 1 week respectively.	High

	Rationale: Reduces the attack window a bad actor has in the event that they have their hands on a valid token.	
1.5	Sensitive requests pertaining to a user's account require re-authentication. *Rationale*: Ensures that bad actors cannot act on a user's account whether they have their access token or physical access to their machine.	High
1.6	Only authenticated users can access PeerPrep services.	High

Refer to section 8.1 for the verification of this NFR.

NFR2. Usability

S/N	**Requirement**	**Priority**
2.1	Users should be able to refresh their access tokens. *Rationale*: Allow users to stay logged in for as long as the refresh token is valid.	High
2.2	Users with overlapping matching criteria should be matched within 5 seconds of the latest user looking for a match.	Medium
2.3	Users should be able to tell how much time they have left while finding a match.	Medium

Refer to section 5.7.5 for screenshots of our user interface.

NFR3. Portability

S/N	Requirement	Priority
3.1	Users should be able to access PeerPrep on any desktop browser.	High
3.2	Users should be able to access PeerPrep on any desktop OS.	High

For NFR 3.1, the team has ensured that the application is accessible from most common web browsers such as Firefox, Google Chrome, and Microsoft Edge. There is an issue with Safari, the default web browser for MacOS that we were not able to address.

For NFR 3.2, we have ensured that users can use PeerPrep regardless of their OS as well as the OS of who they matched with, namely MacOS and Microsoft Windows. We initially found an error in our cross-platform collaboration service which causes the MacOS user to freeze but not the Windows user. We've isolated it to be a line break issue between OSes, pushed a fix in our codebase, and ensured that PeerPrep now better supports cross-platform.

Refer to section 8.2 for the supporting code snippet.

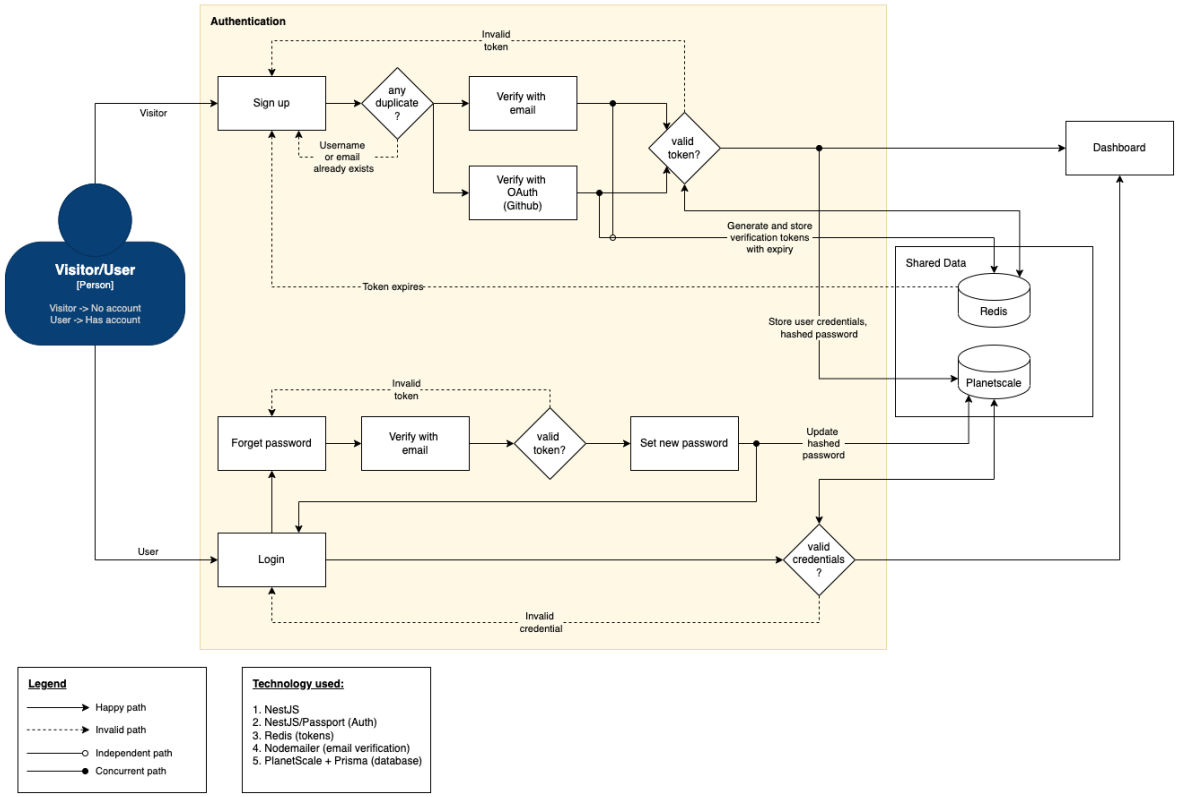
5.7 Services Summary and Design Patterns

This section goes over how each service functions in more detail. As mentioned in section 5.1, we chose to use NestJS for our backend as the generated boilerplate enforces various design patterns and functional cohesion.

Refer to section 5.7.6 for the discussion on how several design patterns were used during the development of PeerPrep.

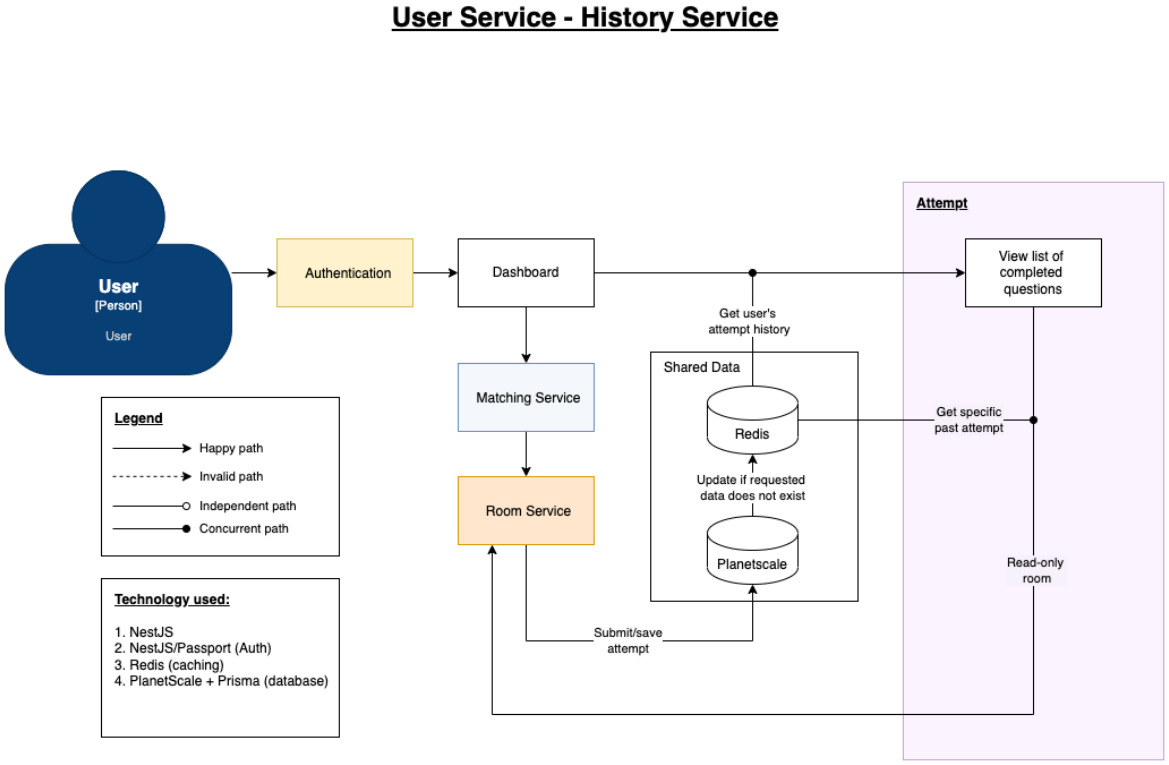
5.7.1 User Service

5.7.1.1 Authentication Service



Clearer version here.

5.7.1.2 User Attempt Service



Bigger version here.

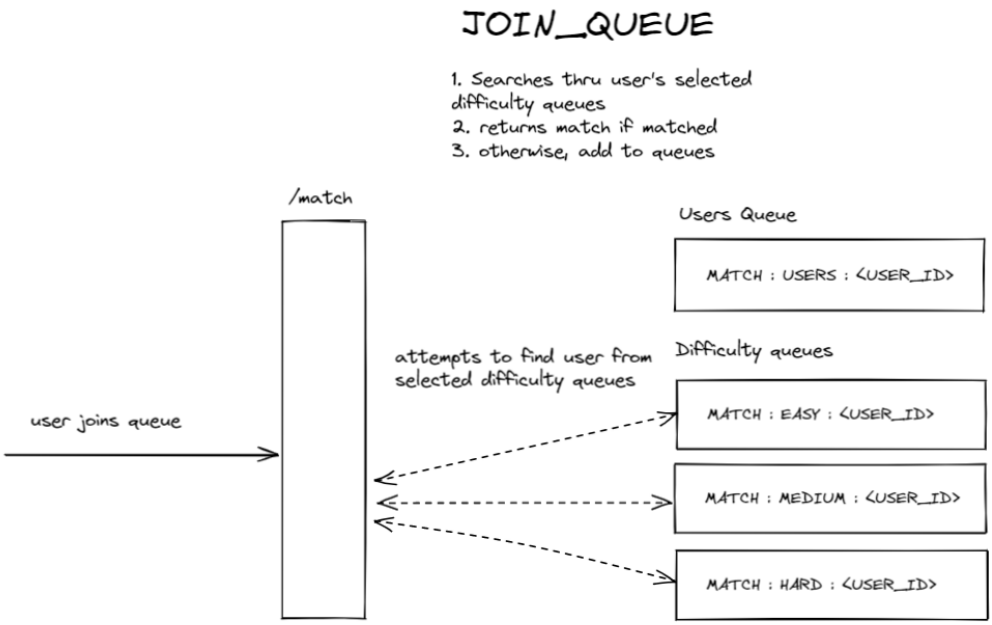
5.7.2 Matching Service

This section goes over the following events related to finding a match on PeerPrep.

1. Finding a match,
2. Join a queue given that no match is found,
3. Match found, joining room, and
4. Rejoining an assigned room.

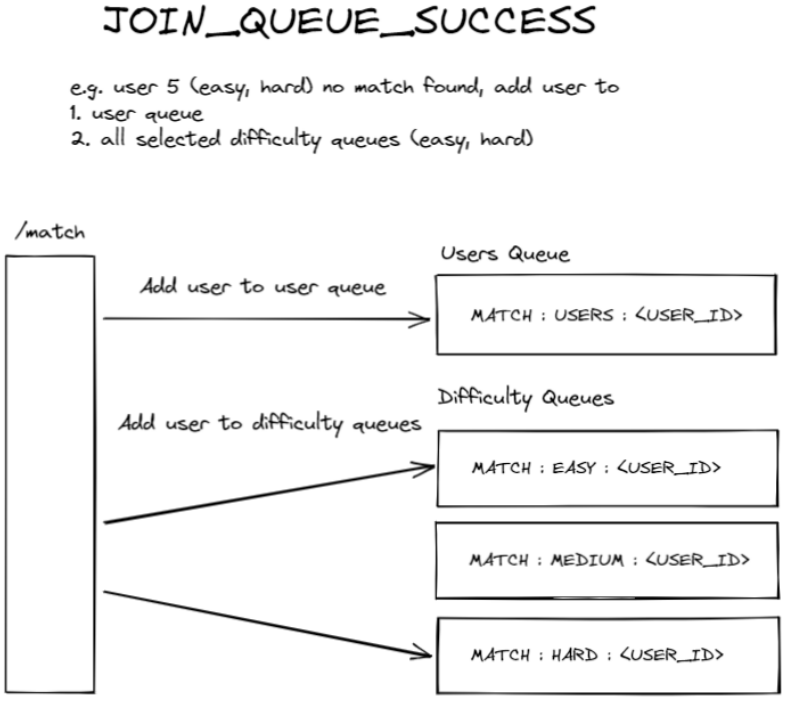
The matching services Redis' namespace and functions similarly to topics in message queues.

5.7.2.1 Finding a Match



This scenario simulates when a new user begins looking for a match. A bigger version here.

5.7.2.2 Join Queue Given No Initial Match

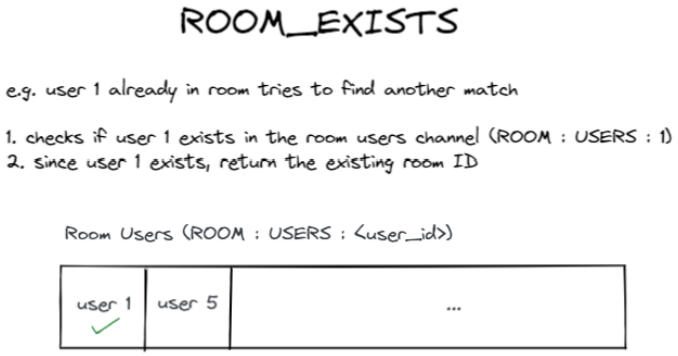


If a new user does not find a match initially (section 5.7.2.1), it joins the respective queues to allow near-instant matching for new users that have an overlapping matching criteria. A bigger version here.

5.7.2.3 Match Found, Joining a Room

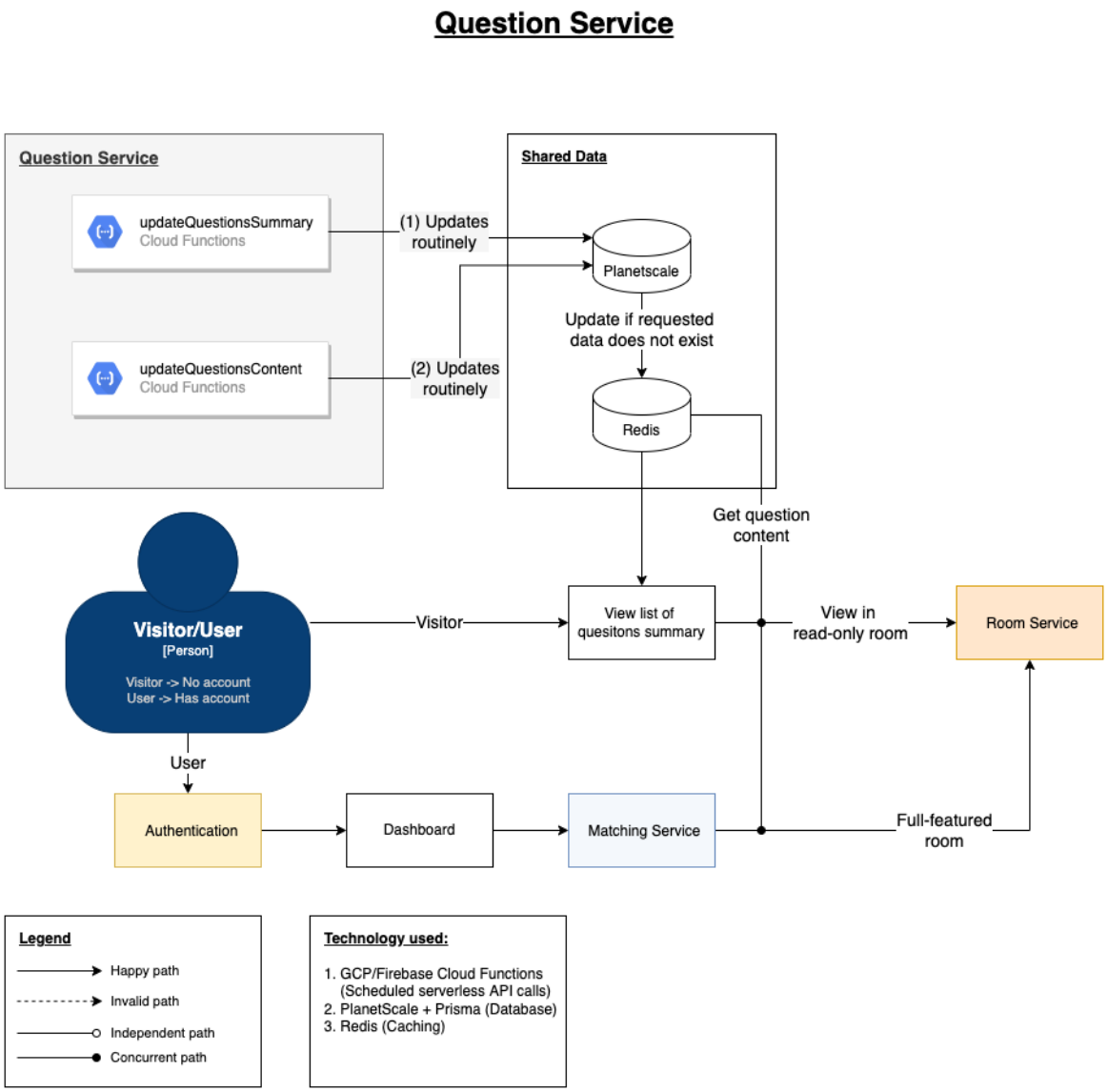
Following 5.7.2.1 and 5.7.2.2, if a match is found, the matching users leave their respective queues and enter a room with a random, unique identifier. A bigger version here.

5.7.2.4 Rejoining an Assigned Room



If a user is already assigned to a room but still looks for another match, it would be brought to the room it was initially assigned to. This may happen in a case where a user accidentally leaves a room and wants to join it again. A user would need to explicitly disconnect from the room before they can search for another room as seen in 5.7.2.1. A bigger version here.

5.7.3 Question Service



Bigger version here.

5.7.4 Room Service

For this section, we would like to highlight that there are 2 versions of room service. The first is a web-socket enabled room which matched users would use, allowing the collaboration and communication service. The other is a room that does not have web-sockets enabled. This version serves visitors of the website and allows them to browse our user interface without needing an account. Users would also use this version of the room as a way to view their specific past attempts, however, due to time constraints, we were not able to implement this portion of the attempt service.

After being matched, users are sent the details of the newly created room. After which, both users are then notified via a pop-up and given the option to join the room. To join the room, the user sends the room websocket gateway server its own user ID as well as the ID of the newly created room. If the information is valid, the websocket server adds the user to the room and notifies all other room users of the newly joined user. The user is then redirected to the "/room/id" page using the room ID of the joined room.

5.7.4.1 Collaboration Service

The collaboration service utilises "yjs" as a way to share data between users in a peer-to-peer manner. This framework was chosen as it also handles the conflict-free replicated data types to prevent conflicts from occurring when users edit on the same data. With "y-monaco", this allows us to use "yjs" alongside the monaco editor and allow users to collaborate on the same document.

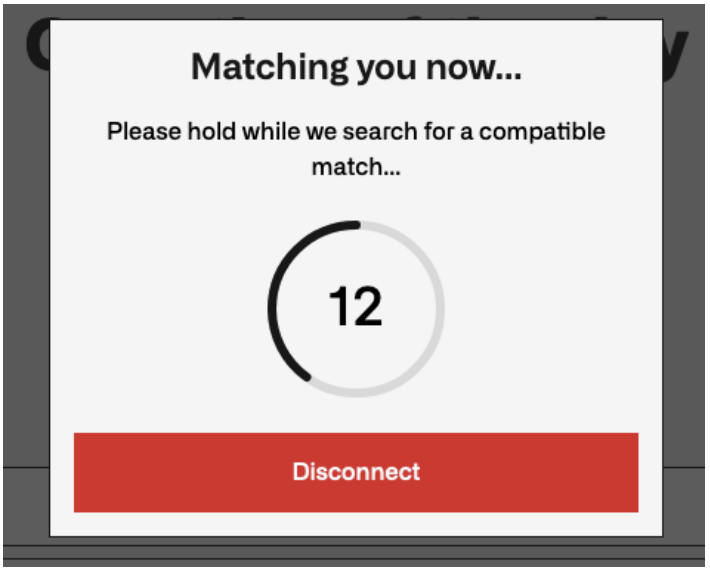
5.7.4.2 Communication Service

Similarly, after the users join the room, they can establish a connection with each other and communicate with each other via video and audio as we utilise "simple-peer" and WebRTC to send the audio and video data to each other. We did not choose to implement a "text" communication service as we feel that the collaboration service is more than sufficient by utilising comment blocks, which can also improve text-only discussions by commenting around code chunks that need to be discussed, akin to commit changes on Github.

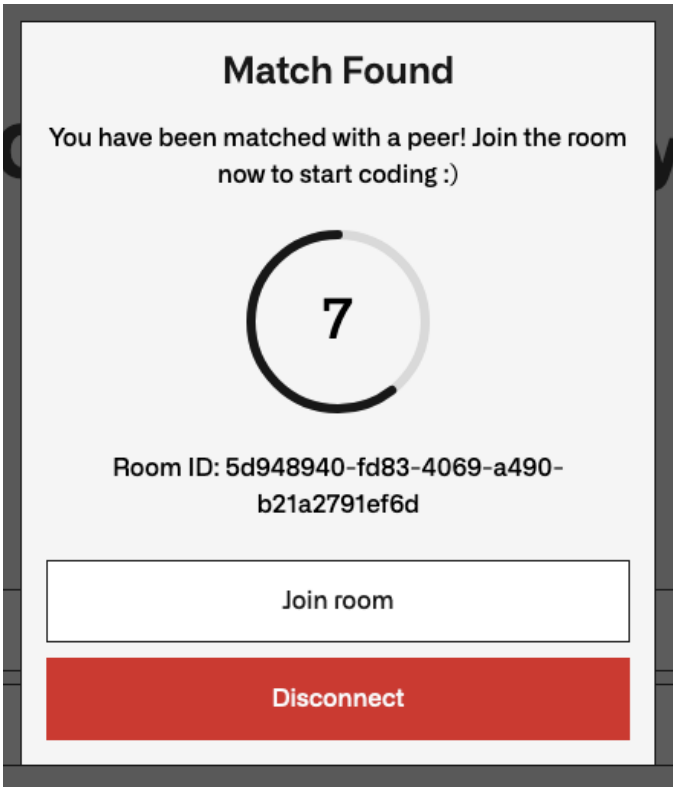
5.7.5 User Interface

With reference to section 5.6.2 NFR2, the following are some screenshots of the UI that we have implemented to improve the user's experience.

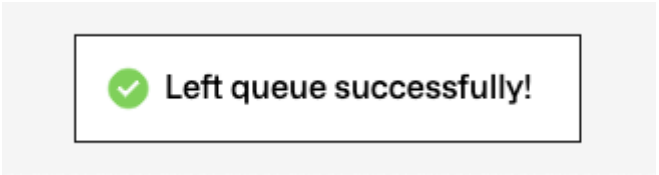
Timer to indicate the remaining time left to search for a match.

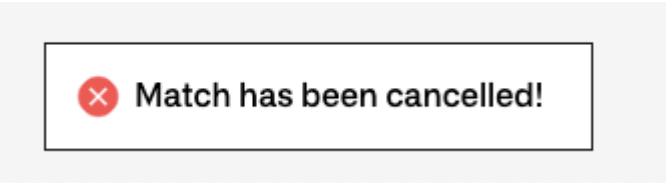


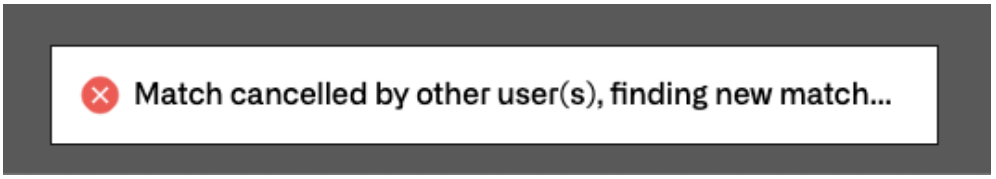
Interface for a user to explicitly accept or decline a match with a 10 second default to decline the match.



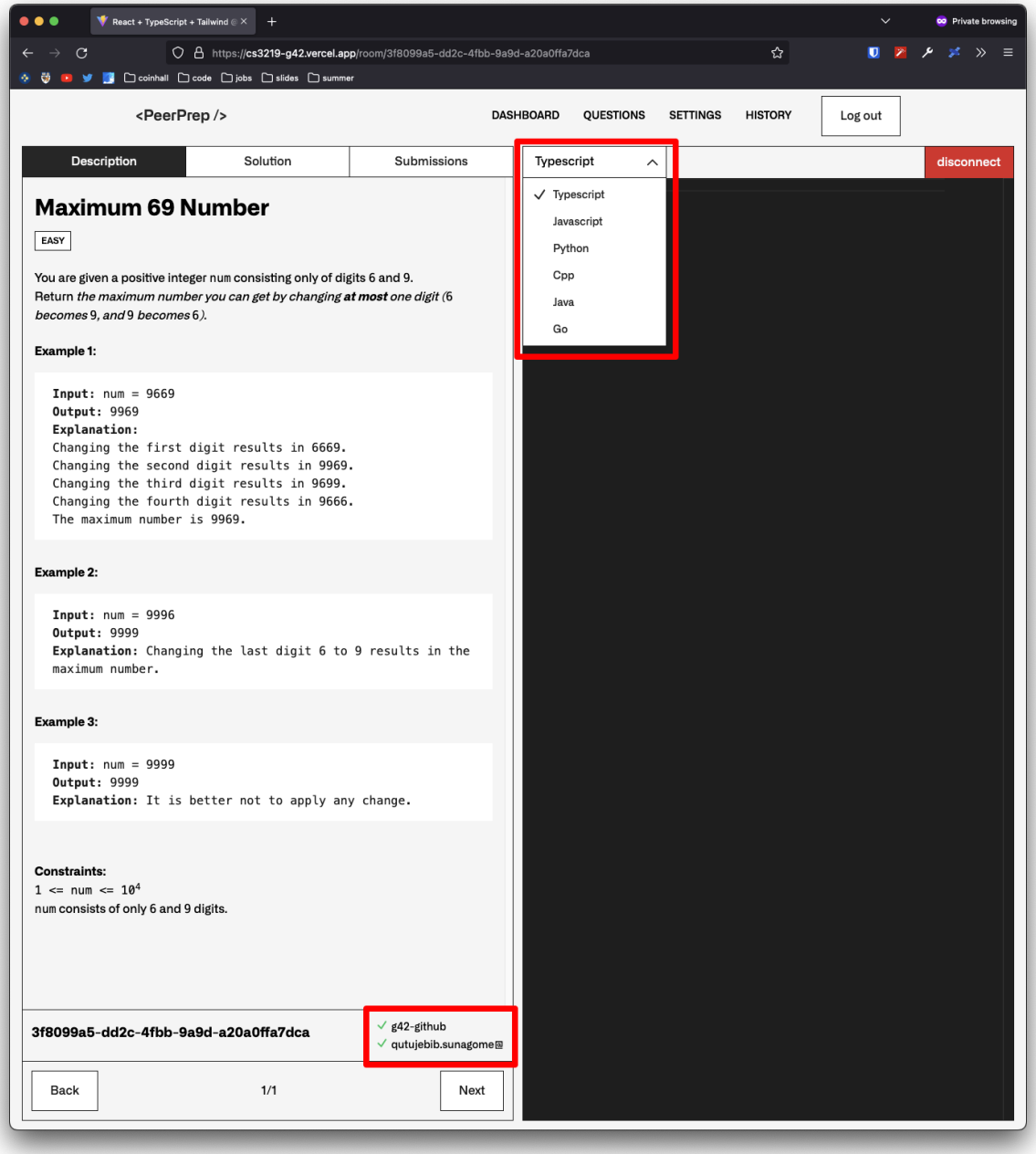
Toast message to indicate that the user has left the search queue due to a timeout or if they have explicitly clicked on the “disconnect” button, or whether the user, or their partner, has declined a match.







Room UI with a split view of the question and editor, an indicator to show who the other user is, an explicit disconnect button to prevent accidentally leaving a room as well as a language dropdown selector which affects the syntax highlighting of the editor.



5.7.6 Design Patterns

As mentioned earlier, one of the many benefits of using NestJS is that it allows us to utilise different design patterns. In our scenario we have utilised the following.

1. Data Transfer Objects (DTOs)
2. Decorators
3. Facade
4. Singleton
5. Strategies

To begin, we would first demonstrate how DTOs and Decorators are used in the controller files to determine if a route is accessible to the public and what data is expected. In this example, we would be going over the sign in process.

g42-proj - auth.controller.ts

```
@PublicRoute()
@Post("/local/signin")
async signin(
  @Body() credentials: SigninCredentialsDto,
  @Res({ passthrough: true }) res: Response
): Promise<SigninResponse> {
  const tokens = await this.authService.signin(credentials);
  this.setCookies(res, tokens);
  return { message: "success" };
}
```

In this snippet, we can see that the “signin” method was decorated with the “@PublicRoute()” to indicate that the route is accessible to everyone. Without that decorator, only users who are signed in would be able to access the route. The decorator is as follows.

g42-proj - public-route.decorator.ts

```
export const PublicRoute = () => SetMetadata("isPublicRoute", true);
```

We then specify that the body of request sent to “/local/signin” should follow the shape as seen below. We then utilise the “@IsString()” decorator to ensure that the email and password provided is a non-empty string.

g42-proj - auth.dto.ts

```
export class SigninCredentialsDto extends createZodDto(SigninApi) {
  @IsString({ each: true })
  public email: string;

  @IsString({ each: true })
  public password: string;
}
```

Following that, the next example shows how the strategy design pattern applied via the “AccessJwtStrategy” is used to secure the application such that all routes require a valid JWT token unless the “@PublicRoute()” decorator is applied to it.

g42-proj - app.module.ts

```
@Module({
  // ... Omitted code ...
  providers: [
    { provide: APP_PIPE, useClass: ZodValidationPipe },
    { provide: APP_GUARD, useClass: JwtAccessGuard },
  ],
})
export class AppModule {}
```

g42-proj - access.guard.ts

```ts
@Injectable()
export class JwtAccessGuard extends AuthGuard("jwt") {
  constructor(private reflector: Reflector) {
    super();
  }

  canActivate(
    context: ExecutionContext
  ): boolean | Promise<boolean> | Observable<boolean> {
    // Set by @PublicRoute decorator
    const isPublic = this.reflector.getAllAndOverride("isPublicRoute", [
      context.getHandler(),
      context.getClass(),
    ]);
    if (isPublic) {
      return true;
    }
    return super.canActivate(context);
  }
}
```

```ts
@Injectable()
export class AccessJwtStrategy extends PassportStrategy(Strategy, "jwt") {
  constructor(config: ConfigService, private users: UserService) {
    const secret = config.getOrThrow("JWT_SECRET");
    super({ /* Omitted code */ });
  }

  async validate(payload: JwtPayload) {
    const [err, user] = await this.users.find({
      id: payload.sub,
    });
    if (err) {
      throw err;
    }
    if (!user) {
      return null;
    }
    return user;
  }
}
```

The facade pattern can be seen between the controller and service files as seen in the question service. From the controller's perspective, it only needs to make a simple call to the "getAllTopics()" method from the question service, but within the question service itself is a little more complicated.

g42-proj - question.controller.ts

```
@PublicRoute()
@Get("/topics")
async getAllTopics(): Promise<GetAllTopicsResponse> {
  const topics = await this.questionService.getAllTopics();
  return topics;
}
```

```
@Injectable()
export class QuestionService {
  constructor(
    private prisma: PrismaService,
    private cache: RedisCacheService
  ) {}
  /** Omitted code */
  async getAllTopics() {
    const topics = await this.getAllTopicsFromCache();
    return topics;
  }

  async getAllTopicsFromCache(): Promise<string[]> {
    const cachedTopics = await this.cache.getKeyInNamespace<string[]>(
      [NAMESPACES.QUESTIONS], QUESTION_TOPICS
    );
    if (!cachedTopics) {
      const res = await this.prisma.topicTag.findMany({
        select: { topicSlug: true },
      });
      const flattenedTopics = res.map((v) => v.topicSlug);
      await this.cache.setKeyInNamespace(
        [NAMESPACES.QUESTIONS], QUESTION_TOPICS, flattenedTopics
      );
      return flattenedTopics;
    }
    return cachedTopics;
  }
  /** Omitted code */
}
```

The singleton pattern is enforced by NestJS by default. By ensuring that the modules are imported by the main “AppModule”, users (developers) are able to access the same single instance in any other module as seen in the example above where “QuestionService” is able to use the same single instance of “PrismaService” and “RedisCacheService”.

g42-proj - app.module.ts

```
@Module({
  imports: [
    ScheduleModule.forRoot(),
    ConfigModule.forRoot(),
    MailerModule,
    RedisCacheModule,
    PrismaModule,
    UserModule,
    QuestionModule,
    AuthModule,
    RoomModule,
    DocumentModule,
    MatchModule,
    HistoryModule,
  ],
  providers: [
    /** Omitted */
  ],
})
export class AppModule {}
```

6. Future Improvements

Despite pushing ourselves to develop an application that would address the main pain point of providing a platform for users to collaboratively work on algorithmic problems, we had to make a lot of assumptions along the way. As such, we feel like our service can be improved with the following features.

6.1 Online Judge

This was a feature that the team initially wanted to try and implement but was not able to due to time constraints. An online judge would be useful as it can determine the correctness of the code. The service that we wanted to utilise was Judge0 which has a free hosted tier as well as a self-hosted version.

6.2 Viewing Attempt Details

Due to the time constraint, our team was not able to fully implement the user attempt service that we had originally planned. The current version allows users to only view when they had attempted the question despite our database storing the content of their attempts.

6.3 CI with Comprehensive Tests

As mentioned in section 4.1.3, our CI pipeline is currently only being used to enforce coding standards in our codebase and testing is done manually on each PR. What we wish we could have done was to take a more test-driven approach where possible. This would ensure that changes to existing code such as optimisation would still give the same results.

7. Reflections

This assignment has given us the opportunity to better ourselves as software engineers and improve our technical as well as non-technical skills.

For starters, this assignment allowed us to explore the entire web development workflow while also offering us the flexibility to decide on the tech stack and software design patterns we choose to implement. While making these decisions, we got to weigh their benefits and realise how useful these patterns are in allowing us to not only improve the development process but also communicate ideas within the group.

Building on communicating ideas, we believe we have improved on how to communicate with each other on a professional level through the course of this assignment. Our group had good chemistry and every member was a great team member. While we did face some disagreements along the way, it did not escalate to any conflict as we kept an open mind and were receptive to feedback and criticism. Additionally, we were not afraid to clarify our doubts and ask for help in a constructive way.

The key problem that we faced during this assignment mainly stemmed from the inexperience in web development. While the back-end was easier to pick up, the majority of the team struggled with building a frontend that would provide a good user experience, which ultimately took a lot of engineering hours away from building as it was spent on learning.

Despite this, we are glad that in the past few weeks, our group managed to successfully develop PeerPrep with an impressive user interface and various features.

To summarise, the key takeaway from this group project was the choice of software design and architecture. We were able to learn a lot by researching the different software design patterns and architecture, and deciding on the appropriate pattern for each use case and different situation as discussed in sedition 5.2. Additionally, despite the main issue that we faced, we are glad that in that struggle, we were able to abide by the team's philosophy and meet the purpose of the project.

8. Appendix

8.1 Security NFR Implementation

NFR 1.1 and 1.2 is enforced with the following code block.

g42-proj - auth.service.ts

```
const hash = await argon2.hash(password);
```

g42-proj - schema.ts

```
const passwordZodString = z
  .string()
  .min(8, { message: "Password must be at least 8 characters" });

const SignupSchema = z.object({
  username: _UserModel.shape.username,
  email: _UserModel.shape.email,
  password: passwordZodString,
});
```

NFR 1.3 and 1.4 are ensured with the following code block.

g42-proj - auth.service.ts

```ts
async signTokens(id: number, email: string): Promise<Tokens> {
  const jwtSecret = this.config.getOrThrow("JWT_SECRET");
  const refreshSecret = this.config.getOrThrow("JWT_REFRESH_SECRET");
  const payload: JwtPayload = {
    sub: id,
    email,
  };
  const [accessToken, refreshToken] = await Promise.all([
    this.jwt.signAsync(payload, { expiresIn: "15m", secret: jwtSecret }),
    this.jwt.signAsync(payload, { expiresIn: "7d", secret: refreshSecret }),
  ]);

  return {
    access_token: accessToken,
    refresh_token: refreshToken,
  };
}
```

NFR 1.5 is shown used in multiple places in the code base but the following example shows how we enforced authentication when executing credential-related actions.

g42-proj - auth.service.ts

```ts
async deleteAccount(id: number, password: string) {
  const [err, user] = await this.users.find({ id, includeHash: true });

  ThrowKnownPrismaErrors(err);

  if (!user) {
    throw new ForbiddenException(AUTH_ERROR.INVALID_CREDENTIALS);
  }

  const isPasswordCorrect = await argon2.verify(user.hash, password);
  if (!isPasswordCorrect) {
    throw new ForbiddenException(AUTH_ERROR.INVALID_CREDENTIALS);
  }

  const [error, userToDelete] = await this.users.delete(id);
  if (error || !userToDelete) {
    throw new ForbiddenException(AUTH_ERROR.UPDATE_ERROR);
  }
}
```

NFR 1.6 is ensured by utilising a custom decorator in NextJS to define publicly accessible routes.

g42-proj - auth.controller.ts

```
@PublicRoute()
@Post("/local/signin")
async signin(
  @Body() credentials: SigninCredentialsDto,
  @Res({ passthrough: true }) res: Response
): Promise<SigninResponse> {
  const tokens = await this.authService.signin(credentials);
  this.setCookies(res, tokens);
  return { message: "success" };
}
```

g42-proj - public-route.decorator.ts

```
export const PublicRoute = () => SetMetadata("isPublicRoute", true);
```

8.2 Portability NFR Implementation

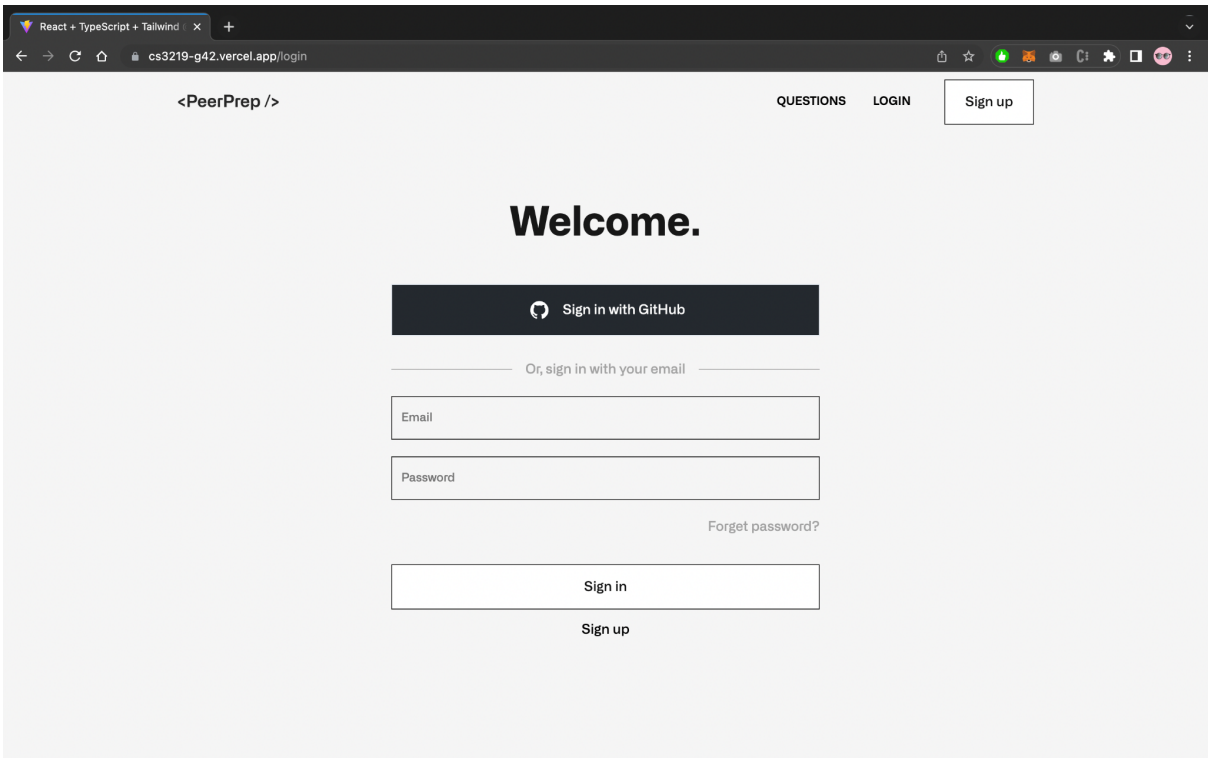
NFR 3.2 on cross-platform compatibility is ensured by the following line of code. Without it, a user from a different OS may get frozen in the room.

g42-proj - createEditorSlice.ts

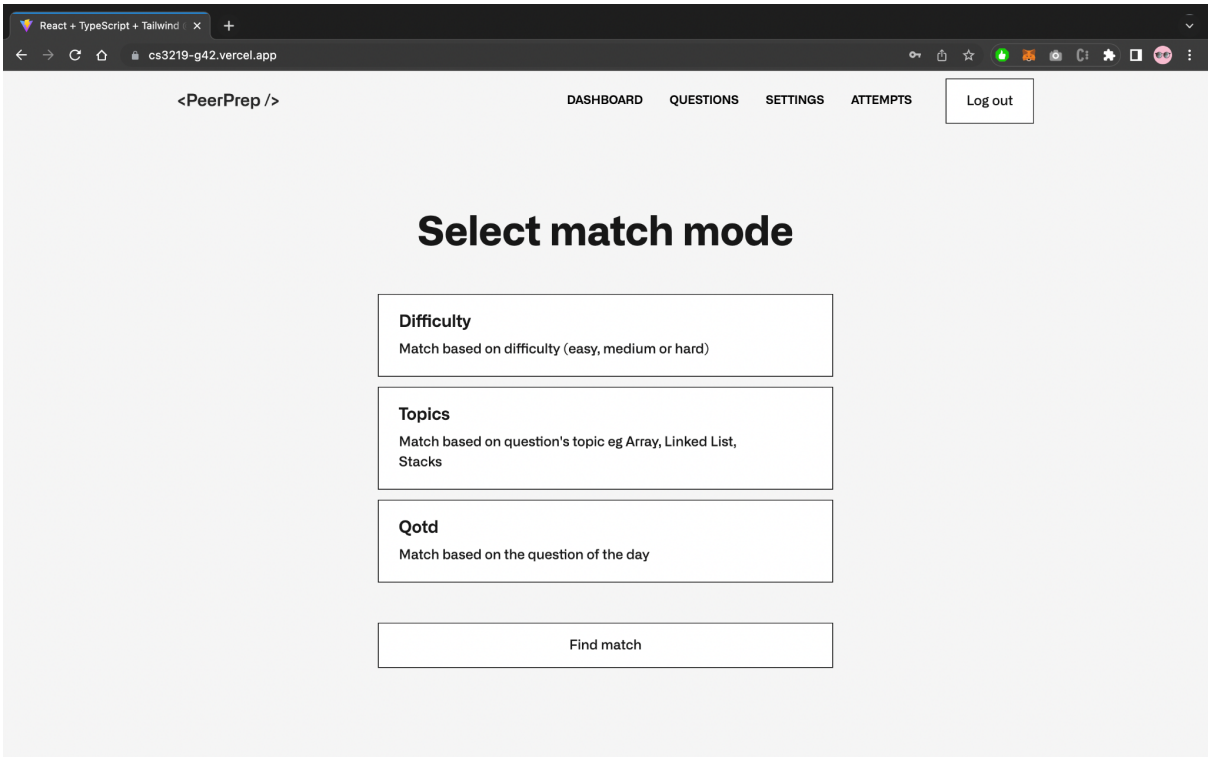
```
model.setEOL(0);
const monacoBinding = new MonacoBinding(
  docText,
  /** @type {monaco.editor.ITextModel} */ model,
  new Set([editor]),
  provider.awareness
);
```

9. Product Screenshots

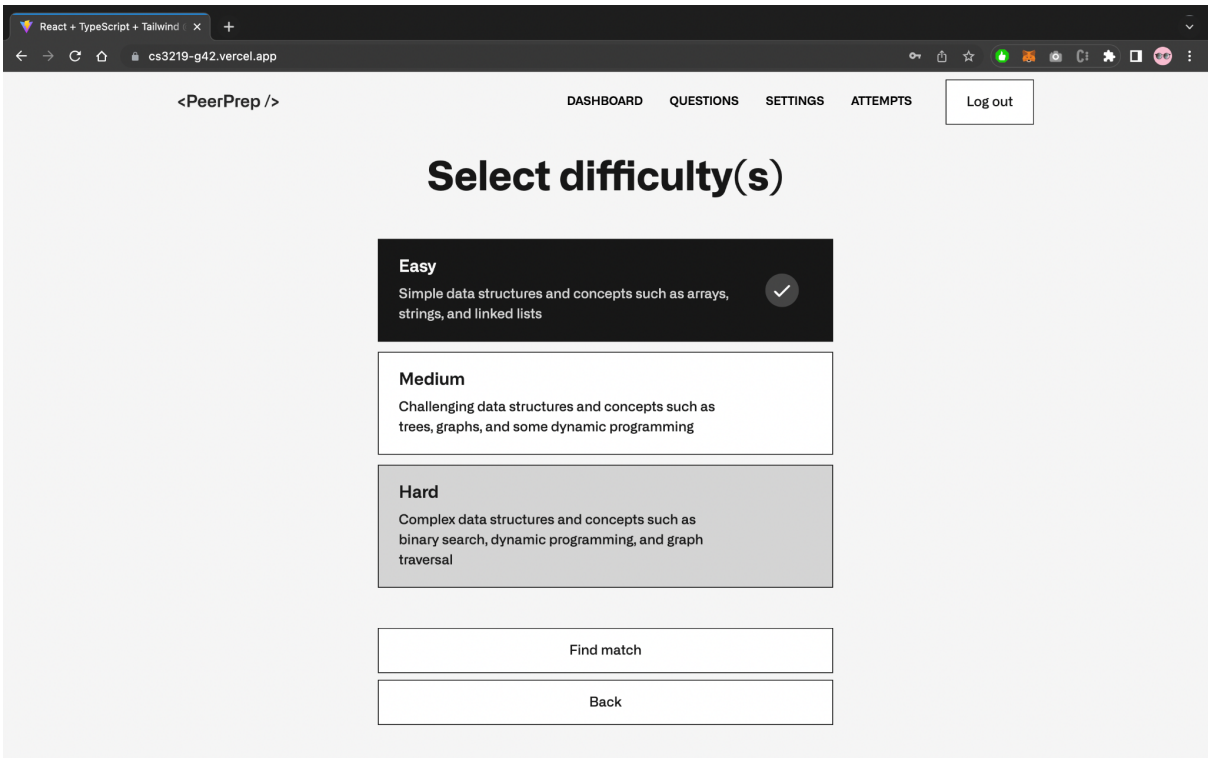
Landing page



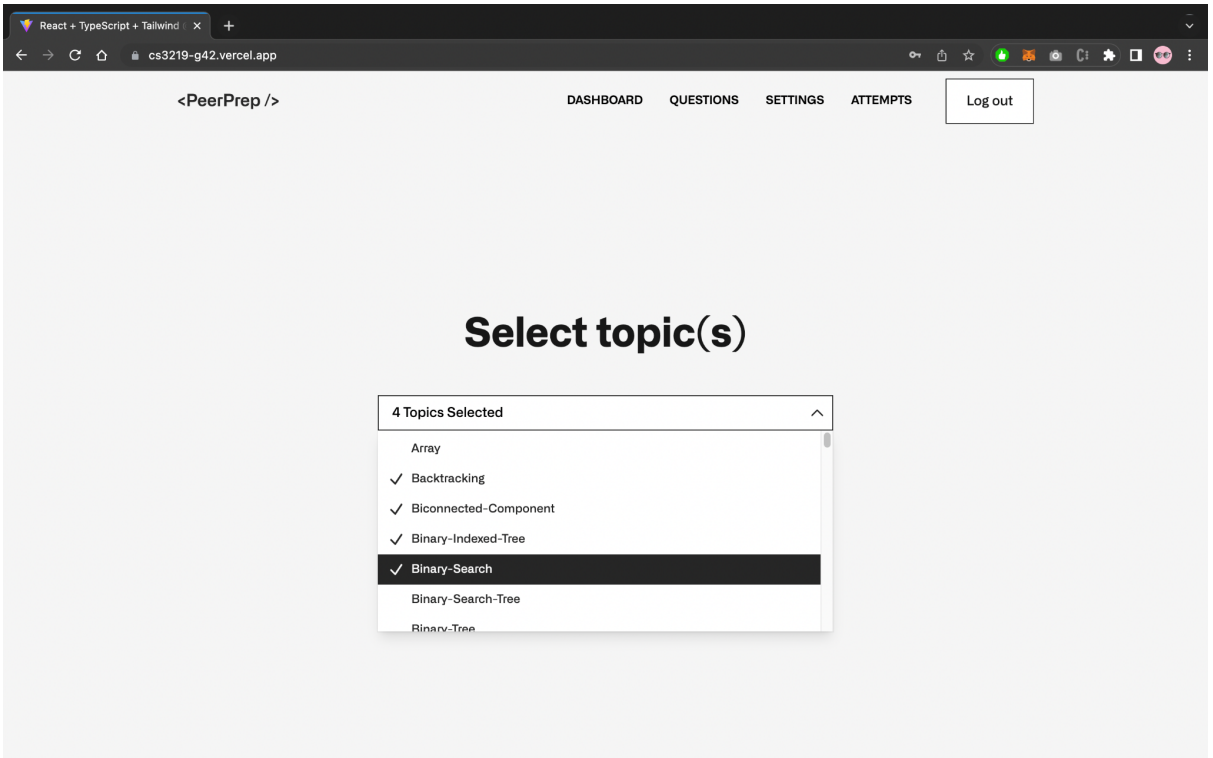
Home page/Dashboard



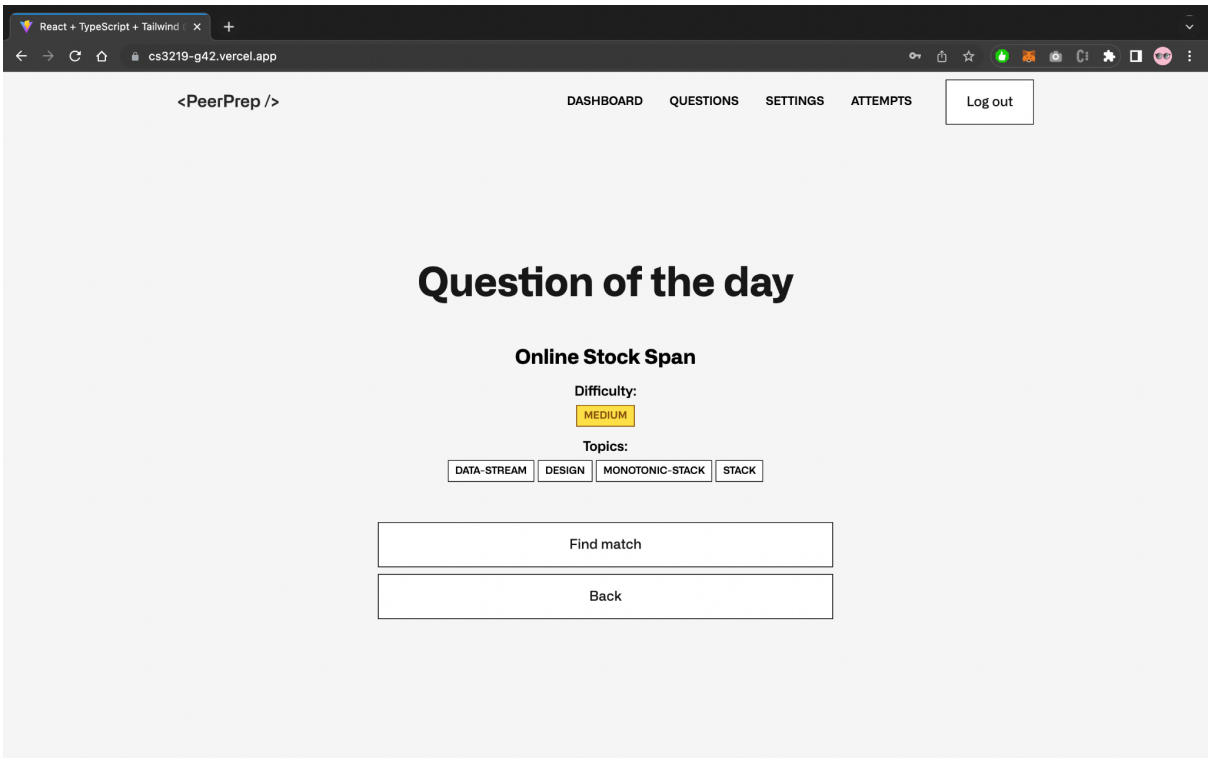
Matching by difficulty(s)



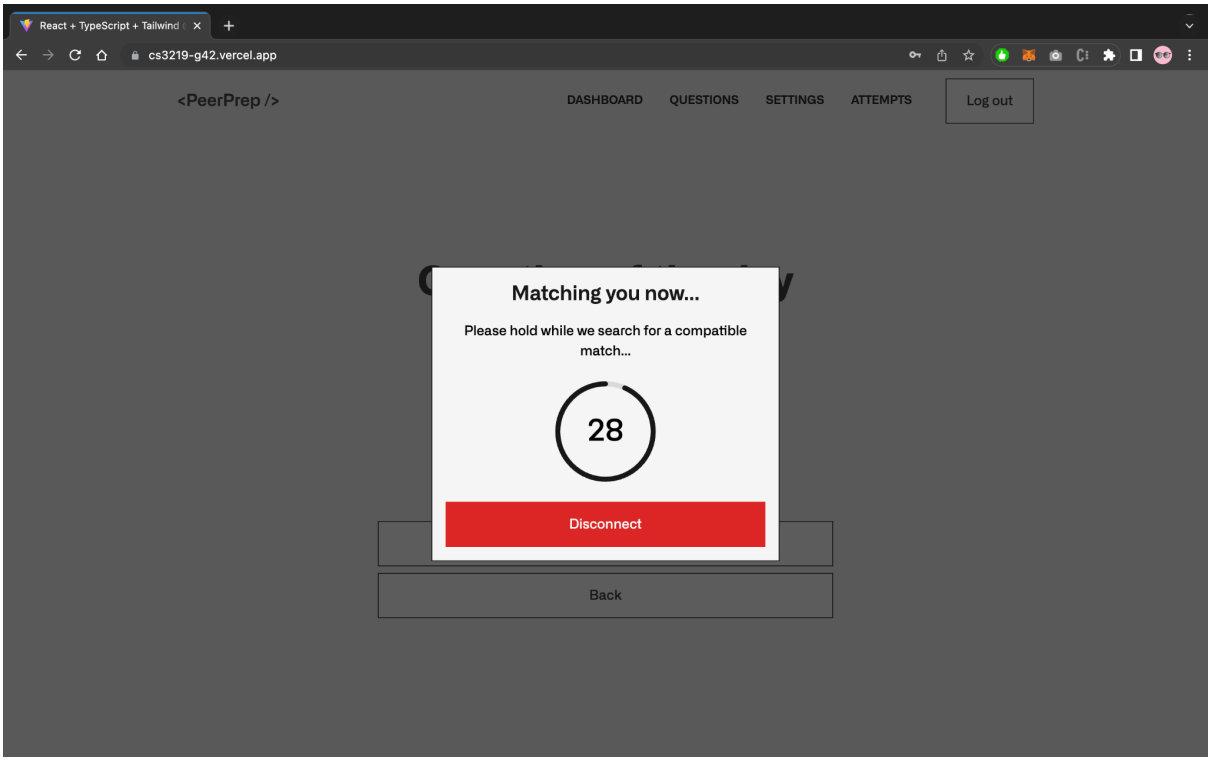
Matching by topic(s)



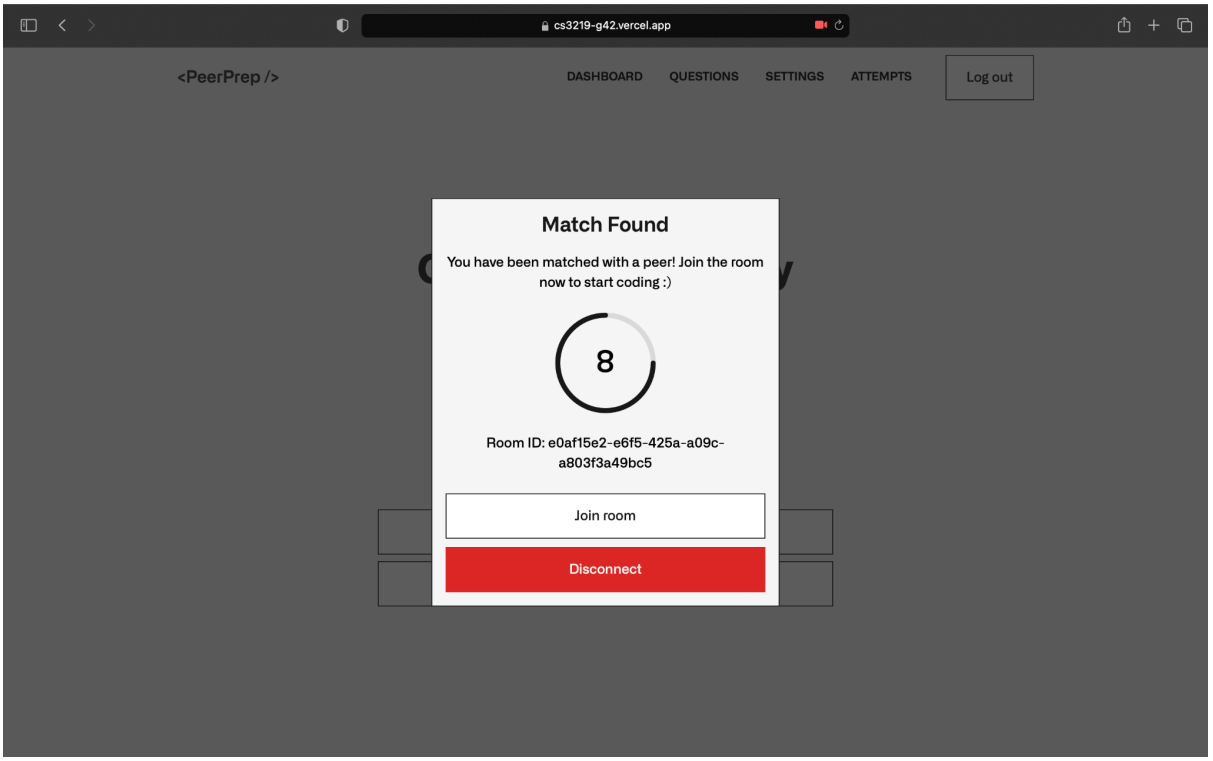
Matching by “Question of the day”



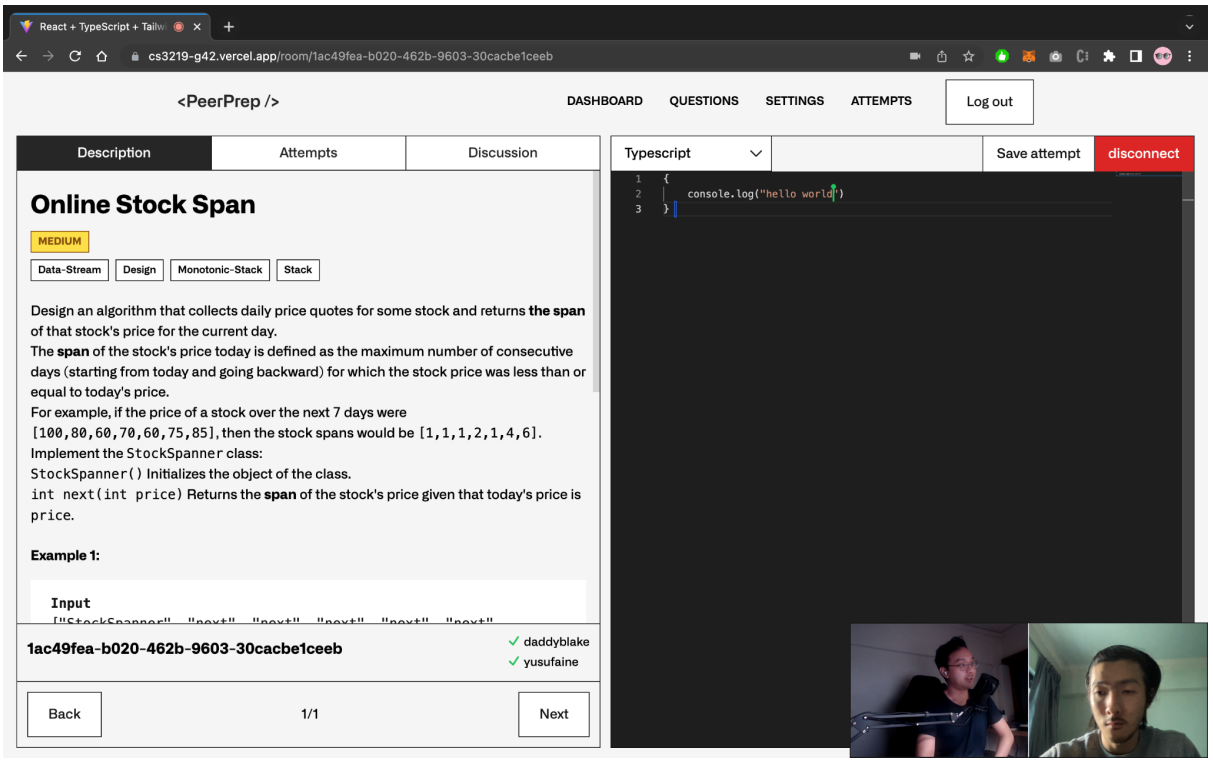
Looking for match



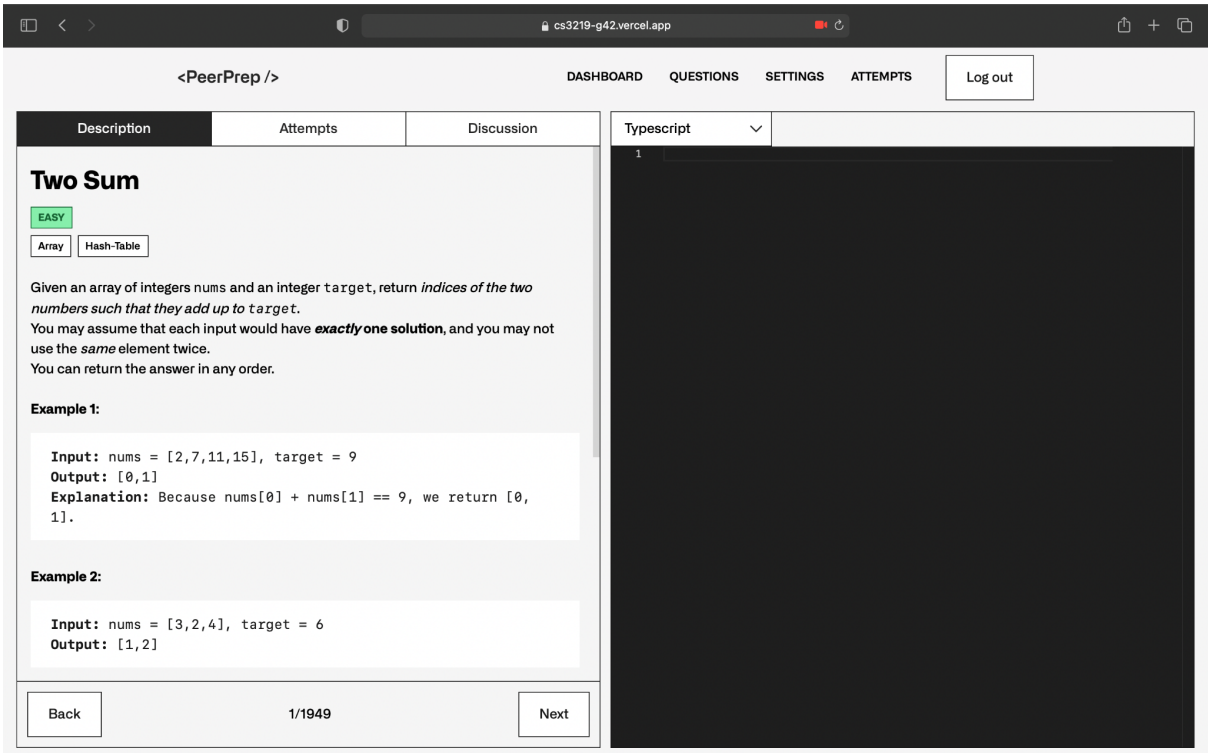
Match found



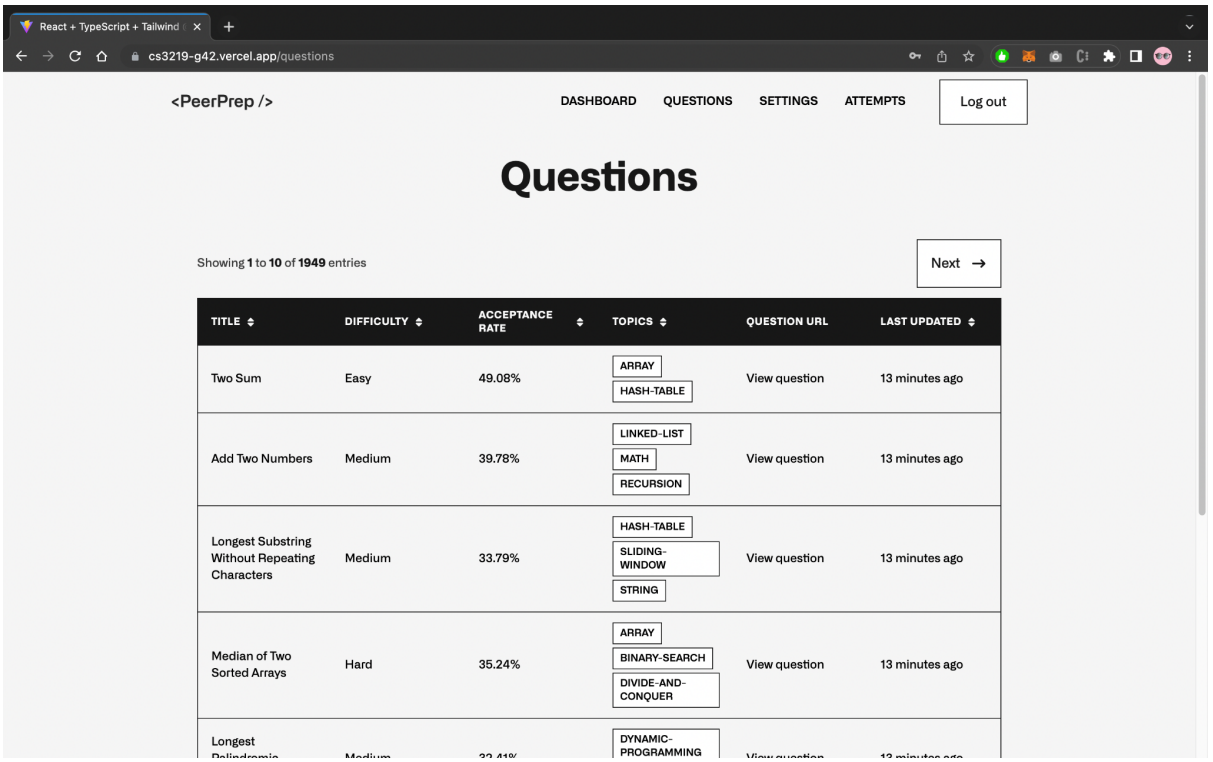
Room page



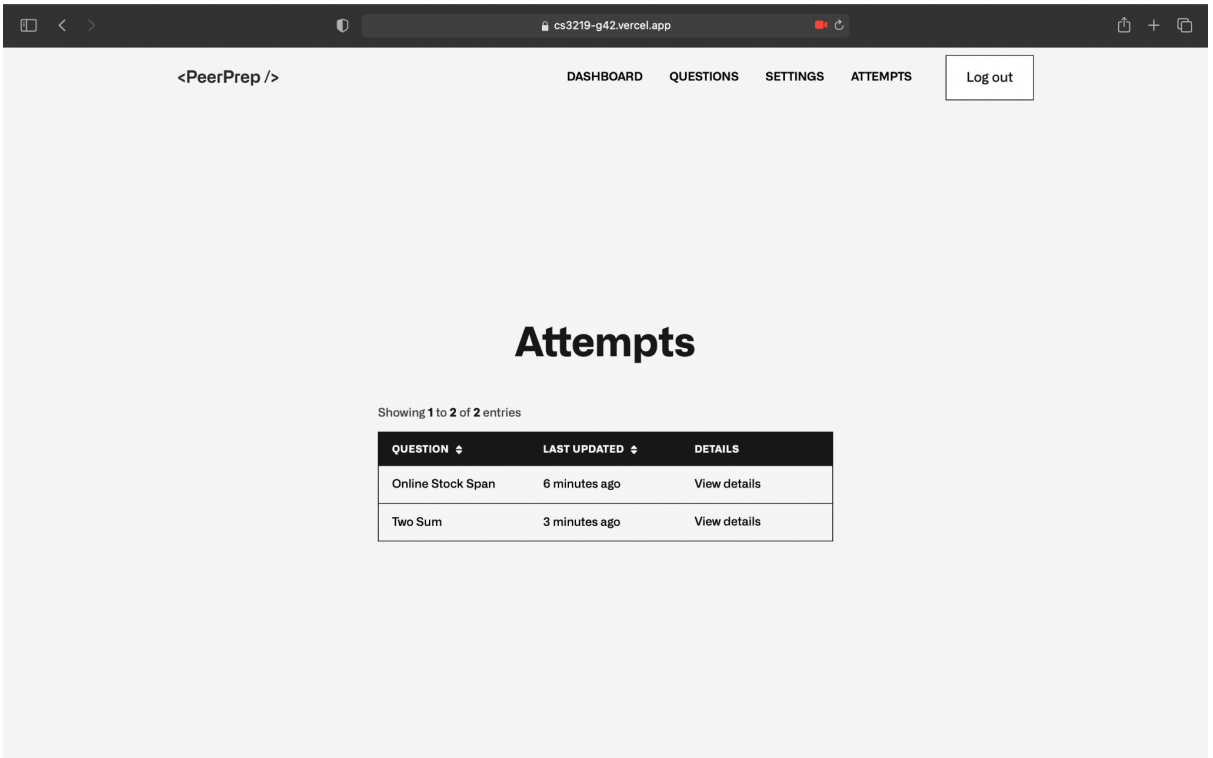
Room page (solo)



Questions page



Attempts page



Settings page