CS3219 Software Engineering Principles and Patterns

Milestone 3 Final Report

Team G19

Name	Matriculation Number
Adam Ho Keng Yuen	A0217872X
Chen Hsiao Ting @Lin Hsiao Ting	A0222182R
Sherman Ng Wei Sheng	A0218131U
Siddharth Srivastava	A0226588N

Table of Contents

GitHub Repository

https://github.com/CS3219-AY2223S1/cs3219-project-ay2223s1-g19

Contributions

Name	Technical Contributions	Non-Technical Contributions
Adam Ho Keng Yuen	Adam handled 3 microservices, namely Matching, Question, and Session service, as well as integrating them with the Frontend.	Adam helped obtain the questions via web scraping and stored them in the question bank. He also wrote on individual portions of the report.
Chen Hsiao Ting @Lin Hsiao Ting	Hsiao Ting handled the overall frontend design, including design of PeerPrep logo and implementation of React JS code for the dashboard, change password, matching popup, and login/sign up page. She also implemented the history service, as well as integrated with the Frontend.	Hsiao Ting helped with the management of DevOps process via the Jira platform. She also set the general structure and wrote on the report.
Sherman Ng Wei Sheng	Sherman handled the User Service as well as the integration of the application with Monaco Editor and Judge0 compiler. He also worked on the live coding and communication logic in the matching microservice. Set up CI/CD pipeline for all microservices on Heroku.	Sherman helped with generating the test cases for the respective questions. He also wrote individual portions of the report.

Siddharth Srivastava	Siddharth handled the frontend design and React JS code for the login/signup, room, and attempt-history page. He also integrated the microservices with the frontend code and was primarily responsible for the smooth user experience in the room page and navigation between the pages.	Siddharth helped create the accounts for MongoDB, Heroku and Vercel. He also worked on individual portions of the report.

1. Introduction

1.1 Background

Technical interviews are necessary for anyone applying for engineering jobs. Therefore, applicants must be strong in problem-solving and articulating technical concepts if they wish to increase their chances of getting hired. Many depend on existing platforms to practise technical interviews, such as LeetCode and AlgoExpert. However, these platforms do not provide sufficient opportunities to practise communication skills essential in technical interviews, such as articulating their thought process with a peer and reasoning algorithmic analysis.

1.2 Purpose

The purpose is to create a web application that helps students better prepare themselves for technical interviews. PeerPrep aims to offer a platform for students to revise core algorithmic concepts with other students uniquely and effectively through collaborative coding. Using a peer learning system, students can learn from each other and break the monotony of revising.

1.3 Project Scope

PeerPrep is an intuitive, user-friendly collaborative application that allows a user to match with someone else to practise technical questions. They have three difficulty options to allow progressive learning. Users are only identified via their usernames throughout the session to promote a safe learning space for everyone.

1.4 Target Audience

Our target audience is anyone seeking peer programming exercises to prepare for technical interviews. We plan to keep the application running for free for the foreseeable future.

2. Overall Description

2.1 Product Perspective

Given the lack of opportunities to practise communication skills in the current technical platforms, PeerPrep aims to offer an avenue for students seeking jobs to revise core algorithmic concepts through collaborative coding.

Through collaboration, users are encouraged to communicate their ideas and algorithms through simple and easily understandable words, an essential soft skill in the technical industry. Furthermore, by matching users with different levels of question difficulties, students are exposed to various problem sets, further strengthening their technical skills.

2.2 Product Features

The list of features implemented in the PeerPrep is as follows.

1. User Interface (Frontend)
2. User Service
3. Matching Service
4. Session Service
5. Question Service
6. History Service

2.3 Operating Environment

The PeerPrep back-end operates in Windows and Mac operating system environments. While the front-end operates within web browser environments such as Google Chrome and Safari.

3. Functional Requirements (FRs)

In the following subsections, the various FRs are organised according to their services. Most FRs were completed, while some were attempted or dropped. The FRs that are attempted are dropped are detailed in Section 3.5 and 3.6, respectively.

3.1 User Service

The User Service is responsible for both the authentication and authorisation of users.

ID	Functional Requirements	Priority
FR1.1	User Service should maintain authentication information about users.	High
FR1.2	User Service should allow users to sign up.	High
FR1.3	User Service should allow users to log in.	High
FR1.4	User Service should allow users to log out of the account.	High
FR1.5	User Service should allow users to delete the account.	High
FR1.6	User Service should allow users to change their password when they are logged in.	High
FR1.7	User Service should provide an interface to authorise users based on the JWT token.	High

3.2 Matching Service

The Matching Service is responsible for pairing up two users, this includes matching, updating shared code and messages, as well as compilation and testing.

ID	Functional Requirements	Priority
Matching Users		
FR2.1	Matching Service should allow users to select the difficulty level of the questions they wish to attempt.	High
FR2.2	Matching Service should be able to match two waiting users with similar difficulty levels and put them in the same room.	High
FR2.3	If there is a valid match, the Matching Service should match the users within 30s.	High
FR2.4	Matching Service should inform the users that no match is available if a match cannot be found within 30 seconds.	High
Collaborative Coding Platform		
FR2.5	Matching Service should allow users to update the code in the code editor.	High
FR2.6	Matching Service should display the updated code for both users.	High
FR2.7	Matching Service should allow users to leave the room.	High
Communication Platform		
FR2.8	Matching Service should allow users to send messages to each other.	Medium
FR2.9	Matching Service should allow users to receive messages from each other.	Medium
FR2.10	Matching Service should allow users to view all the messages for that session.	Medium

Compilation and Testing		
FR2.11	Matching Service should allow users to compile their code.	Medium
FR2.12	Matching Service can compile the source code.	Medium
FR2.13	Matching Service should allow users to test their code with test cases.	Medium
FR2.14	Matching Service can run compiled code against test cases.	Medium
FR2.15	Matching Service should allow users to know the test case results.	Medium

3.3 Question Service

The Question Service is responsible for storing and retrieving question information.

ID	Functional Requirements	Priority
FR3.1	Question Service should store questions (description, constraints, examples, test cases, bonus tasks).	High
FR3.2	Question Service can retrieve a question by difficulty levels.	High
FR3.3	Question Service can retrieve a question by ID.	Medium

3.4 Session Service

The Session Service is responsible for storing, updating, retrieving and removing a collaborative session between two users.

ID	Functional Requirements	Priority
FR4.1	Session Service can store session information (room ID, question ID, code attempt etc.)	High
FR4.2	Session Service can retrieve session information by username.	High
FR4.3	Session Service can update a user’s code attempt in the user’s session information.	High
FR4.4	Session Service can update a user’s status once the user completes the session and leaves the room.	Medium
FR4.5	Session Service can remove a session once both matched users complete the session.	Medium

3.5 History Service

The History Service is responsible for storing and retrieving past completed collaborative attempts.

ID	Functional Requirements	Priority
FR5.1	History Service can store session information once users complete their session.	Medium
FR5.2	History Service should allow users to view a summary of their completed sessions.	High
FR5.3	History Service should allow users to view their attempted solutions to attempted questions.	Medium
FR5.4	History Service can retrieve a user’s attempted solutions.	Medium
FR5.5	History Service should allow users to view the number of questions they have attempted by difficulty level.	Low
FR5.6	History Service allows users to sort their history by certain fields (DateTime, question difficulty etc.).	Low

3.6 Dropped FRs

Additionally, our team has also agreed to drop entirely certain FRs that we deem do not add value to the PeerPrep application given the constraints and limitations. The table below states the FRs and their respective ID labelled in Milestone 1 report, as well as justifications on why we removed them.

ID*	Functional Requirements	Priority	Justification
FR1.8	User Service should allow users to reset their password when they forget while logged out.	Low	Good to have but the team decided to channel more attention towards other more high priority FRs.
FR3.4	Communication Service should allow the conversation to persist for the duration of the session.	Medium	Good to have but not necessary for the application to be fully functional. We drop the requirements as we think the current messaging tool suffices in terms of supporting users communication.
FR3.5	Communication Service should allow users to video call the other participant in the room.	Low	
FR4.3	Question Service should retrieve the sample answer to a given question.	Medium	Good to have but not necessary. Decided to not include the answer to the question, but does not affect
FR7.2	History Service should allow users to get the answer to the attempted questions.	Medium	

			the main goal of the application.
FR4.4	Question Service should retrieve a question from a set of unattempted questions to the user.	Medium	Good to have features to retrieve unattempted questions. However, we dropped the idea due to the complexity and limited time available.
FR7.3	History Service should inform users if they have attempted all questions of a selected difficulty level.	Low	

* The IDs are based on Milestone 1 requirements

4. Non-Functional Requirements (NFRs)

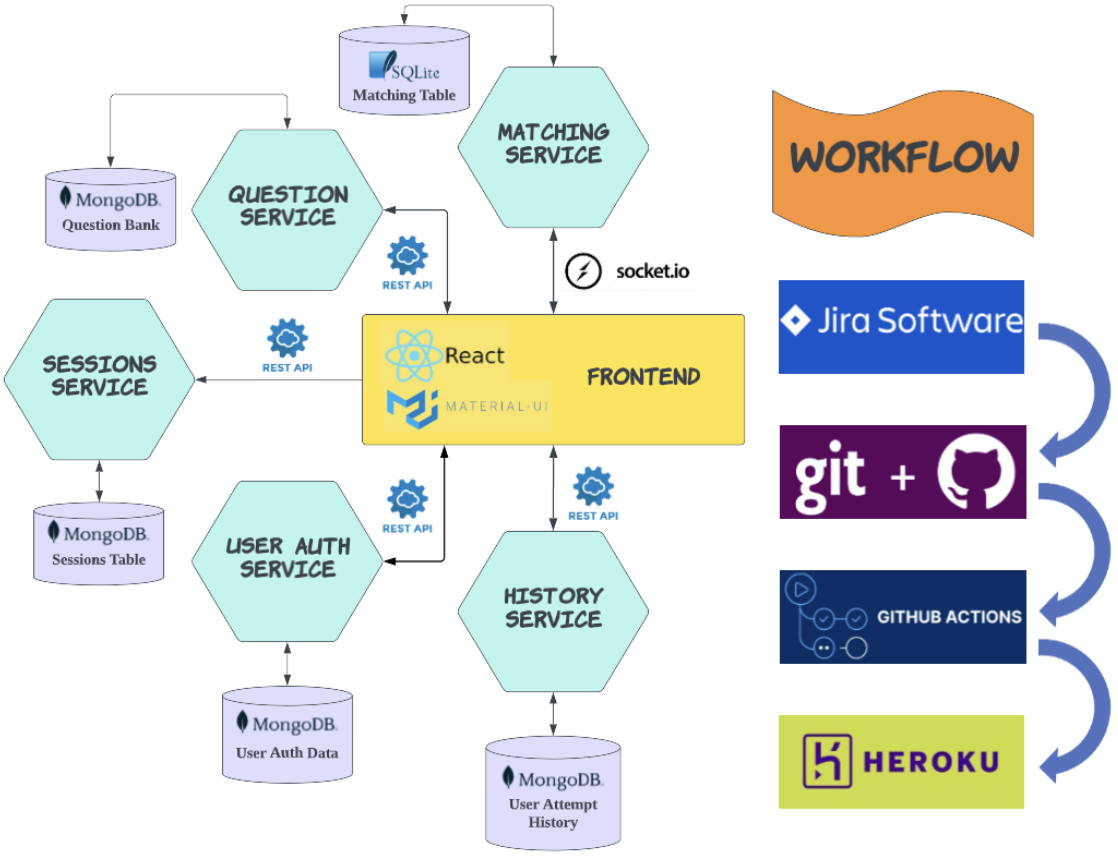
ID	Non-Functional Requirements	Priority	Quality Attribute
1	A user's password should be hashed and salted before storing them in the DB.	High	Security
2	All API calls should be authorised with a valid JWT.	High	Security
3	The dynamic live update of users' code in the code editor should not exceed 2 seconds.	High	Performance
4	The dynamic live update of messages in the chat system should not exceed 2 seconds.	High	Performance
5	User History should be fetched within 3 seconds.	High	Performance
6	User History should update the number of questions attempted by a user within 3 seconds.	High	Performance
7	The questions should be displayed in the panel within 2 seconds.	High	Performance
8	Session information has to be maintained throughout a session between matched clients.	Medium	Robustness
9	An intuitive, presentable, and user-friendly UI for users to access the application's functionalities.	Medium	Usability

Attempted NFRs

ID	Non-Functional Requirements	Priority	Quality Attribute	Justification
1	JWT Session Token blacklist size should not be ever-increasing.	Medium	Security	Token storage is deemed to be manageable in size at this point.

5. Architecture Decisions

5.1 Architecture Diagram



Our PeerPrep project architecture consists of a Frontend and 5 microservices namely Matching Service, Question Service, Sessions Service, User Auth Service and History Service. Each service has its independent database and endpoint and the frontend acts like an orchestrator interacting with the other microservices either to REST APIs or through Socket.IO.

Our development workflow consisted of the creation of tasks on JIRA. Then we as a team would design and code either independently or in pairs. Our codebase was constantly changing and we used GitHub for both version control as well as hosting the code. We also made use of GitHub actions for CI/CD and deployed our microservices and frontend code to their respective endpoints.

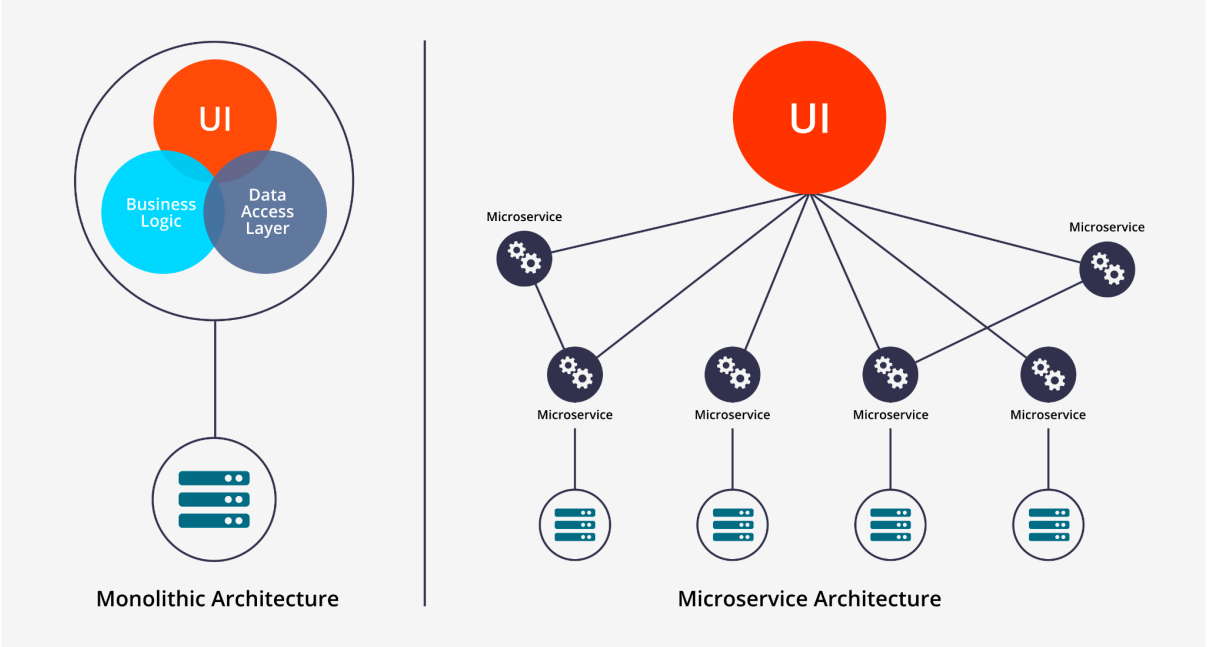
5.2 Architecture Considerations

5.2.1 Consideration 1 - Microservice over Monolithic

Description

Monolith architecture is built as a single codebase containing all the required functionalities. While **microservices architecture** is built as a collection of services that allows rapid, frequent and reliable delivery of large, complex applications.

The diagram below shows the diagrams of monolith architecture and microservices architecture.



Ref: https://medium.com/hengky-sanjaya-blog/monolith-vs-microservices-b3953650dfd

To understand which architecture is more suitable for our application, we consider the differences between the two.

Categories	Monoliths	Microservices
Architecture	Single build of a unified code.	Collection of small services.
Scalability	Difficult to scale as the application grows.	Independent scaling and better usage of resources.

Development	Slow build and release cycle since the code base is enormous, which slows down the development and testing cycle.	More agile since each microservice can build a module and deploy it independently.
Deployment	Easy deployment since it only requires a single deployable code instead of making updates in separate entities.	Relatively more complex deployment as multiple deployable codes are needed for different microservices.
Reliability	If one service fails, the entire application goes down.	More reliable as the application will not go down as a whole if one service fails.
Performance	Better performance due to centralised code and memory, which reduces API calls.	Lower performance due to the need to make multiple API calls to multiple microservices to load the UI..

Final Architecture Decision

Given that PeerPrep is a team-based project and a highly agile application that demands swift speed of delivery, as well as a high level of reliability and scalability, our team has decided to leverage the microservices architecture over monolithic architecture.

Justifications

The microservices architecture enables our team to develop, test and deploy different services independently at the same time. By working on a separate branch for every functionality, rather than building on the same code base, it removes the need for members to wait for one another for updates due to the reduced coupling among the modules. This increases productivity and reduces complexity when working as a team.

With evolving requirements, the microservices architecture allows us to make continuous changes to functionalities, and deploy each functionality in a way that will not affect the rest of the code. This makes development easy since the code base of services is relatively small and independent of one another, hence are easily understandable and modifiable.

Moreover, the microservices architecture ensures the reliability of PeerPrep. Given the loose coupling between microservices, If one of the services fails, the rest of the services are not affected in unforeseen circumstances.

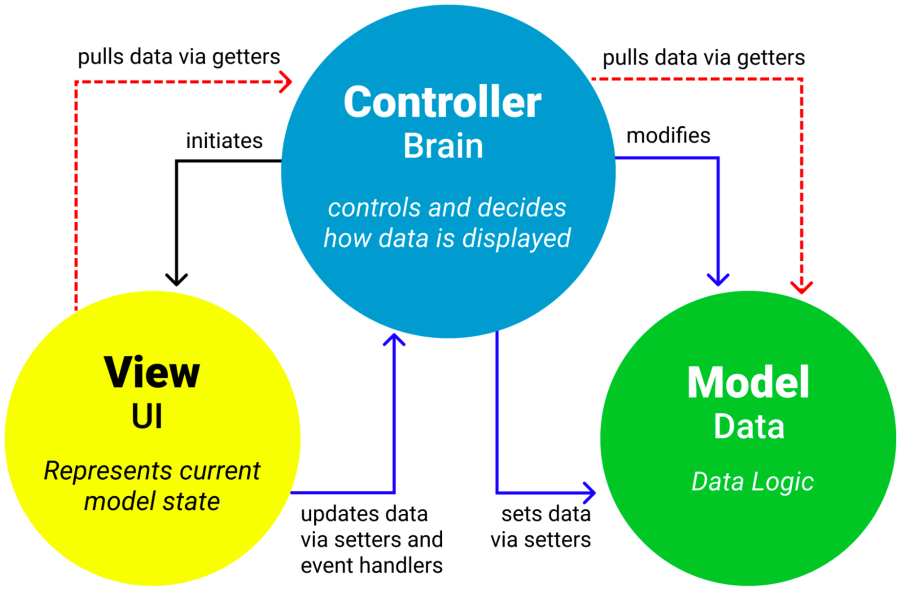
Additionally, since the code base is divided into smaller and simpler individual services, it also improves testing, maintainability and scaling of the project.

5.2.2 Consideration 2 - Model View Controller (MVC)

Description

The model-view-controller (MVC) architectural pattern contains three main components in software development:

1. Model - backend that contains all the data logic
2. View - frontend or graphical user interface (GUI)
3. Controller - brains of the application that connects the model and the view, and controls how data is displayed



Ref:
https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/

The table below lists some pros and cons of the MVC pattern.

Pros	Cons
Development of the application becomes fast due to a less complicated code base.	Increased complexity.
Easy for multiple developers to collaborate and work together due to the separation of concerns.	Hard to understand MVC architecture due to its complex structure and frequent updates.
Easier to debug due to the multiple levels properly written in the application.	Cost on performance due to frequent updates as views could be overloaded with update requests if the model is changed frequently.
Able to provide multiple views.	Strict rules over the methods.
Support for asynchronous technique, which helps developers to build quick loading web applications.	Difficulty of using MVC with modern user interface tools.
Modification does not affect the entire model as all three components are able to work independently.	-
Returns data without formatting, enabling developers to call components and put them to use with many interfaces.	-

Final Architecture Decision

Given the nature of PeerPrep, which is a large-size web application that supports easy and faster development, asynchronous method invocation and multiple views, our team has decided to adopt the MVC architectural pattern after weighing its advantages and disadvantages.

We have implemented the controller and model components in each microservices, while the views are implemented in the front-end. We will discuss the rationales and justifications in the following section.

Justifications

MVC supports rapid and parallel development. For instance, if a member is working on the view, other members can also work on the controller and model at the same time since the three components are independent of one another. This significantly helps to accelerate up to three times the average speed of the development process of an application.

Moreover, the MVC architectural pattern works well with JavaScript and its framework, hence supporting asynchronous techniques which enables us to build faster-loading web applications.

Additionally, with the MVC architectural pattern, our team is able to create multiple views for the application, without much code duplication because it separates data and business logic from the display.

5.2.3 Consideration 3 - Event-based, Implicit Invocation

Description

The implicit innovation architecture for a system is structured around event handling. Instead of explicitly invoking procedures, a component can announce/broadcast one or more events to other components who register themselves to an event of interest. When the event is announced, the system invokes all of the procedures registered for the event. Furthermore, events and listeners can themselves trigger other events.

Final Architecture Decision

PeerPrep heavily involves the nature of event-handling at runtime of the application. We require the ability for users to dynamically match with other potential users and attempt the same question within the same room. We achieved this with Socket.IO implementation for the Matching Service to trigger the necessary events and the set of procedural calls required to match two users.

Justifications

To match two users, the clients and the Matching server communicate through a series of events invoked, first, by a “match” event. In our implementation, the following events include handling (un)successful matches, signalling matches found and joining a room.

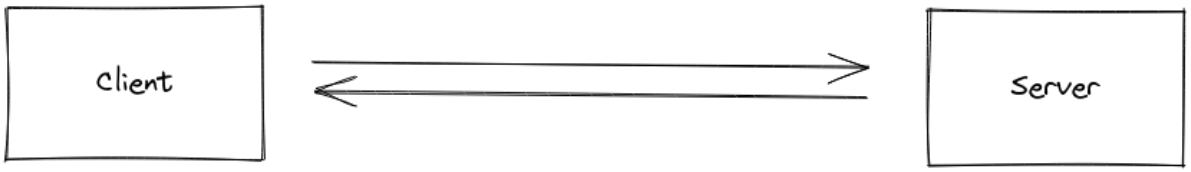
Since the clients and the Matching server communicate with each other dynamically at runtime, the components are loosely coupled. We can bind procedures with events at runtime without compile-time determination, which increases the flexibility and maintainability of the software. Clients can easily introduce themselves into the system by registering for the events regarding matching, aiding greatly in reusability. There is also no need for them to modify their own/other component’s interface during registration, which provides scalability.

6. Design Patterns

6.1 Pub-Sub Messaging Pattern

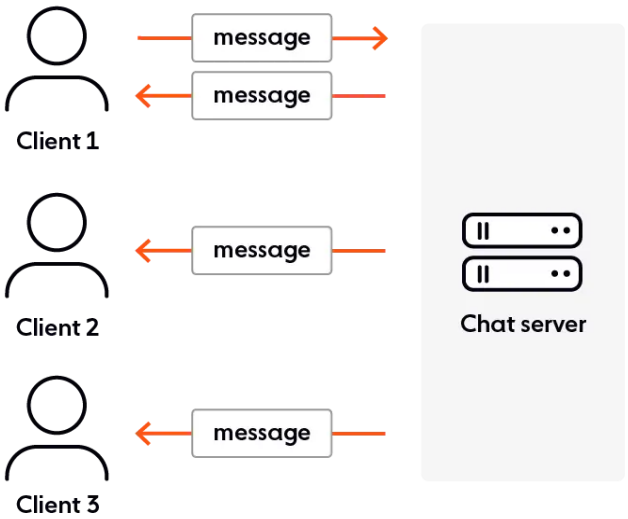
6.1.1 Description

Socket.IO is a library that enables low-latency, bidirectional and event-based communication between a client and a server. It is built on top of the WebSocket protocol and provides additional capabilities such as automatic reconnections, broadcast support, or falling back to HTTP long polling.



Ref: https://socket.io/docs/v4/

Socket.IO provides a basic chat application. When the server receives a new message, it will send the message to all the clients and notify them, bypassing the need to send requests between client and server.



Ref: https://ably.com/topic/socketio

The table below shows some of the pros and cons of Socket.IO.

Pros	Cons
Real-time updates. E.g. real-time analytics models can send data to clients for a real-time graphical UI.	Limited native security features. E.g. end-to-end encryption, or a mechanism to generate and renew authentication tokens.
Bidirectional communication, where both the client and the server send and receive messages.	Only supported in a single region, can lead to issues such as increased latency and even system downtime if users are in different regions.
Ease of access and setup.	Unnecessary binary overhead.
No internet dependency.	Callback-centric. A client / server can overwhelm a server / client by sending / producing events rapidly, causing the server to run out of memory or CPU.
Event-based WebSocket wrapper between Client and Server.	No mechanism for message guarantees
Able to handle connections routed via proxy serves and load balancers.	No specification for the Socket.io wire protocol, which is important for interoperability.
Supports broadcasting, in which a server sends the same message to multiple clients.	-
Fallback Options.	-

6.1.2 Final Design Pattern Decision

Our team decided to use Socket.IO to support the chat feature in the PeerPrep collaboration room. We will discuss some of the reasons in the following section.

6.1.3 Justifications

Socket.IO is a highly performant and reliable library optimised to process a large volume of data with minimal delay. This is essential for our chat service since we would want the users to be able to send and receive messages in real-time.

Moreover, it follows the WebSocket protocol and provides better functionalities, such as fallback to HTTP long-polling or automatic reconnection. Unless instructed, otherwise a disconnected client will try to reconnect forever, until the server is available again. This ensures that the client can still re-establish connection with the server after experiencing unforeseen circumstances such as a network breakdown.

While there is no mechanism for message guarantee, one workaround is to use the acknowledgements feature, which acknowledges that a message is successfully received.

Despite the limitations of Socket.IO, we were appealed by the ease of access and setup. The figure below shows a sample code of Socket.IO.

```
io.on('connection', socket => {
  socket.emit('request', /* … */); // emit an event to the socket
  io.emit('broadcast', /* … */); // emit an event to all connected sockets
  socket.on('reply', () => { /* … */ }); // listen to the event
});
```

6.2 Provider Pattern

6.2.1 Description

Provider pattern is not a design pattern covered by the Gang of Four. However, during our implementation process, we realised the need to pass around a set of data to React Child components, information such as the current user and even objects such as sockets. With the use of the Provider Pattern, we are able to make data available to many components. Instead of passing the data down each layer through props, which is the traditional method of passing information around React components.

In order to use the Provider Pattern, we make use of React Context to share certain states among all the child components. However, React Context also comes with its own limitations.

The table below shows some of the pros and cons of using React Context in a Provider Pattern.

Pros	Cons
Simple to implement	Component re-renders whenever the context changes
Reduce coupling among child components for data flow	-
Allows bi-directional data flow	-

6.2.2 Final Design Pattern Decision

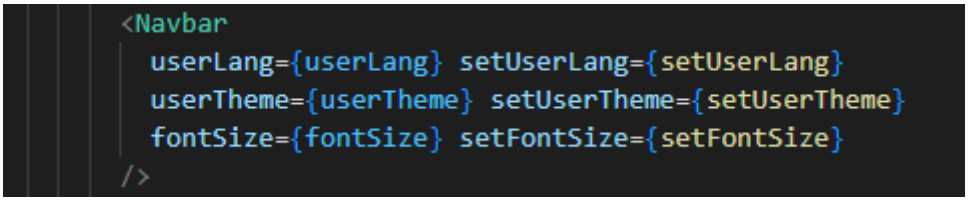
Between the two options, using a Provider Pattern via React Context or passing data as props from parent to child components, our team decided to use both.

6.2.3 Justifications

We tried to limit the use of Provider Pattern in instances where the values are unlikely to refresh often, which will not cost countless re-rendering of the components. A key component that we have adopted a Provider Pattern is the AuthContext.

We discussed and concluded that the user state is likely to remain unchanged as the most common change case would be when the user logs out and logs back in. However, we recognise that the user identity (his/her username) is a data that is frequently requested by many of the components.

In other cases where there is no need for extensive sharing of data with other components, the parent component continues to pass data via props to the child component. This is evident in our Navbar for the Codeblock where the React State for the “userLang”, “userTheme” and “fontSize” continues to be passed as props.

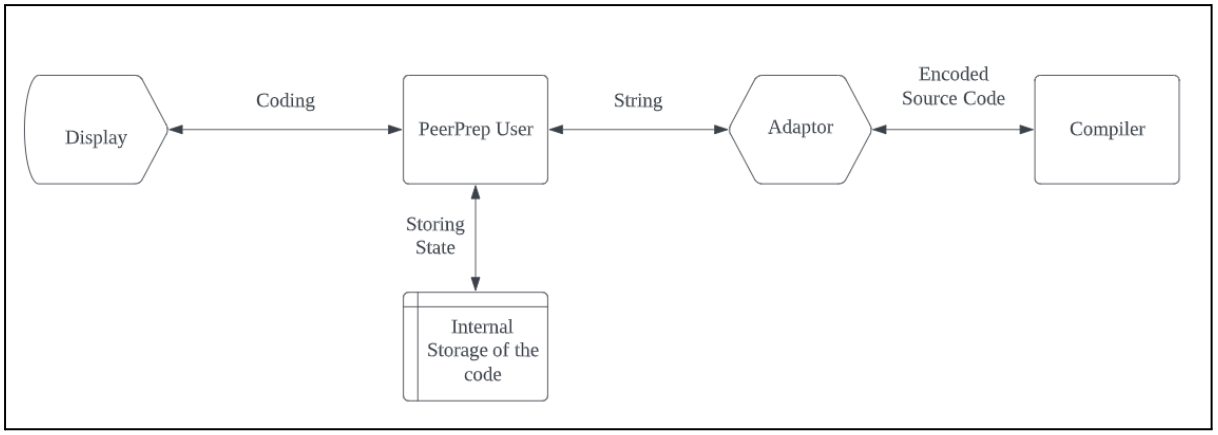


Since this is a pattern not covered in the scope of module, the source referenced for this Provider Pattern are as follows:
https://www.patterns.dev/posts/provider-pattern/
https://blog.logrocket.com/pitfalls-of-overusing-react-context/

6.3 Adapter Pattern

6.3.1 Description

Adapter pattern is a structural design pattern that enables collaboration between two or more incompatible interfaces. In our project we tried to implement compilation of the users code and running test cases on the code. The compilers accepted code but were facing problems due to the incompatible interface of the compilers and the users code. Since the interfaces were incompatible, we created a simple adapter that allows smooth collaboration between the user and the compiler by converting the string to the encoded source code.



Below are mentioned some of the pros and cons of using the adapter pattern

Pros	Cons
Simple to implement	Creation of new adapters may require additional code
Single Responsibility Principle is applicable since this separates data conversion code from the business logic	The addition of adapters may increase the complexity and readability of the code
Open/Closed Principle is also applicable since this allows extensibility to create new adapters for other services without significant changes to the codebase	-

6.3.2 Final Design Pattern Decision

Since we decided to use a third party compilation service to compile the users' code and run test cases we decided to use an adapter to make the two interfaces compatible.

6.3.3 Justifications

The adapter allowed us to simply convert the user's code in string format to the third-party compiler's acceptable code in a base64 encoded JSON object.

Our team decided to use this adaptor to separate the data conversion code and the business logic. The creation of the adapter allows us to easily make the two interfaces compatible while also enabling extensibility. Since this adapter is easily extensible it allows the scope for the addition of other third party integration like minification of code or run time complexity calculation.

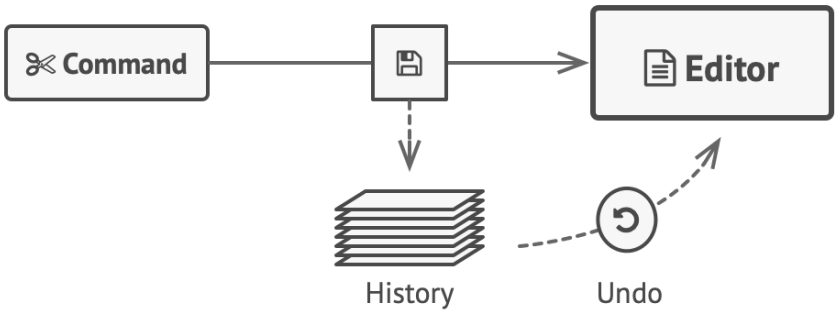
6.4 Memento Pattern

6.4.1 Description

Memento is a behavioural design pattern that enables the saving and restoring of the previous state of an object without revealing the details of its implementation.

Memento pattern uses three actor classes, namely

1. Memento - contains the state of an object to be restored
2. Originator - creates and stores states in Memento objects
3. Caretaker - restores object state from Memento



Before executing an operation, the app saves a snapshot of the objects' state, which can later be used to restore objects to their previous state. Ref: https://refactoring.guru/design-patterns/memento

The table below shows some of the pros and cons of the Memento pattern.

Pros	Cons
Produce snapshots of the object's state without violating encapsulation.	The app might consume lots of RAM if clients create mementos too often.
Simplify the originator's code by letting the caretaker maintain the history of the originator's state.	Caretakers should track the originator's lifecycle to be able to destroy obsolete mementos.
-	Most dynamic programming languages cannot guarantee that the state within the memento stays untouched.

6.4.2 Final Design Pattern Decision

In order to allow users to access the previously attempted questions and solutions, our team has decided to utilise the Memento pattern to create a state snapshot of each finished session.

The room service serves as the Originator class, which produces snapshots of the previous collaborative session state upon submission of code attempts. The history object is the Memento, which can act as a snapshot of the originator's state. While the front-end serves as the Caretaker, which controls when and why to capture the originator's state as user history, and when the state should be restored.

The justifications for the Memento pattern are discussed in the following section.

6.4.3 Justifications

We wanted to produce a snapshot of users' collaborative session and be able to restore to a previous state of that object. The Memento pattern allows us to create a copy of the state of the collaborative session, including information such as question id, attempted solution, and username of the matching partner. This can later be used by users to retrieve all previously attempted questions and solutions, without having to alter the original collaborative's state contained in the snapshot.

7. Technology Stack

	Technology/Application
Frontend	React, Material-UI, Monaco Editor
Backend	Express.js, REST API
Database	MongoDB, SQLite
Deployment	Heroku, Vercel
Cloud Providers	MongoDB Atlas
Pub-Sub Messaging	Socket.IO
CI/CD	Github Actions
Project Management Tool	JIRA

8. DevOps

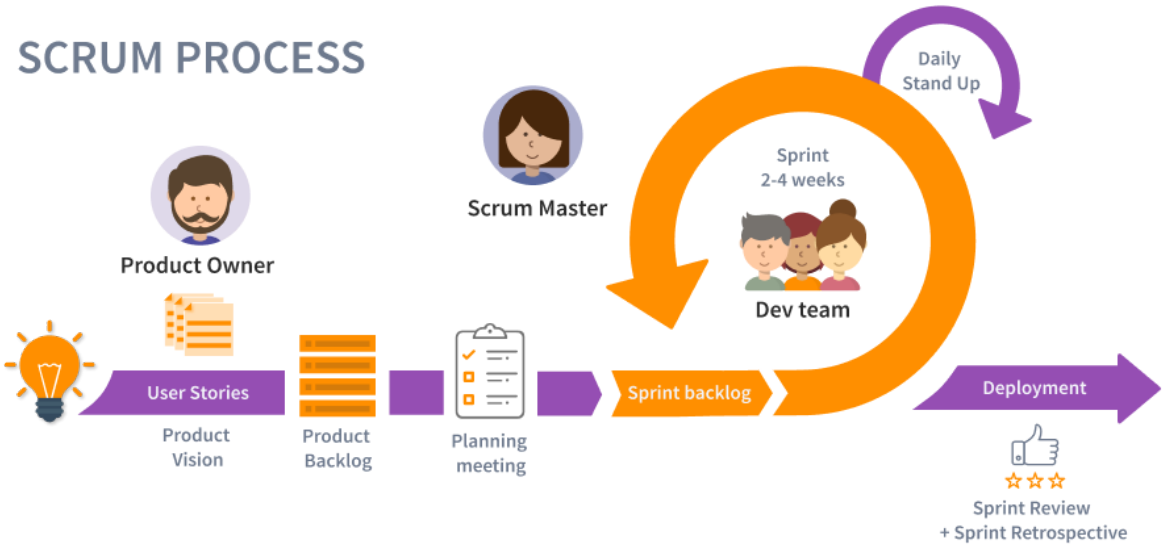
8.1 Scrum Development Process

As an agile development methodology, scrum is used in the development of software based on iterative and incremental processes. It is an adaptable, fast, flexible and effective agile framework that is designed to deliver value to the customer throughout the development of the project.

The scrum development process enables our team to satisfy evolving needs collectively and continuously. It also helps us visualise a general idea of the project requirements, and manage a list of product backlogs ordered by priority so that we are able to deliver the necessary features at varying milestones.

Our team uses the scrum methodology for the following reasons.

1. **Easily scalable** - scrum processes are iterative and are handled within a specific period of time. This makes it easier for our team to focus on a certain number of functionalities for each period.
2. **Compliance of expectations** - enables all team members to be aware of the client expectations and their respective priority. The Product Owner will then verifies if the requirements have been met on a regular basis.
3. **Flexible to changes** - allows easy adaptation to changing requirements throughout the development process.
4. **Time to market reduction** - allows users to start using the most important functionalities of the PeerPrep application before it is completely ready.
5. **Higher software quality** - ensures higher quality software through continuous feedback, improvement and testing after each iteration.
6. **Timely prediction** - allows our team to set sprint and goals for different specific period of time, hence able to estimate which features will be available at a certain time.
7. **Reduction of risks** - enables team members to start working on the functionalities with the highest priority. Coupled with the ability to estimate the project progress, scrum process thus allows us to clear risks effectively in advance.



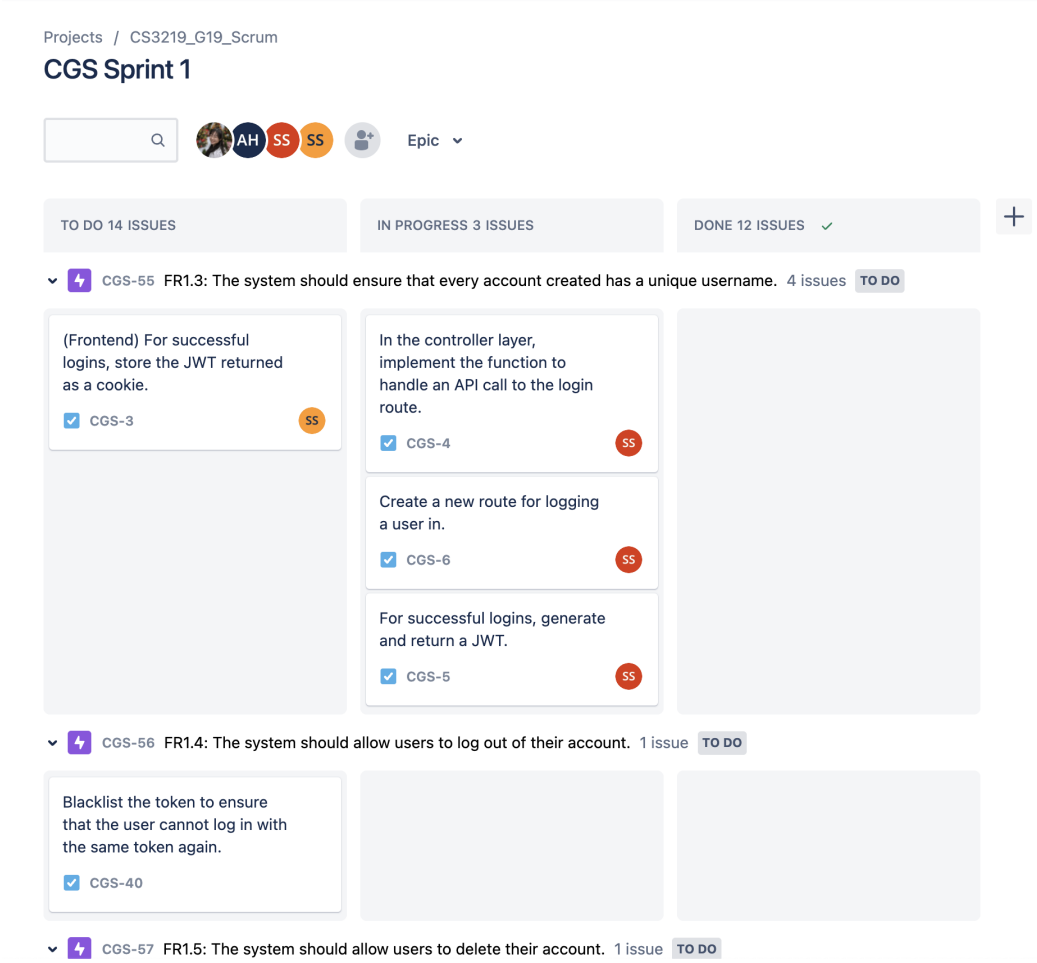
Ref: https://www.tuleap.org/agile/agile-scrum-in-10-minutes

8.1.1 Sprint

A sprint is a short iteration ranging from 2 weeks to 1 month, during which the development team will design, realise and test new features.

As part of our team's sprint planning, we meet up every week, either physically or virtually, to list out all the product backlog and discuss their respective priorities. We then set the tasks with higher priority as our deliverables in the earlier sprint.

The figure below shows our sprint 1 planning on Jira, a widely used platform for planning, tracking and managing agile and software development projects. In sprint 1, we planned to achieve the most basic functionalities of PeerPrep such as allowing unique accounts to be created and logged out.



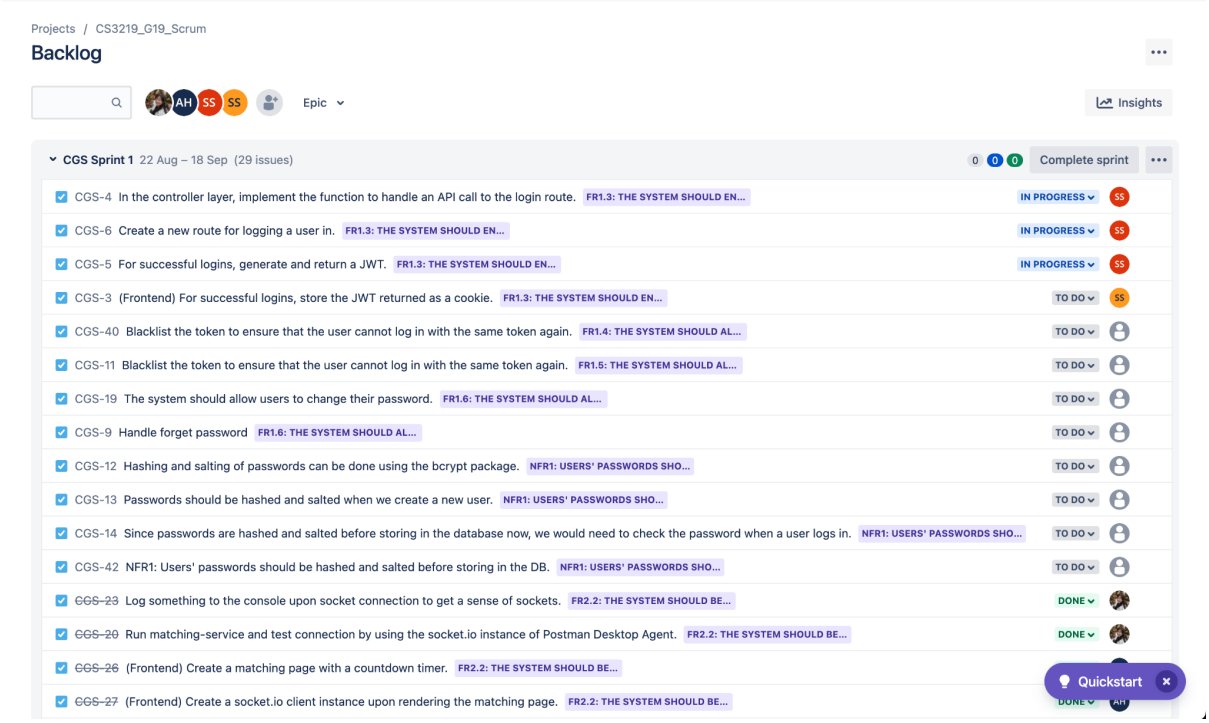
Sprint 1 showing our team's plan to achieve the most basic functionalities of PeerPrep

The tasks are categorised into 3 main categories, namely "To Do", "In Progress" and "Done". This helps our team to easily track and visualise the project's progress at each sprint stage.

8.1.2 Product Backlog

The product backlog is an evolving, ordered list of what is needed to improve the product. After reaching agreements on the goals and features to be implemented, our team has broke down each feature into smaller tasks that can be easily handled by a single member. We subsequently added all these tasks into the product backlog.

The figure below shows a snapshot of our product backlog. The tasks are neatly categorised into their respective epic, which represents the macro functionality of PeerPrep. This enables us to track the amount of tasks to be done at one glance, and evenly distribute them among team members and between different stages of development.

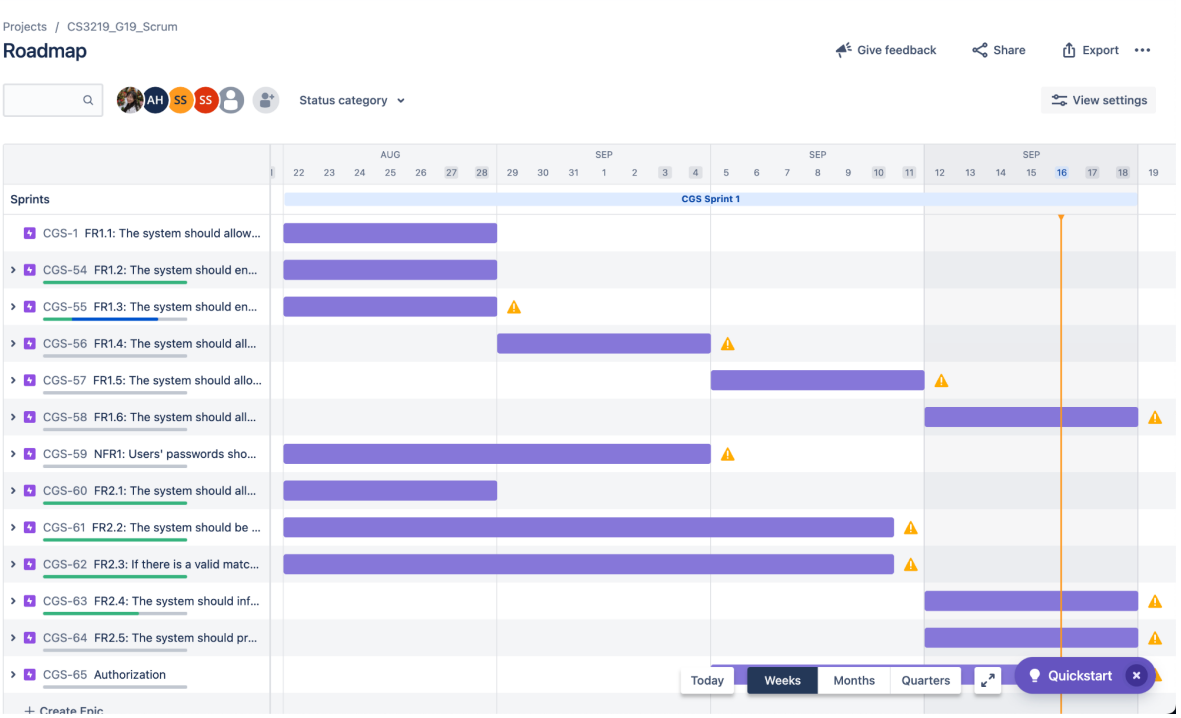


A snapshot of our product backlog showing tasks neatly categorised into their respective epic

8.1.3 Roadmap

Our scrum product roadmap allows us to visualise upcoming sprints in our product development workflow. When juggling multiple epics or features, it can be difficult to keep track of which features will be tackled during each sprint cycle. A scrum roadmap therefore, is useful in getting all team members on the same page, and makes it easy to communicate which features are currently in process of development.

The diagram below shows our team's roadmap on Jira. By setting a start and end date for each epic, we are able to visualise at one glance the progress of the project. This helps to make sure that our team works on sufficient features at each milestone, and ensure that we deliver on time.



Our team's roadmap on Jira

8.2 Deployment

8.2.1 Local Deployment

We have a single GitHub repository with all the frontend and backend code. Please follow the below mentioned steps to run the application locally.

Steps

- Clone our GitHub Repository from the following link (link)
- Since our repository has multiple services we have a folder each for every service which is run independently of each other when we run locally
    1. Frontend:
        a. From the Root Directory “cd frontend”
        b. Install the dependencies “npm install”
        c. Create an Environment File “touch .env”
        d. Add environment variables
            i. “REACT_APP_ENV=DEV”
            ii. “REACT_APP_RAPID_API_KEY=8a69aec1c3mshe36c3a3f9cd7233p159784jsn36bbc669097a”
        e. Run the Frontend Project “npm start”

    2. User Service:
        a. From the Root Directory “cd user-service”
        b. Install the dependencies “npm install”
        c. Create a MongoDB URL
        d. Create an Environment File “touch .env”
        e. Add three environment variables
            i. “ENV=DEV”
            ii. “DB_LOCAL_URI=<A MongoDB URL>”
            iii. “SECRET_KEY= <A Secret Hash>”
                1. You can generate a hash from this link
        f. Run the User Service: “npm run dev”

    3. Matching Service:
        a. From the Root Directory “cd matching-service”
        b. Install the dependencies “npm install”
        c. Run the User Service: “npm run dev”

4. Question Service:
    a. From the Root Directory “cd question-service”
    b. Install the dependencies “npm install”
    c. Create a new MongoDB URL or reuse the one created above / Get the MongoDB URL with the pre-populated questions from the development team
    d. Create an Environment File “touch .env”
    e. Add three environment variables
        i. “ENV=DEV”
        ii. “DB_LOCAL_URI=<A MongoDB URL>”
    f. Run the User Service: “npm run dev”
5. History Service
    a. From the Root Directory “cd history-service”
    b. Install the dependencies “npm install”
    c. Create a new MongoDB URL or reuse the one created above
    d. Create an Environment File “touch .env”
    e. Add three environment variables
        i. “ENV=DEV”
        ii. “DB_LOCAL_URI=<A MongoDB URL>”
    f. Run the User Service: “npm run dev”
6. Session Service
    a. From the Root Directory “cd session-service”
    b. Install the dependencies “npm install”
    c. Create a new MongoDB URL or reuse the one created above
    d. Create an Environment File “touch .env”
    e. Add three environment variables
        i. “ENV=DEV”
        ii. “DB_LOCAL_URI=<A MongoDB URL>”
    f. Run the User Service: “npm run dev”

❖ Now you can open http://localhost:3000/ to view the application

8.3 CI/CD using Github Actions and Heroku

We configured GitHub actions to deploy our microservices to their respective endpoints using Heroku. Our GitHub action allows us to smoothly do continuous deployment since it deploys all our microservices and the frontend on any changes to the main branch. You can view our GitHub Action file on the right.

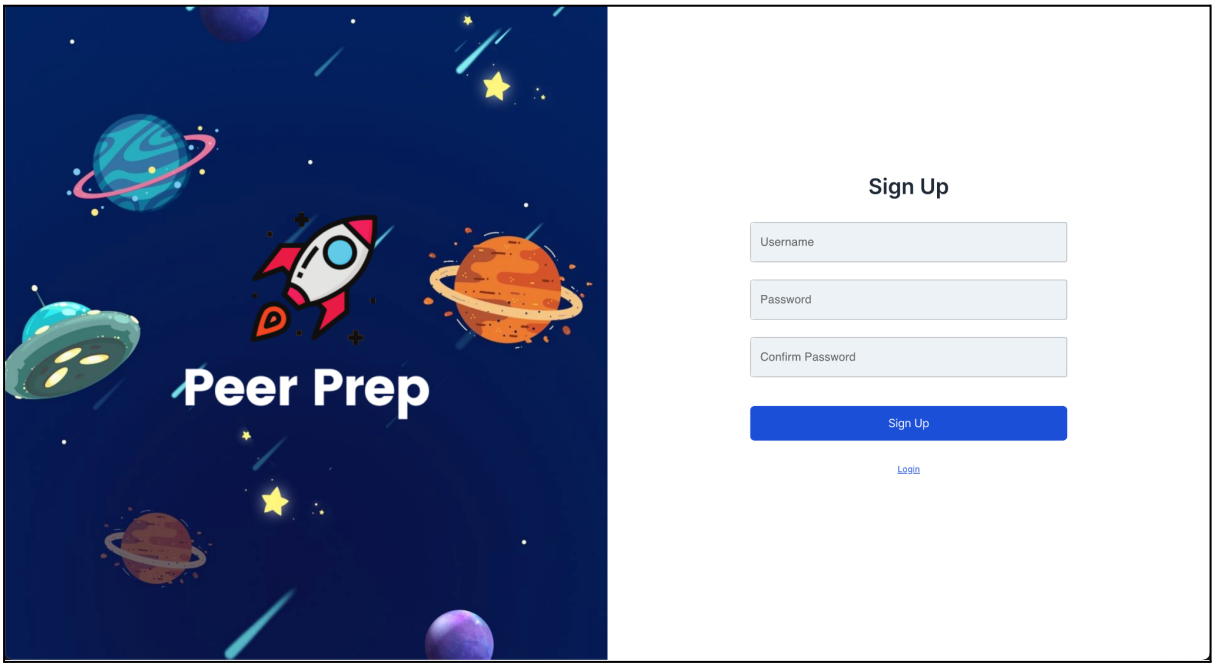
```yaml
name: Node.js CI / Heroku CD

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        node-version: [18.x]

    steps:
      - uses: actions/checkout@v2
      # Continuous Delivery - Frontend
      - uses: akhileshns/heroku-deploy@v3.12.12
        with:
          heroku_api_key: ${{secrets.HEROKU_API_KEY}}
          heroku_app_name: "peerprep3219" #Must be unique in Heroku
          heroku_email: "siddharth.srivastava.2024+cs3219@gmail.com"
          appdir: frontend
      # Continuous Delivery - User Service
      - uses: akhileshns/heroku-deploy@v3.12.12
        with:
          heroku_api_key: ${{secrets.HEROKU_API_KEY_STAGE}}
          heroku_app_name: "peerprep-cs3219-userservice" #Must be unique in Heroku
          heroku_email: "shermannws@gmail.com"
          appdir: user-service
      # Continuous Delivery - Question Service
      - uses: akhileshns/heroku-deploy@v3.12.12
        with:
          heroku_api_key: ${{secrets.HEROKU_API_KEY_STAGE}}
          heroku_app_name: "peerprep-cs3219-qnservice" #Must be unique in Heroku
          heroku_email: "shermannws@gmail.com"
          appdir: question-service
      # Continuous Delivery - Matching Service
      - uses: akhileshns/heroku-deploy@v3.12.12
        with:
          heroku_api_key: ${{secrets.HEROKU_API_KEY_STAGE}}
          heroku_app_name: "peerprep-cs3219-matchservice" #Must be unique in Heroku
          heroku_email: "shermannws@gmail.com"
          appdir: matching-service
      # Continuous Delivery - History Service
      - uses: akhileshns/heroku-deploy@v3.12.12
        with:
          heroku_api_key: ${{secrets.HEROKU_API_KEY_STAGE}}
          heroku_app_name: "peerprep-cs3219-historyservice" #Must be unique in Heroku
          heroku_email: "shermannws@gmail.com"
          appdir: history-service
      # Continuous Delivery - Session Service
      - uses: akhileshns/heroku-deploy@v3.12.12
        with:
          heroku_api_key: ${{secrets.HEROKU_API_KEY_STAGE}}
          heroku_app_name: "peerprep-cs3219-sessionservice" #Must be unique in Heroku
          heroku_email: "shermannws@gmail.com"
          appdir: session-service
```

We have set GitHub secrets for the deployment keys for privacy purposes and it can clearly be seen that each microservice or the frontend code has its own respective heroku application and thus its separate endpoint.

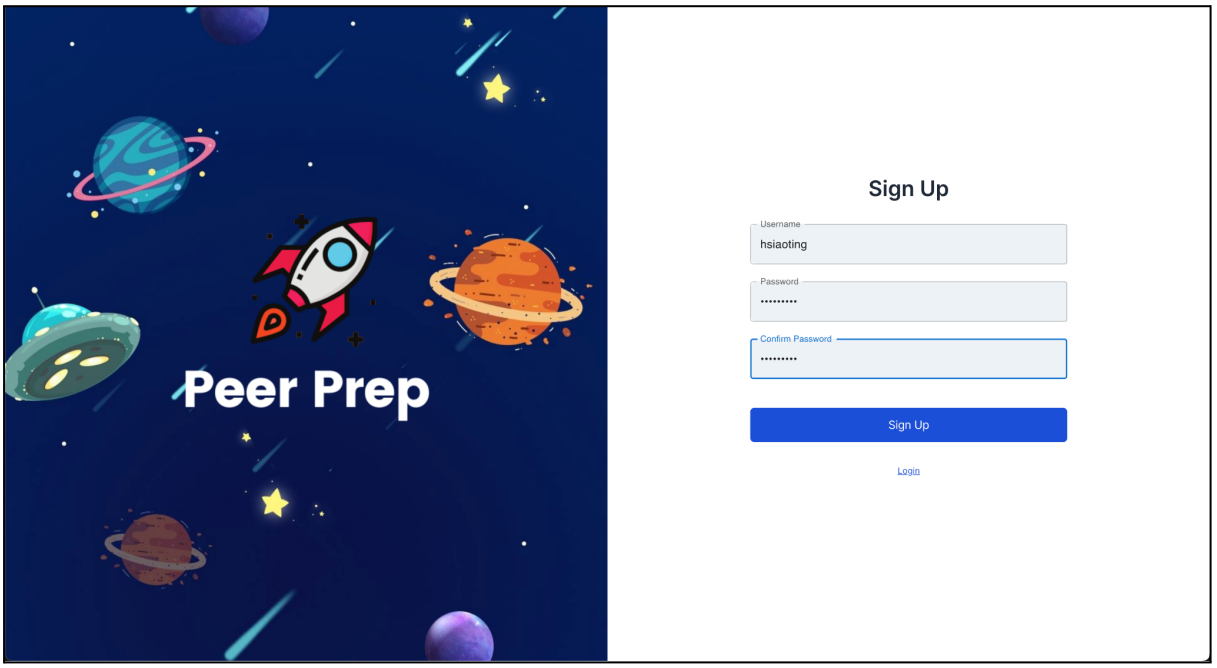
Heroku is the deployment service we decided to use since it allows a simple integration to our github repository while ensuring the stability of the endpoints. The main reason for using Heroku was that it was free and allowed us to deploy the maximum number of endpoints while the only disadvantage of Heroku is that it sets the endpoints to sleep if they are inactive.

9. Application Screenshots

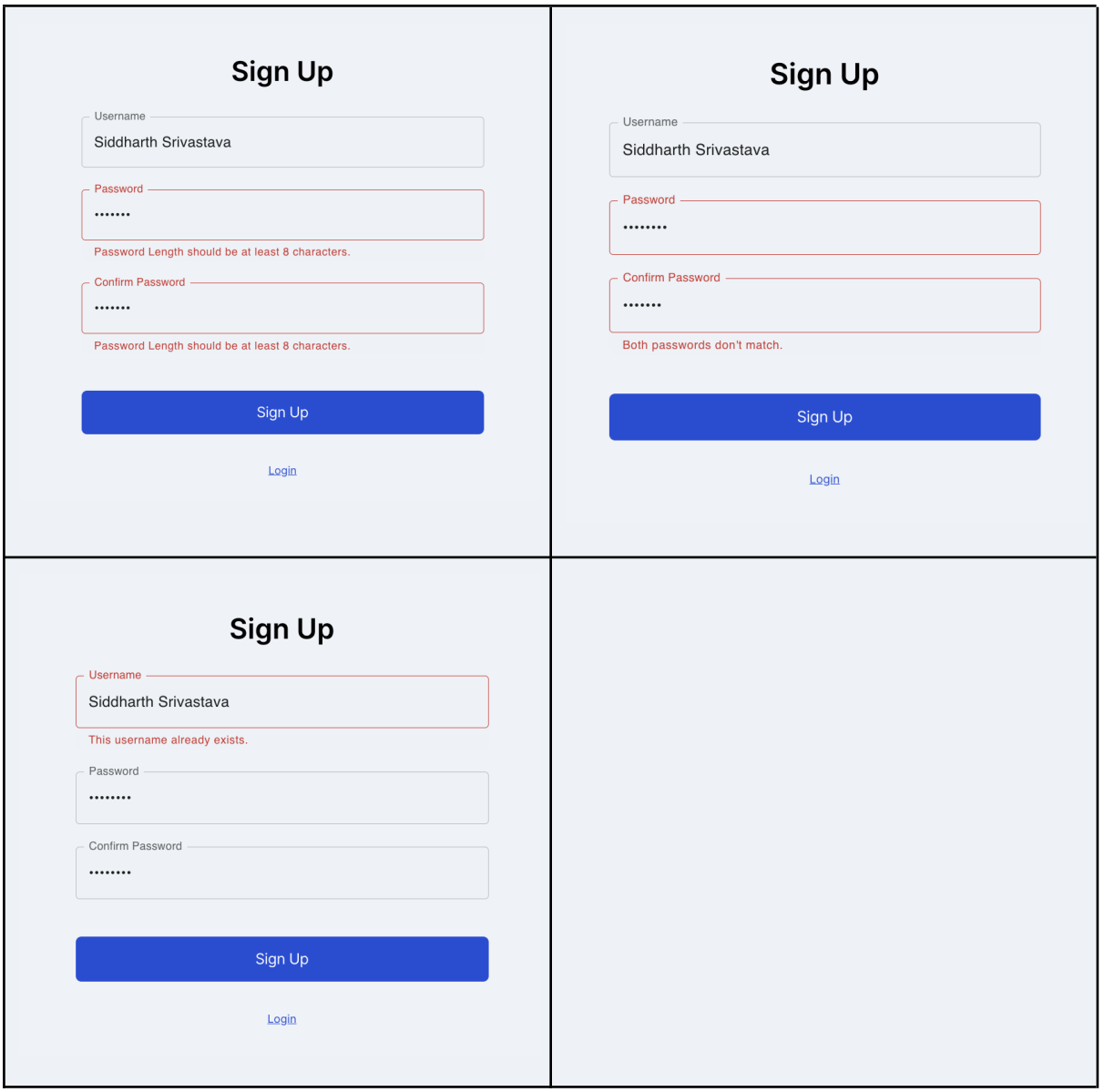
9.1 Sign Up Page



Allows user to sign up for an account with unique username and set a password.

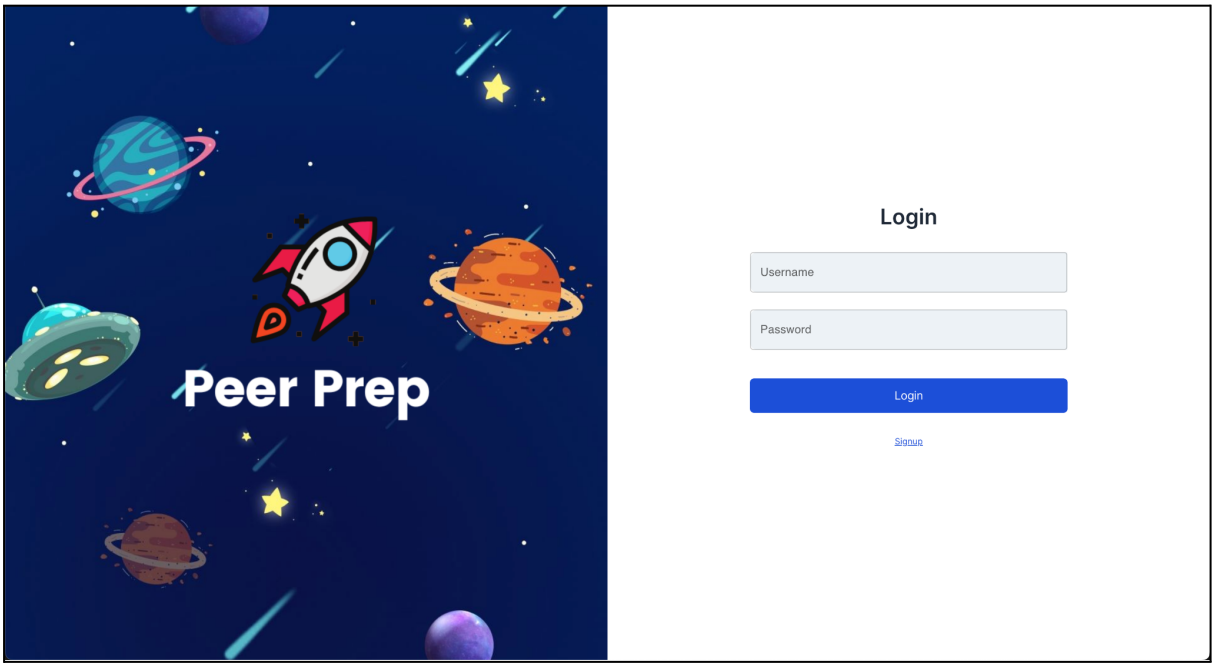


Passwords are hidden and pasting of password in the confirm password text box is disabled for security reasons.

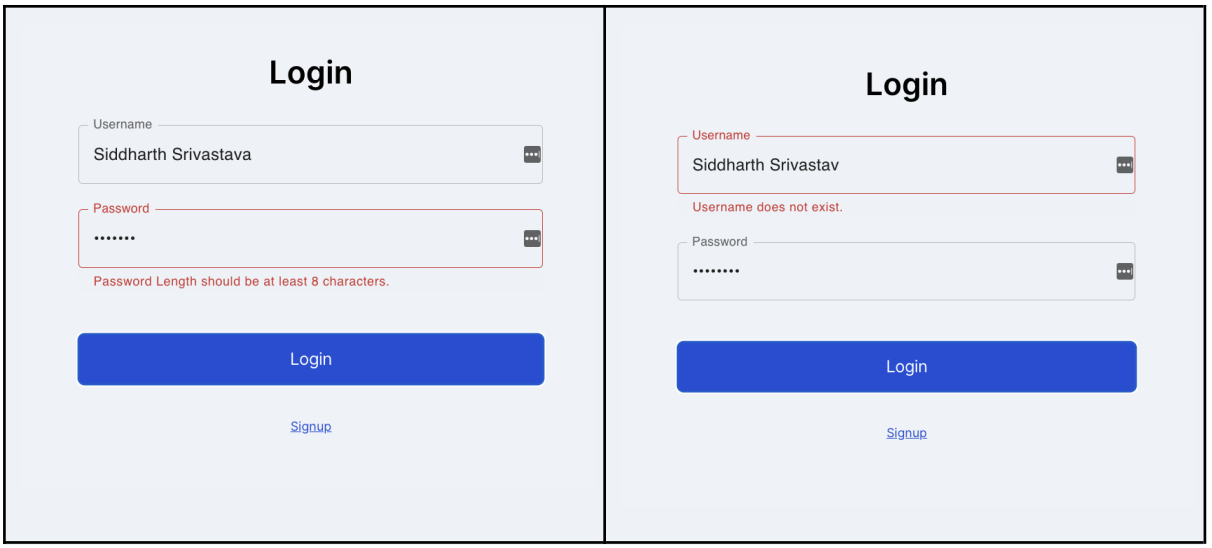


Variety of error validations and meaningful error messages, such as password length should not be less than 8 characters and passwords do not match, to inform and offer corrections to the users.

9.2 Log In Page

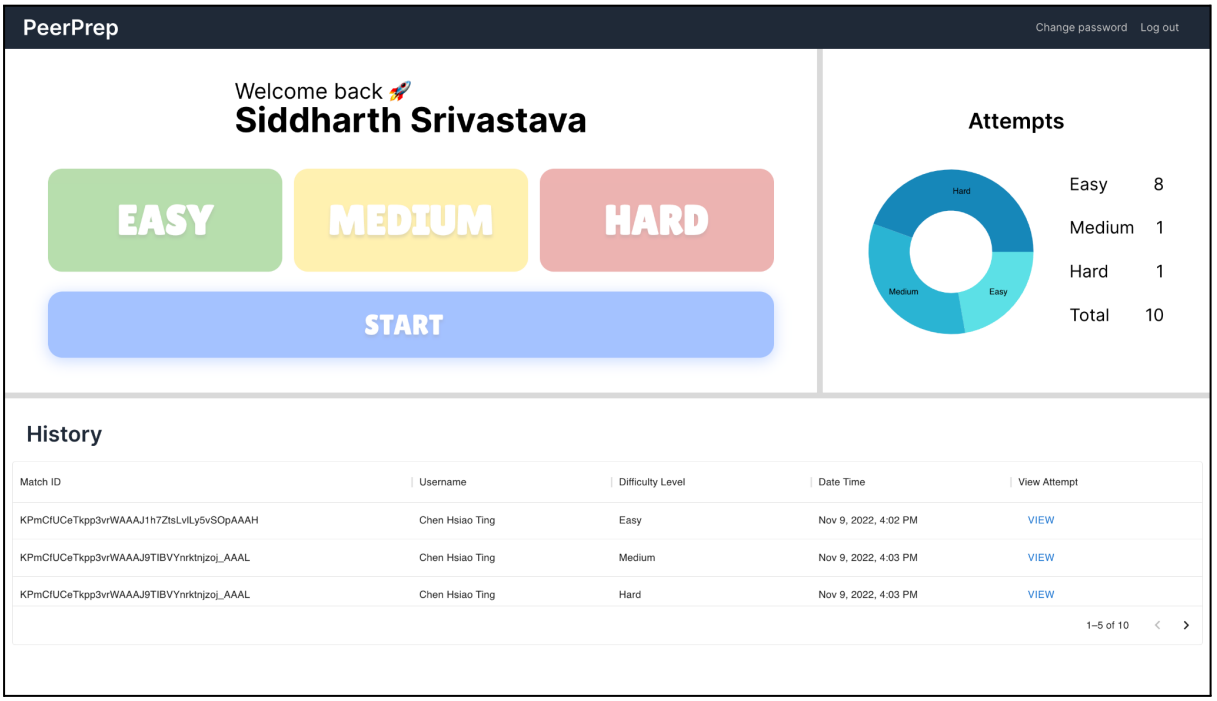


Allow existing users to login with username and password.

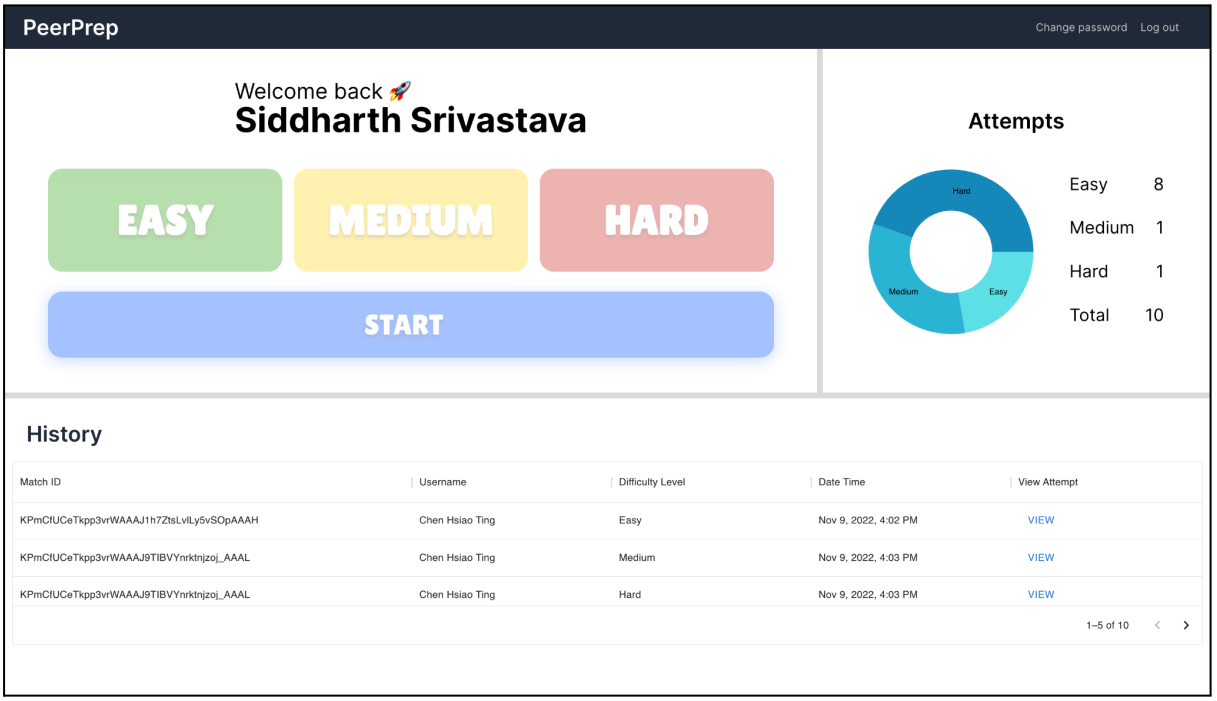


Variety of error validations to inform and offer corrections to the users.

9.3 Dashboard Page

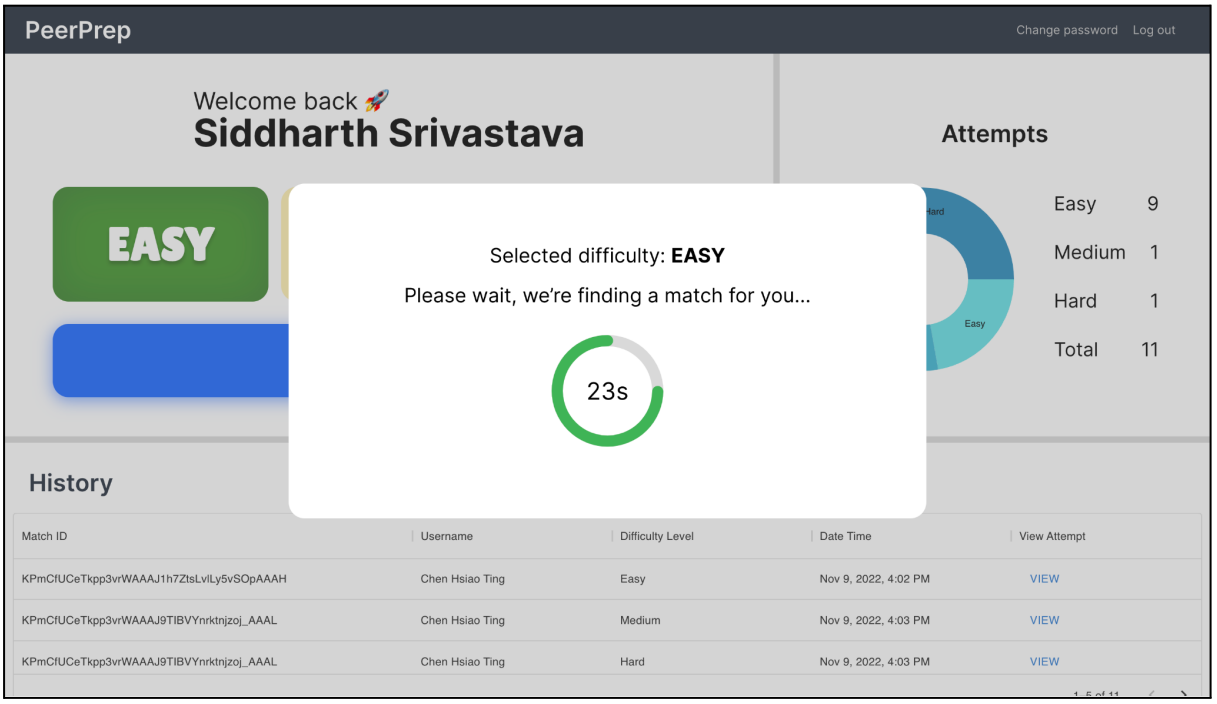


Dashboard page which includes features and/or user interactions such as selection of difficulty level, number of attempted questions by difficulties, user history and other settings like change password and log out.



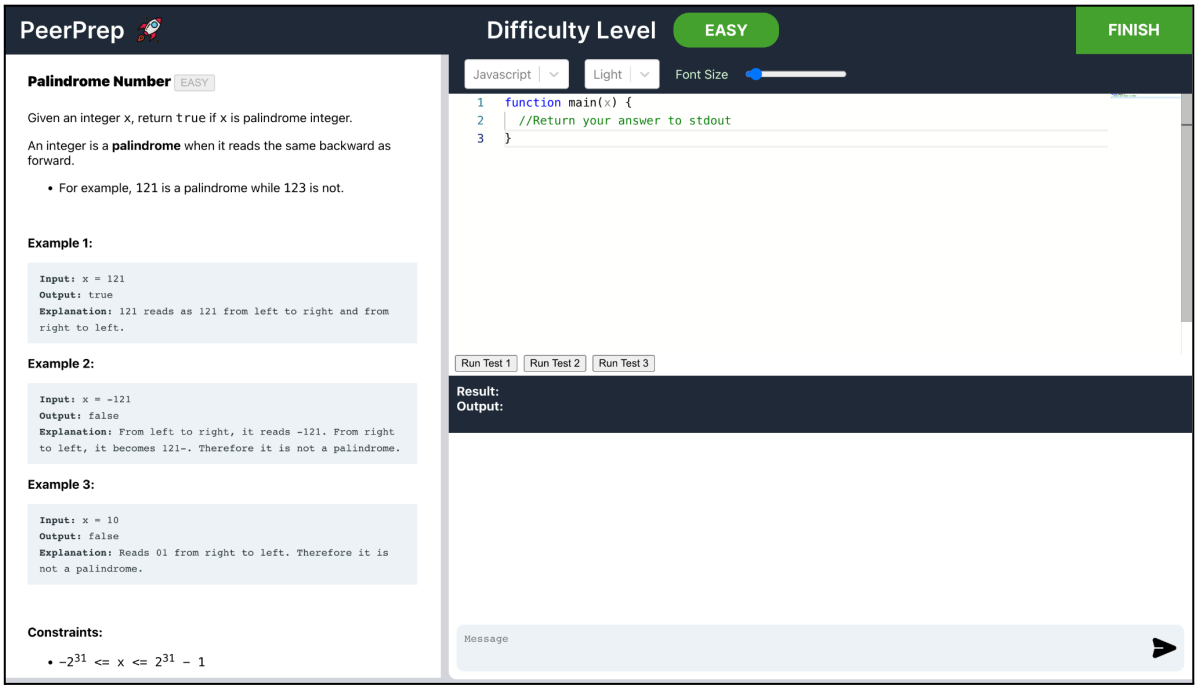
Users can select the desired difficulty level and start finding for a match to attempt a question with a peer.

9.4 Matching Page



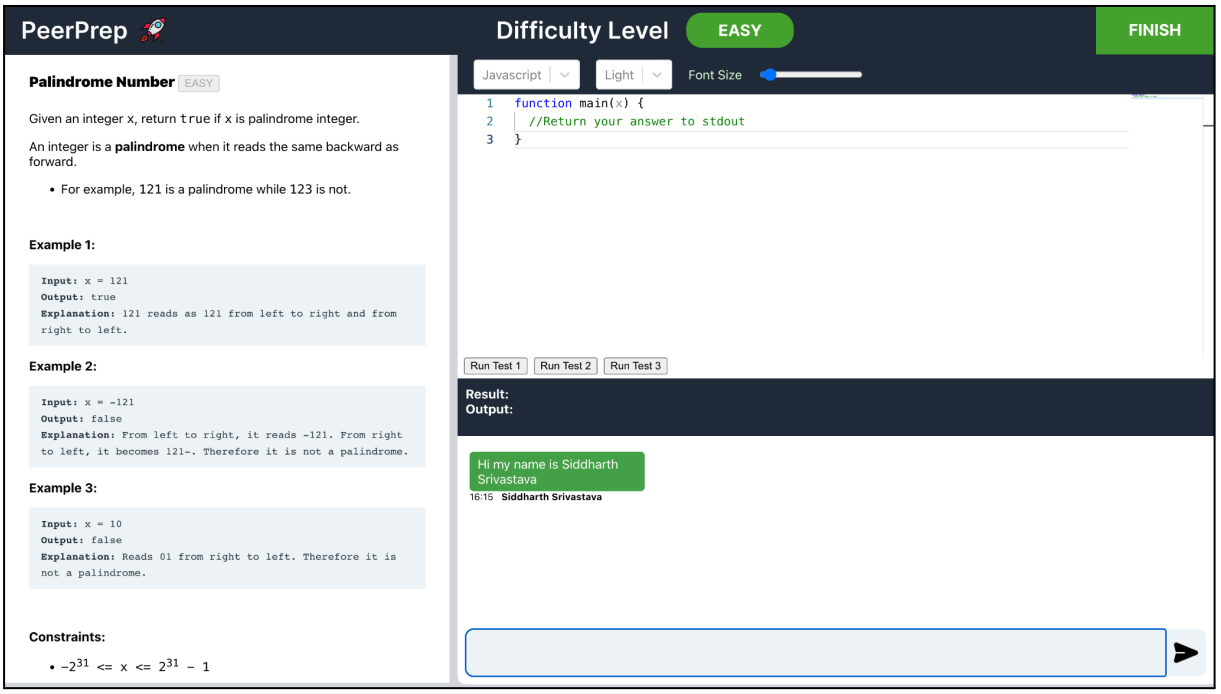
Matching pop up with colour animated to count down timer to match two users who have chosen the same difficulty level within 30 seconds. The pop up automatically closes after 30 seconds. If two users are successfully matched they are able to enter the room and start their attempt.

9.5 Room Page



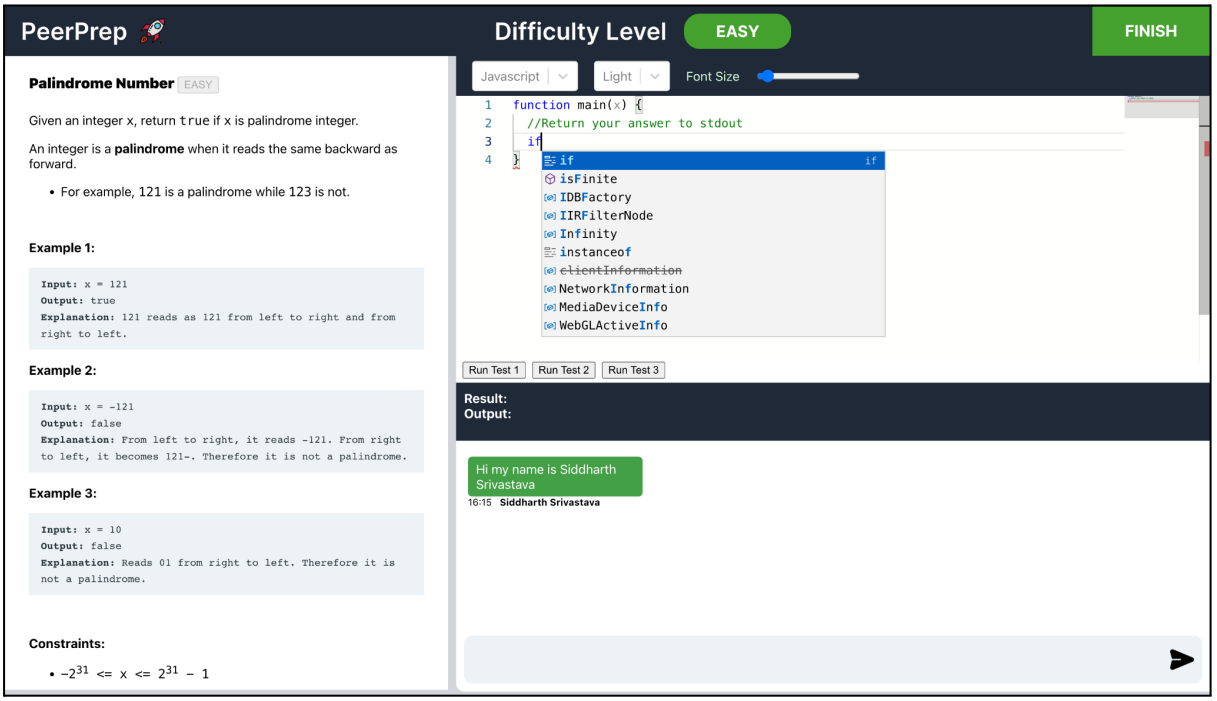
Users are able to seamlessly code with their partner and are able to view the question, code and message at the same time without requiring to scroll or change pages.

Messaging:



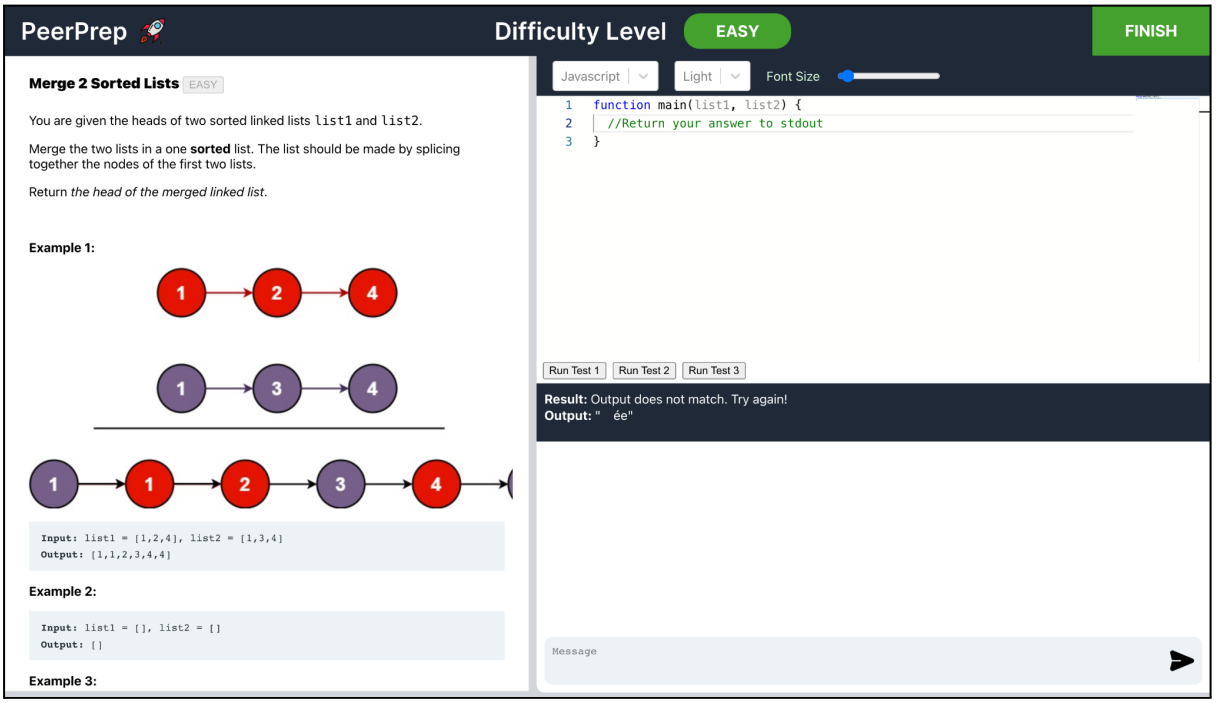
Two matched users can communicate real-time via the chat box at the bottom right of the screen. Information of the time that a message was sent and the username will be displayed to both users. Both users can also view the chat history by scrolling inside the pane.

Code Editing:

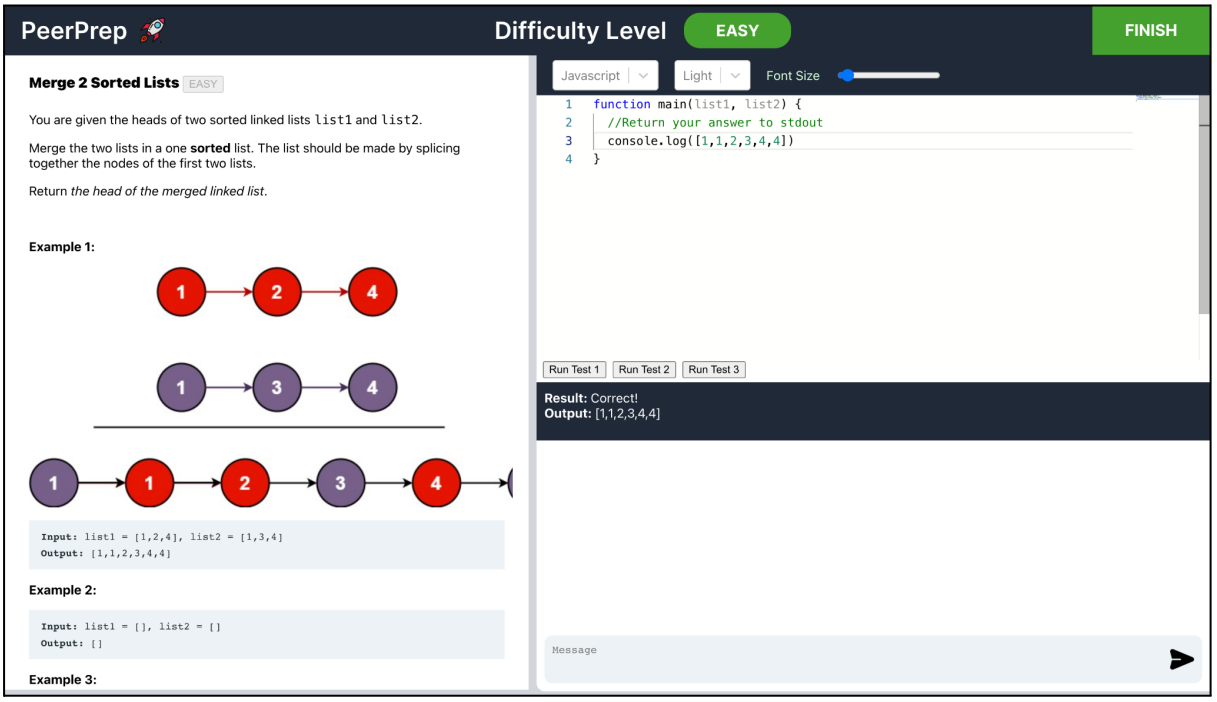


PeerPrep offers code suggestions and colored texts that aid users in programming. Our text editor we have incorporated runs static checks on the code and recommends syntax as seen in the picture above.

Compilation:

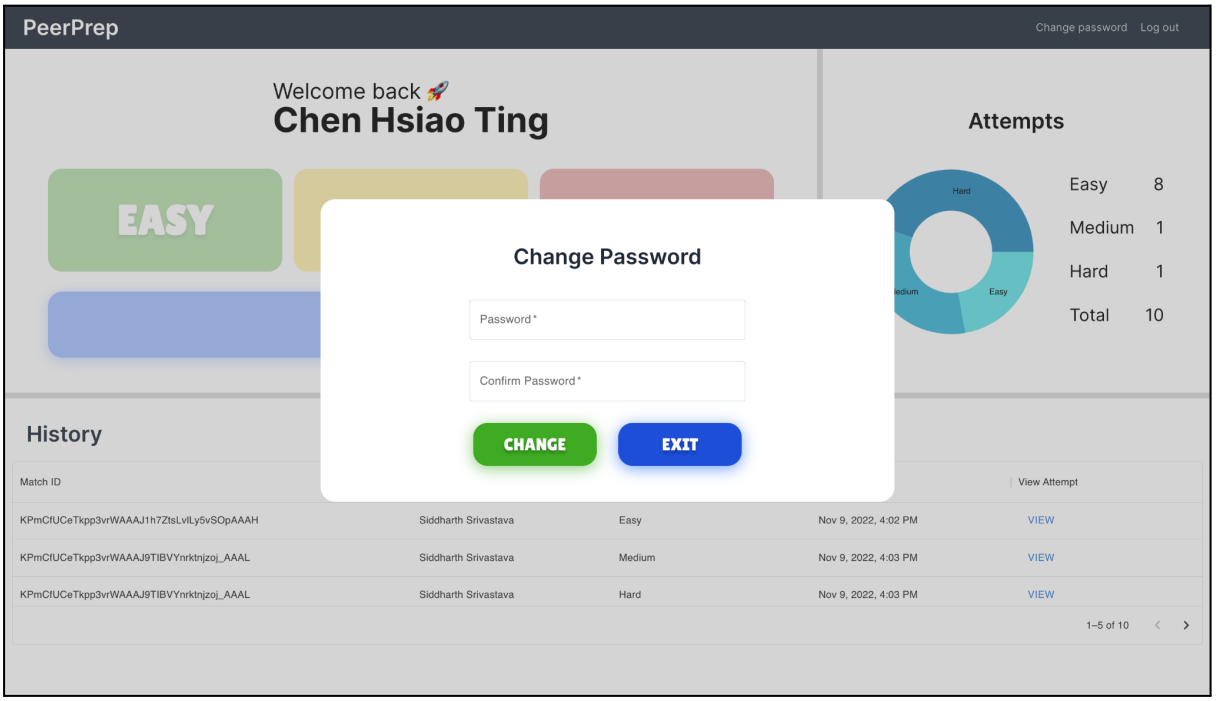


PeerPrep offers compilation and testing features at the bottom of the code editor allowing the user to run the test cases as seen on the question panel as examples. The compiler integrated allows use to check the syntax of the code and run each test case independently

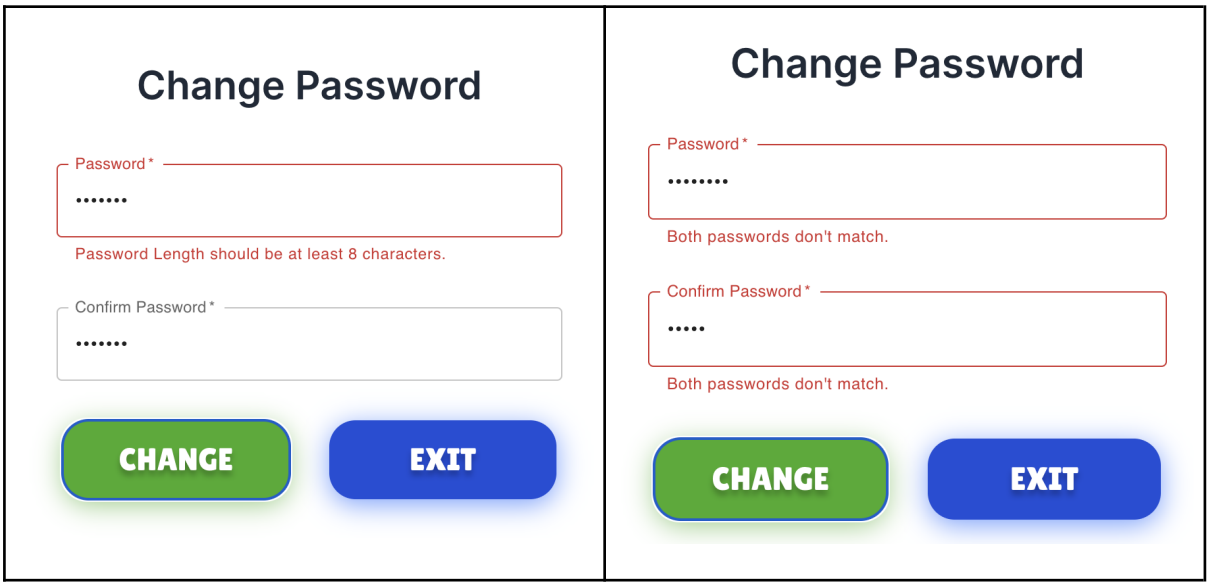


The two screenshots in this section show both unsuccessful and successful test case run.

9.6 Change Password



Users can change his or her password with ease from the dashboard by clicking on the button in the header.



Variety of error validations and meaningful error messages, such as password length should not be less than 8 characters and passwords do not match, to inform and offer corrections to the users help users added passwords successfully.

10. Suggested Enhancements

This project has immense potential to do much more. As we were working on this project, we also noted some suggested enhancements that we believe are key improvements that can be made, if given more time.

1) One of the most significant enhancements is the addition of an API Gateway to help to redirect requests from the frontend to the various microservices. This will not only simplify the authorisation but also routing between the pages.

2) The communication logic could be decoupled from the matching service. This communication microservice can be a standalone service that integrates chatting with video and audio communication. This would be beneficial for users on PeerPrep platform due to the increased demand for video-based coding interviews in recent times. Some APIs that we are considering are Twilio.

3) We also propose having support for more programming languages in the code editor and code compiler. Essentially, the delivered application that we have for this submission underpins the technical feasibility of the implementation. However, being integrated with more programming languages would definitely make this application more relevant to more users.

4) As the application scales, an important quality attribute to consider would be scalability. We propose load testing our APIs and make sure that the application can support hundreds of peer programmers coding at the same time.

11. Reflections and Learning Points

1. It is important to start integration and deployment early. In our case, our development was largely done locally. However, integration and deployment surfaced many other issues that were not relevant when developing locally. Cookie storage, environment variables, and latency are just some of these issues. This is a valuable takeaway for any future assignment and even in our careers to avoid last-minute nightmares.

2. Progressive documentation is key. As developers, we should not solely focus on the code but we should also focus on communicating the value in our programmes. This allows for more effective communication among team members and with other stakeholders in the future.

3. Collaborative learning through effective communication. Through this experience, we have learnt that we have so much to learn from one another. Sometimes, we just have to ask and someone within the group already has an answer to our queries. Through which, we also learn from one another, and help one another become a better version of ourselves.

In all, this experience has been fruitful for the entire team. We have learnt through one another's experiences and through our mistakes. Here's to creating more impactful applications using good design principles and patterns!