NUS Orbital 2025
Milestone 3

Yap Jia Wei
Lee Kuan Yi

Table of Contents

Introduction

Posters

Fig 1. Liftoff Poster

Fig 2. Milestone 1 Poster

Fig 3a. Milestone 2 Poster

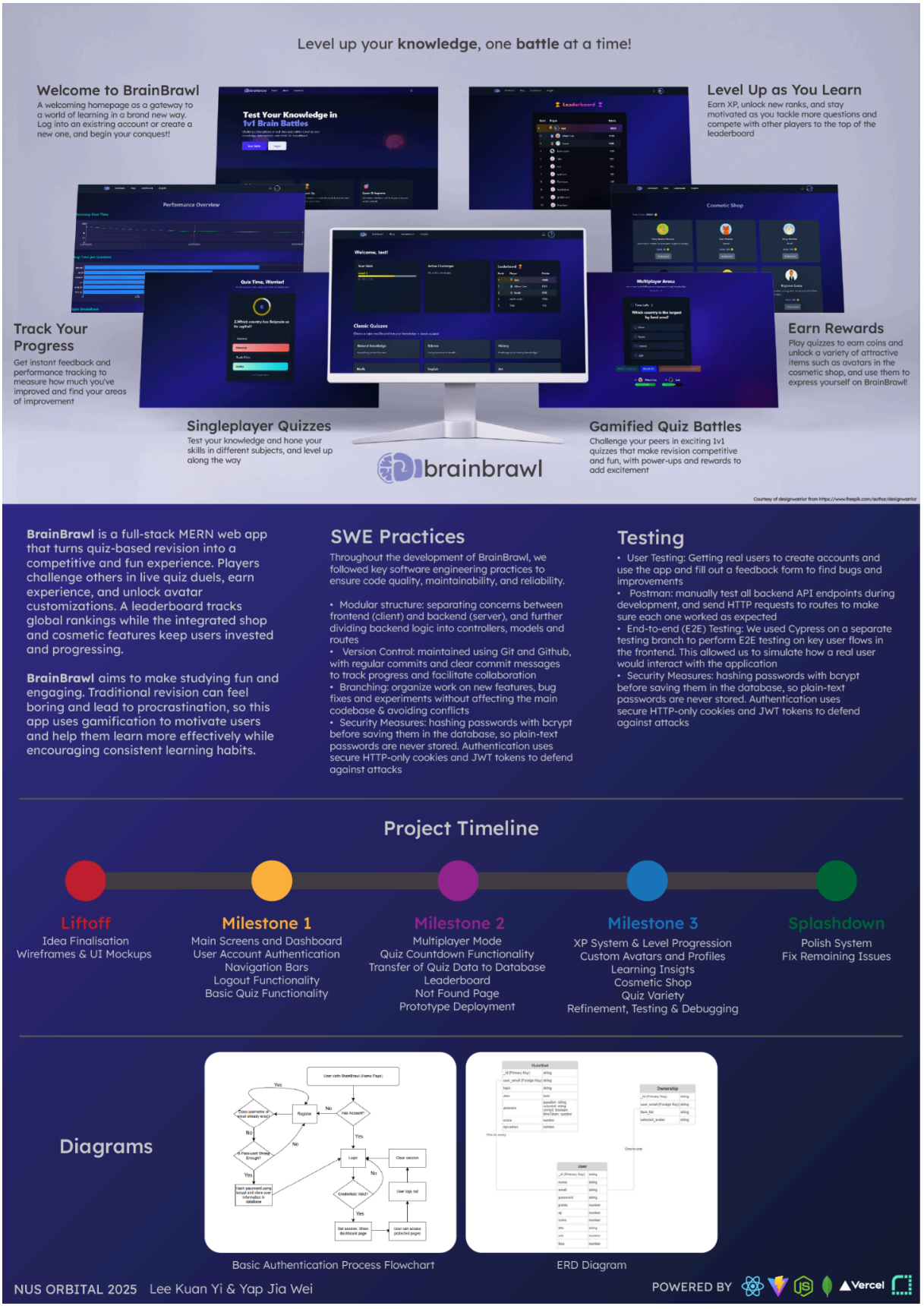


Fig 3b. Milestone 3 Poster

Deployment Link

The web application is deployed on Vercel (frontend) and Render (backend).

Feel free to explore the live demo and test the core features:
https://brainbrawl-frontend.vercel.app/

Proposed Level of Achievement

Apollo 11 — We aim to go beyond basic features with real-time multiplayer, gamification, and user progression systems, fully deployed with complete documentation and UI polish.

Aim

BrainBrawl seeks to make studying fun by combining competitive gaming with academic quizzes. The goal is to create a platform where students revise through real-time 1v1 battles, gain XP, level up, and customize avatars, transforming boring study sessions into exciting challenges.

Motivation

The motivation behind this project is to make studying more engaging and exciting, so that they will learn concepts more effectively and hence be better equipped with relevant toolsets for the future. Traditional revision methods can feel monotonous, leading to procrastination. By introducing gamification, like 1v1 battles, level-ups, and interactive feedback, this app aims to encourage consistent learning habits while keeping users motivated.

Scope

One-Sentence Summary

A real-time quiz battle app that gamifies learning through XP, leveling, and custom avatars.

Descriptive Scope

BrainBrawl is a full-stack MERN web app that turns quiz-based revision into a competitive and fun experience. Players challenge others in live quiz duels, earn experience, and unlock avatar customizations. A leaderboard tracks global rankings while the integrated shop and cosmetic features keep users invested and progressing.

User Stories

1. As a student who wants to revise effectively, I want to participate in quizzes so that I can test and improve my knowledge.
2. As a competitive user who enjoys challenges, I want to engage in 1v1 quiz battles so that I can compete and level up.
3. As a learner who values progress tracking, I want to view my performance history and level progression to stay motivated.

Target Users

BrainBrawl is for **students**, **casual gamers**, and **competitive learners** who enjoy fun, fast-paced quizzes. Those who want to challenge their friends in real-time and keep track of their scores. The app is great for studying, classroom games, or just having fun.

Technology Stack

- Frontend: React + Vite + TailwindCSS
- Backend: Node.js + Express.js + Socket.io
- Database: MongoDB (hosted via MongoDB Atlas)
- Authentication: JWT + bcrypt
- Version Control: Git + GitHub
- Design: Canva (posters), Draw.io (flowcharts), Figma (UI mockups)

Features

Basic Feature: Login & Registration (implemented)

[Description]

BrainBrawl includes a secure user authentication system to ensure only authorized users can access the platform and participate in quizzes. This system helps prevent fake or duplicate accounts and protects user data.

Registration and Login Flow:

- **Registration**: Users sign up with a unique username, a valid email address, and a strong password (minimum 6 characters). The backend checks for

existing email and securely hashes passwords before storing them.

Note: Email verification is not yet implemented; users can log in immediately after registering (this will be fixed in milestone 3)

- **Login**: Users log in with their email and password. The backend verifies credentials and, if valid, issues a JWT token stored as an HTTP-only cookie for secure session management.
- **Logout**: Logging out clears the JWT cookie, ending the session and preventing access to protected pages.

[Technical Proof of Concept]

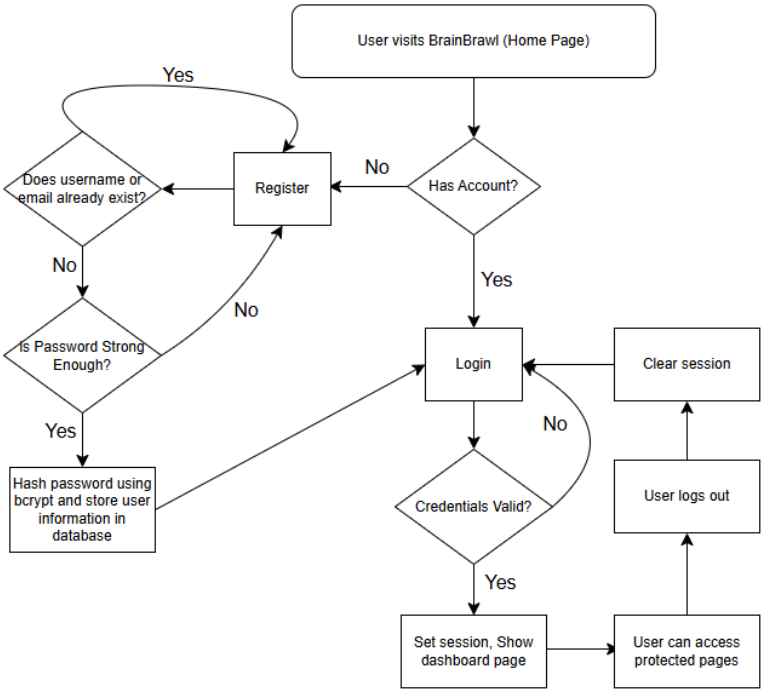


Fig 4. Basic Authentication Process Flowchart

When a user registers on BrainBrawl, they provide a unique username, a valid email, and a strong password. The backend checks if the username or email already exists, and if not, it hashes the password securely using bcrypt before saving it in the database. When logging in, the user's credentials are checked against the stored data: the backend compares the entered password with the hashed version in the database. If the credentials are correct, the backend creates a secure session using a JWT token stored in an HTTP-only cookie, which keeps the user logged in across page refreshes. On each page load, the frontend checks with the backend to see if the user is still authenticated, and only allows access to protected pages if the

session is valid. Logging out clears the session cookie, ensuring the user is securely signed out. This process keeps user data safe by never storing plain-text passwords and by protecting sessions with secure cookies.

[User Guide]

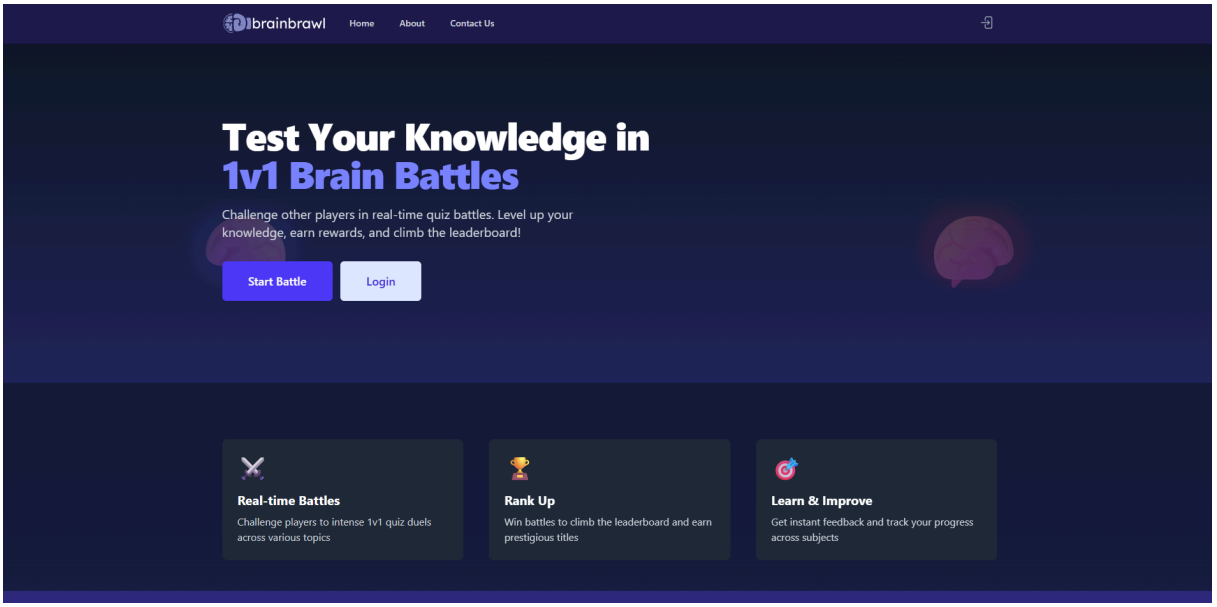


Fig 5. Home Page UI

For new users:

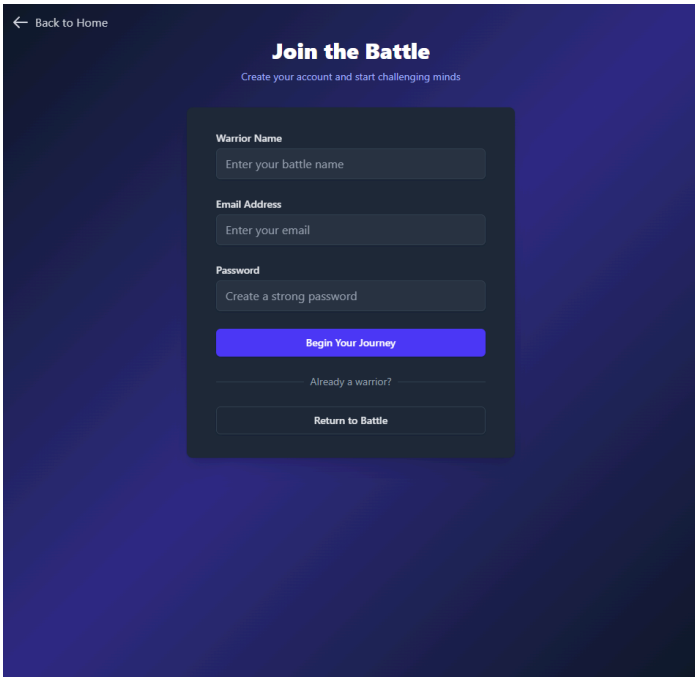


Fig 6. Registration Page UI

1. Click “Start Battle” on the home page as shown on Fig 5.
2. When directed to the registration page as shown on Fig 6., enter your username, email address and password.
3. Click “Begin Your Journey” to create your account.

For returning users:

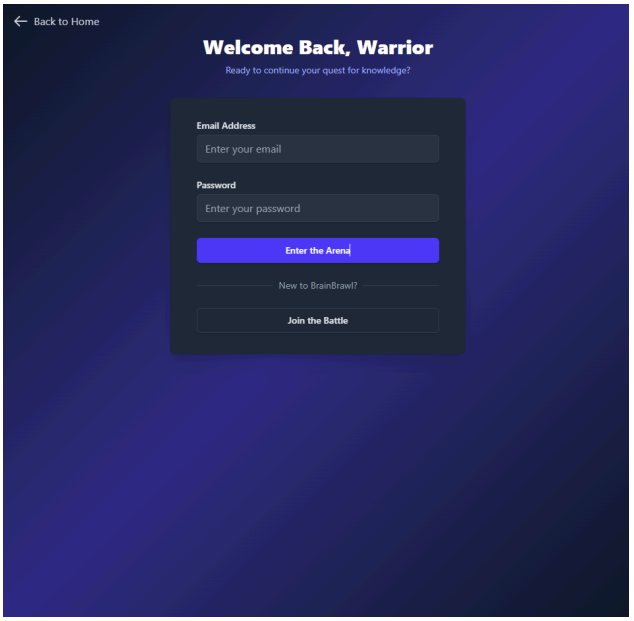


Fig 7. Login Page UI

1. Click “Login” on the home page as shown on Fig 5.
2. When directed to the login page as shown on Fig 7., enter your email address and password.
3. Click “Enter the Arena” to access your BrainBrawl Account.

Core Feature 1: Quiz System with Question Pool **(implemented)**

The quiz system allows users to take quizzes from a pool of questions, with various topics to choose from, providing a variety of engaging quizzes that keeps users interested. The quiz system is designed to be extensible, allowing for easy addition of new topics and questions in the future.

```
const mongoose = require('mongoose');

const { Schema } = mongoose;


const quizSchema = new Schema({

   ans: Number,
```

```
    option1: String,
    option2: String,
    option3: String,
    option4: String,
    question: String,
});


function getQuizModel(topic) {
    const modelName = `${topic}quiz`;
    if (mongoose.models[modelName]) {
        return mongoose.model(modelName);
    }
    return mongoose.model(modelName, quizSchema, `${topic}quizzes`);
}


module.exports = { getQuizModel };
```

Fig 8. Singleplayer Quiz Schema

The quiz system employs frontend and backend functionalities, with the questions and answers written down and stored in the MongoDB database in the form of a collection. When the quiz starts, the backend server retrieves the questions and their options from the MongoDB collection and sends it to the frontend code responsible for displaying the quiz page, and the user is presented with these questions and multiple-choice answers, each question at a time. When the user selects an answer for a question, the functions in the frontend code will check from the question data if it is correct or not.

Once the user selects an option, the answer will be revealed instantly, and the user cannot select any other options and will automatically be redirected to the next question. If the answer is correct, the user gains 1 point, and if it is incorrect, the user does not lose or gain any points.

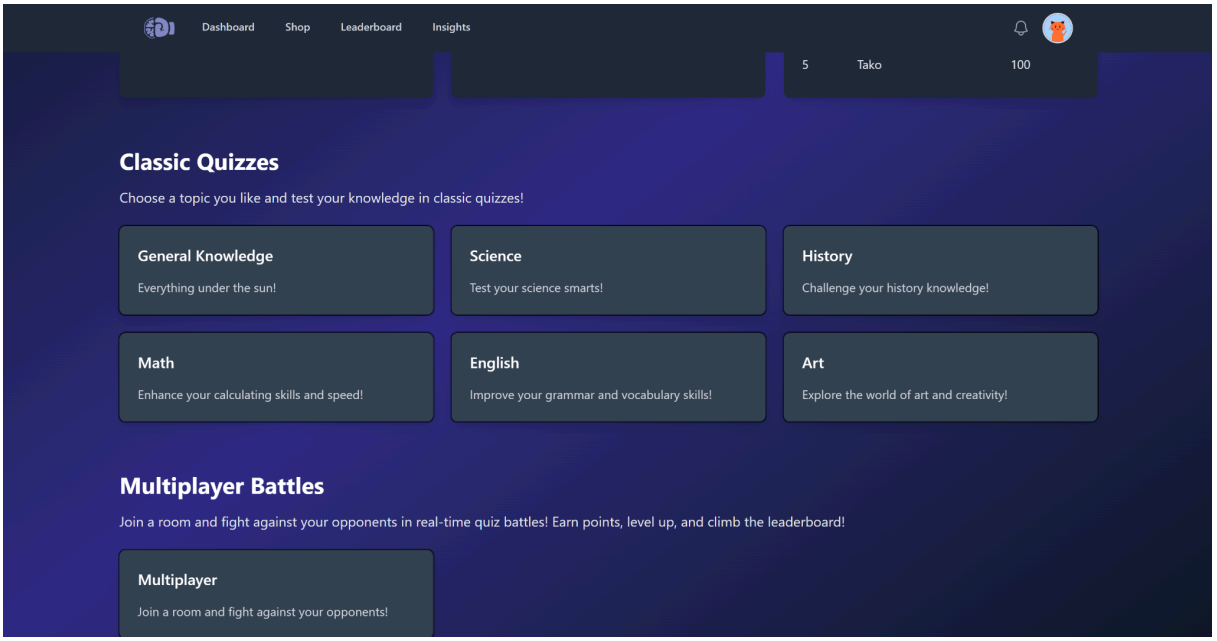


Fig 9. Quiz Selection Menu on the Dashboard Page (with the Multiplayer button)

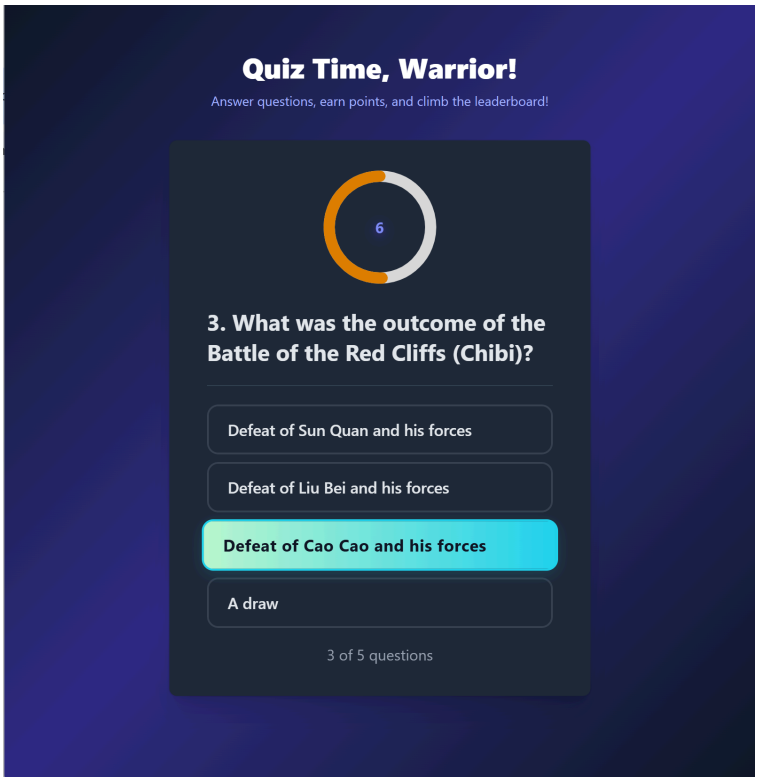


Fig 10. Quiz UI with Timer and correctly selected option in green

After the quiz ends, the user is presented with their score (the number of questions answered correctly) and is notified with a toast message that the user has earned XP. The user can earn up to 100 XP. Then the user can exit the quiz interface and return to the dashboard.

Feedback and statistics are collected such as accuracy and average time per question, and they constitute part of the Learning Insights functionality.

Core Feature 2: Real-time 1v1 Battle Mode **(implemented)**

The multiplayer system bears similarities to the singleplayer quiz with the same quiz format and similar answer selection functionality. However, it also has many differences. One obvious difference is that it enables more than one player to versus each other in a race to be the last one standing.

The backend server is responsible for retrieving the multiplayer quiz questions from the MongoDB database via a Mongoose schema and transferring them to the frontend code that displays the content.

The feature uses the Socket.io plugin, and the backend serves as the Socket.io server. When a user opens the Multiplayer page, the user will be prompted to enter a room code. Once the room code is entered, the user can wait in the waiting lobby for another user to join the room. Once another user has joined, the user can click on the “Ready” button, and once the other player does the same too, the frontend will send a Socket.io message to the backend to notify the start of the battle, and all players will be shown the questions.

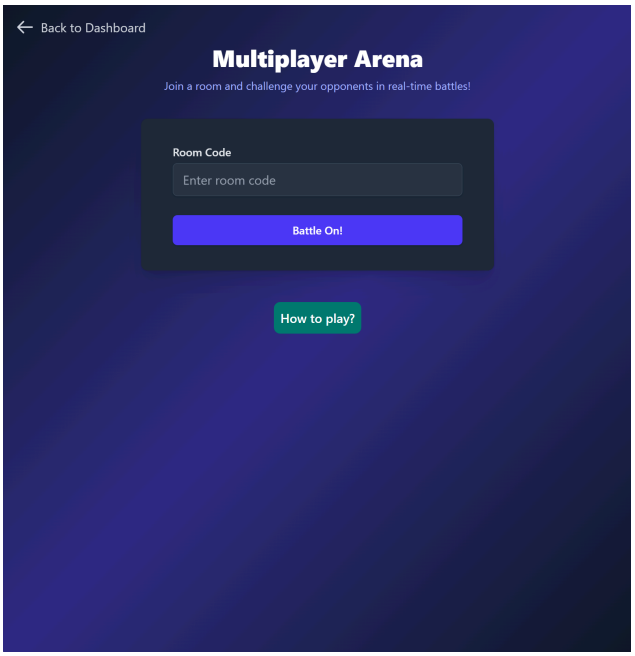


Fig 11. Multiplayer Room Code Page

The questions retrieved from the backend are randomised in no particular order. Below the question list are health bars of each player that start with maximum health of 3. If the user answered a question correctly first, there will be a toast message that broadcasts to all players that the user has answered the question correctly, the health bar of the opponent will decrease by 1 health point, and the frontend will send a message to the backend to switch to the next question for all users. If the user answers the question incorrectly, no toast message will appear, no one's health points will be deducted, and the user will be redirected to the next question.

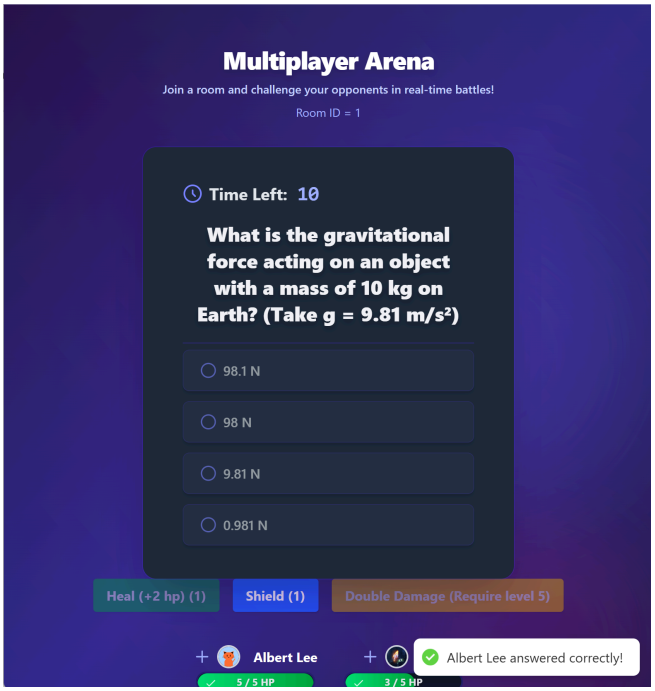


Fig 12. Multiplayer Quiz UI with Toast Message

The game ends when one player's health points are first reduced to zero, and the winner is the one who survives. The frontend will send a message to the backend notifying the server that there is a winner and the server will end the session. The user can click a button to exit the game and return to the dashboard. The winner will earn 100 points, 100 coins and 50 XP, which will be stored in the MongoDB database.

Core Feature 3: Leaderboard **(implemented)**

[Description]

Fig 13. Leaderboard Page

The BrainBrawl leaderboard is a feature that displays the top-performing users based on their quiz scores. It allows users to see how they rank compared to others, encouraging friendly competition and motivating users to improve their performance. The leaderboard shows their ranking, usernames and points. Points are obtained when a user gets a victory against their opponent in the 1v1 quiz game mode.

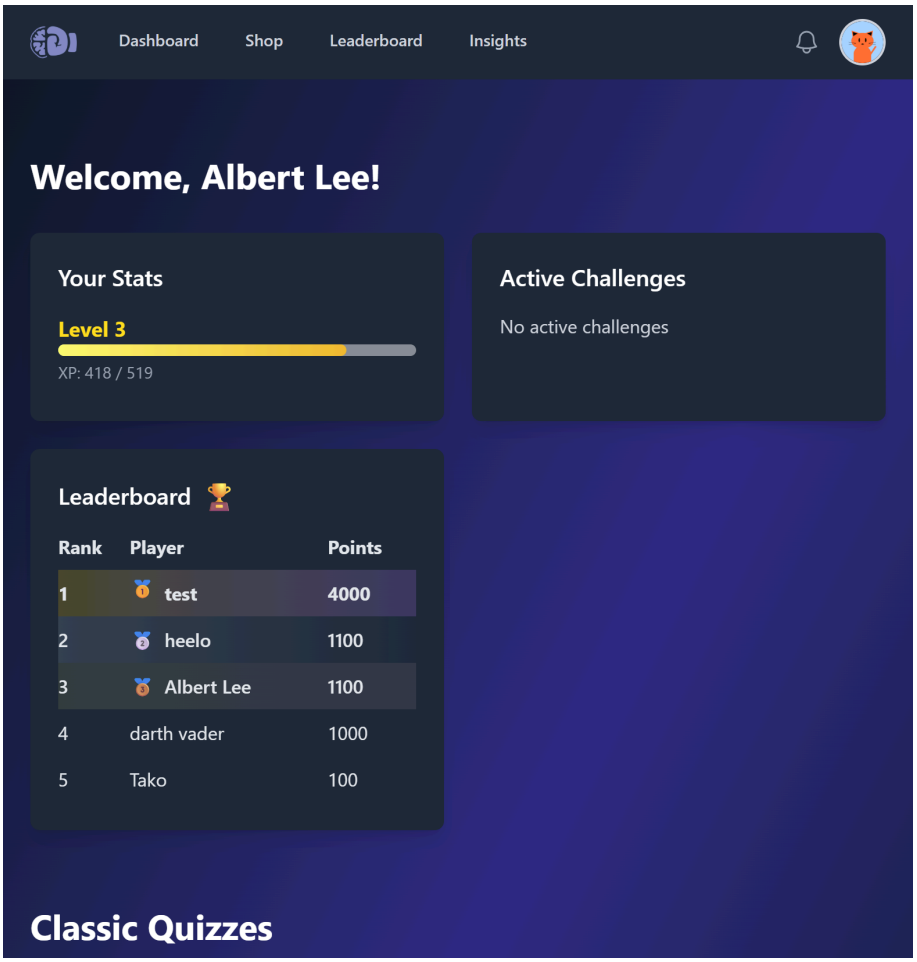


Fig 14. Dashboard page mini leaderboard display

In addition, the dashboard page displays the top 5 ranked global players, this acts as an additional motivation for users to aim to reach the top of the leaderboard ranking.

[Technical Proof of Concept]

```
const mongoose = require('mongoose');
const { Schema } = mongoose;

const userSchema = new Schema({
   name: {
        type: String,
        unique: true,
   },
   email: {
        type: String,
        unique: true,
   },
   password: String,
   points: {
        type: Number,
        default: 0,
   },
   xp: {
```

```
        type: Number,
        default: 0,
    },
    coins: {
        type: Number,
        default: 0,
    },
    title: {
        type: String,
        default: "Noob",
    },
    win: {
        type: Number,
        default: 0
    },
    loss: {
        type: Number,
        default: 0
    }
});

const UserModel = mongoose.model('User', userSchema);

module.exports = UserModel;
```

Fig 15. MongoDB User Schema

The leaderboard is powered by the user data stored in the MongoDB database. Each user document includes a “points” field, which tracks the total points a user has earned from quizzes. When a user wins against their opponent in the 1v1 quiz game mode, their score is added to their “points” in the database.

The backend exposes an API endpoint (‘/leaderboard’) and retrieves all user data from the database. It then sorts them in descending order by their “points”, and returns the data to the frontend which will then display this data in a leaderboard table.

Extension 1: XP System, Level Progression **(implemented)**

The progression level of a user is recorded and measured using XP (Experience Points). As shown in Fig. 15, the ‘xp’ numeric variable is included in the MongoDB User Schema and hence the collection, which stores how much XP the user has in the database. Currently, XP can be earned from playing singleplayer and multiplayer quizzes. When the user loads the dashboard, the backend exposes an API endpoint ‘/level’ and retrieves the user’s data and extracts the ‘xp’ variable from the data. With the help of helper functions that calculate the XP level of the user, how many more XP to the next level etc., the processed XP data is sent to the frontend and displayed in a box on the dashboard page, with the XP Level, a XP progression bar and the

number of XP displayed on it.

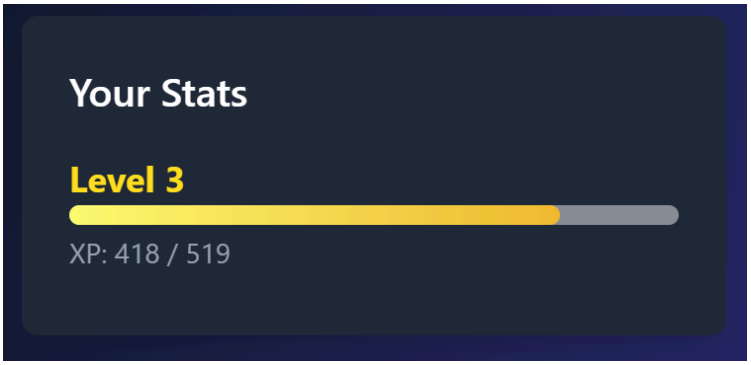


Fig 16. XP Level Box displayed on the Dashboard Page

Extension 2: Custom Avatars and Profiles **(implemented)**

We believe in giving our users the freedom to express themselves on BrainBrawl and modify their account information as they please. Hence, we have implemented a Custom Avatar and Profiles system. Custom avatars are basically static or animated profile pictures that are displayed on the user's profile page, the leaderboard and Navigation Bar etc. to add flair to the user's account.

The avatar images currently available are stored in the frontend with different ID tags assigned to each item. There is a collection in the MongoDB database that stores the IDs of the items/avatars owned by each user that is represented by the Ownership schema shown below in Fig. 17. By default, each user has the "noobbrain" avatar.

```
const mongoose = require('mongoose');
const { Schema } = mongoose;


const itemSchema = new Schema({
   item_id: {
       type: String
   }
})


const ownershipSchema = new Schema({
   user_email: {
```

```
        type: String,
    },
    item_list: [itemSchema],
    selected_avatar: {
            type: String,
            default: "noobbrain"
        }
});


module.exports = mongoose.model('Ownership', ownershipSchema);
```

Fig 17. Ownership MongoDB Schema

A profile page has been set up to display the user's avatar, Progression Level and other information such as the number of wins and losses and the number of quizzes attempted. When the profile page is loaded, the frontend code sends an Axios Get request that triggers the backend to expose the relevant API endpoints to retrieve the number of quizzes attempted from the quizStat schema, XP and user data from the User schema and data of owned items from the Ownership schema (`/quizStats/${user.email}`, '/level', 'profile' and `/ownership/${user.email}` respectively, with user.email referring to the user's email address). All the data is sent to the frontend and displayed on the profile page where necessary.

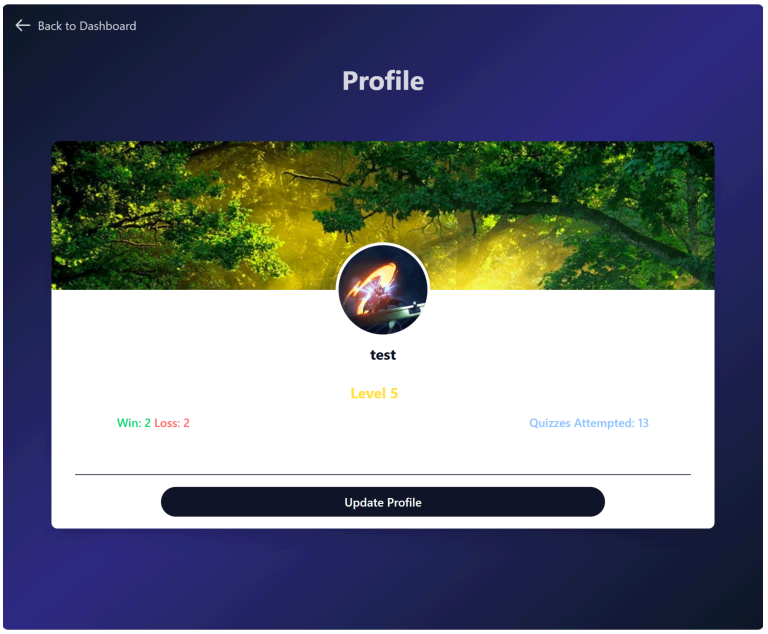


Fig 18. Default Profile Page

An “Update Profile” button is located at the bottom of the page, which allows users to change their username and email address. When clicked, the profile page changes to display a form with Name and Email text fields that display the current username and email address of the user, and all the avatars the user owns. The user can modify their name and email address, and select the owned avatar they want to change into. The user can choose to save or cancel the changes. When the save button is clicked, the frontend code sends an Axios Put request (‘/updateProfile’) to get the backend to expose the API endpoint and send the data to the backend for modification. When successful, the user will be able to see their updated avatar, username and email address.

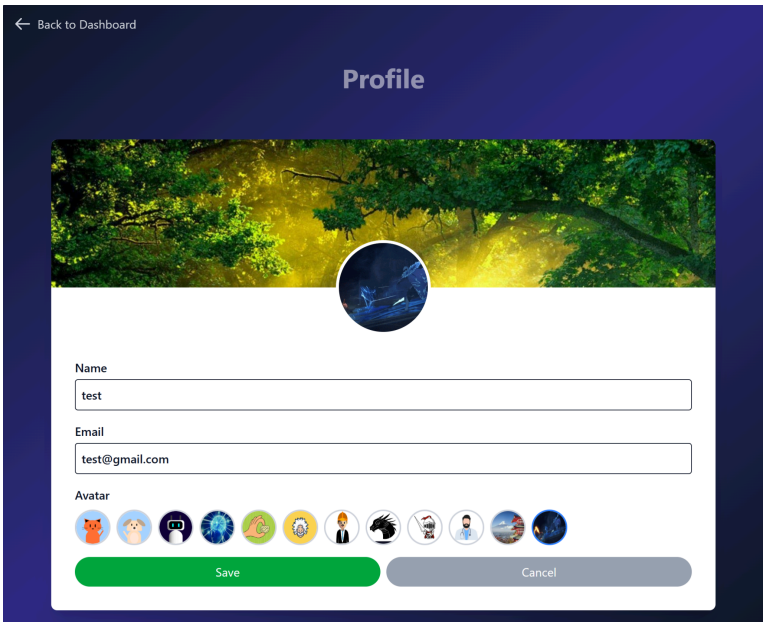


Fig 19. Profile Page in Edit Mode

Extension 3: Learning Insights **(implemented)**

While our aim is to gamify learning and make it more fun and rewarding, we also want to help users improve their educational skills and knowledge. Thus, we have come up with the Learning Insights functionality to provide users statistics on their performance such that they have a clearer idea on where to work on.

The Learning Insights page displays an interactive line chart of the overall accuracy over time across all quiz categories, a bar chart of the average time per question for each singleplayer quiz category, a visual breakdown of the accuracy by topic ranked strongest to weakest, singleplayer quiz history and recommendations on which quiz topic to revisit and practise more, with a link to the quiz category.

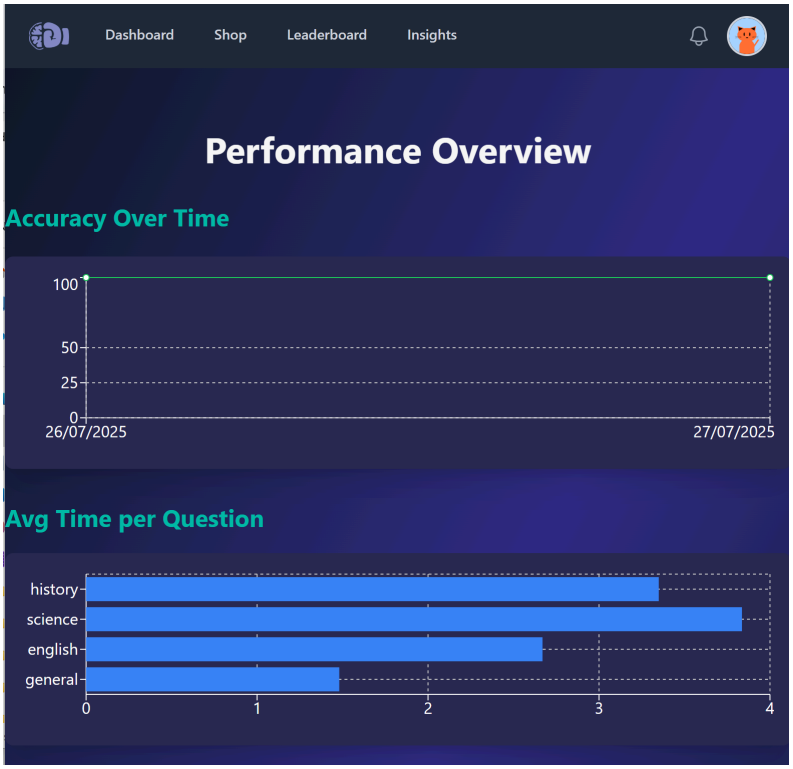


Fig 20a. Learning Insights Page (Top)

Fig 20b. Learning Insights Page (Bottom)

After the user completes a singleplayer quiz, the frontend code will call an Axios Post

command (‘update-quizStats’) to send statistics such as the time, topic, answers and score into the backend and store them into a MongoDB collection defined by a Mongoose Schema called QuizStat, as defined below in Fig. 21.

```
const mongoose = require('mongoose')
const { Schema } = mongoose


const quizStatSchema = new Schema({
   user_email: {
       type: String,
       required: true
   },
   topic: {
       type: String
   },
   date: {
       type: Date, default: Date.now
   },
   answers: [
       {
           question: String,
           selected: String,
           correct: Boolean,
           timeTaken: Number
       }
   ],
   score: Number,
   xpGained: Number
})
```

```
module.exports = mongoose.model('QuizStat', quizStatSchema);
```

Fig 21. QuizStat Schema

When the Learning Insights page is loaded, the frontend Insights.jsx code sends an Axios Get request that signals the backend to get the API endpoint (`/quizStats/${user.email}`), retrieve the statistics from the QuizStats collection via the Schema and send them over to the Insights.jsx code for further calculations e.g. average time taken, accuracy of answers, and then display the processed statistics on the webpage. The weakest subject is determined from the accuracy of the subjects.

Extension 4: Cosmetic Shop **(implemented)**

Of course, the avatars aforementioned have to be obtained from somewhere. To incentivise users to continue playing BrainBrawl's quizzes, we have set up a cosmetic shop where users can redeem items they want. Currently, only avatars can be redeemed from the shop.

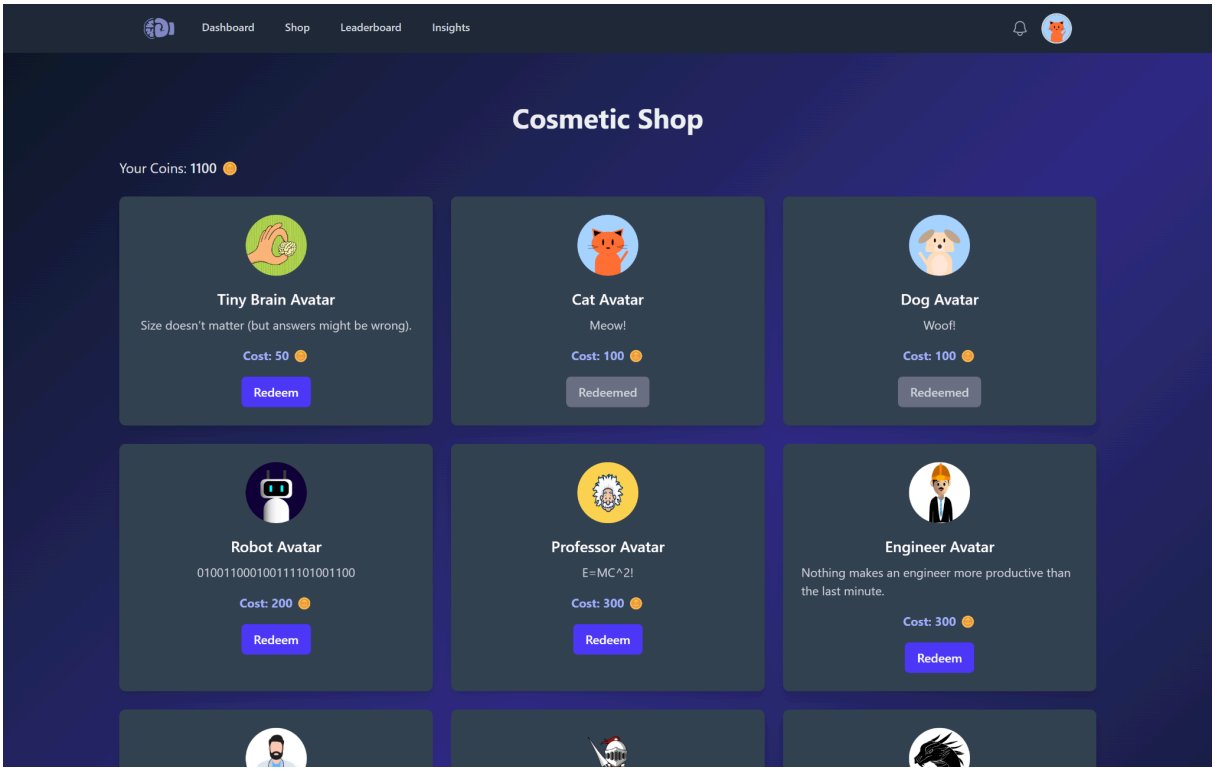


Fig 22. Cosmetic Shop Page

All available avatars are displayed on the shop page. The backend opens the API endpoint '/owned-items' to retrieve data from the Ownership schema (as shown in Fig. 17) and use that data in the frontend to check the items owned by the user. For each item, the redemption button is enabled or disabled by the frontend code depending on whether the user already owns the item or not. The number of coins the user owns is also retrieved in the same fashion, but using the API endpoint

‘/coins’, and then the frontend receives the amount and displays the coins balance at the top of the page. If the user attempts to purchase an item greater than the number of coins they own, the frontend code will check and prevent a successful purchase, and instead send a toast message notifying the user that they have insufficient coins.

In the event the user is able to afford an item/avatar and clicks on the “Redeem” button for it, the frontend code will attempt an Axios Post ('/update-coins') to push the user’s email and the cost of the item into the backend code, where the user’s coins balance in the Users schema is searched up using the email address. When the balance is retrieved successfully, it is subtracted by the cost of the item and reinserted back into the MongoDB collection. Afterwards, another Axios Post ('/update-ownership') is executed, which pushes the user’s email and item ID into the backend to log the item into the user’s owned items list in the Ownerships database, using the user’s email as a key.

Software Engineering Practices

Overview

Throughout the development of BrainBrawl, we followed key software engineering practices to ensure code quality, maintainability, and reliability. The project uses a modular structure, separating concerns between frontend (client) and backend (server), and further dividing backend logic into controllers, models and routes as shown in Fig. 23.

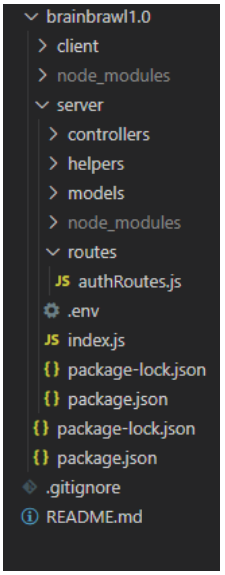


Fig 23. BrainBrawl Directory Layout

Version Control

Version control was maintained using Git and Github, with regular commits and clear

commit messages to track progress and facilitate collaboration.

Branching

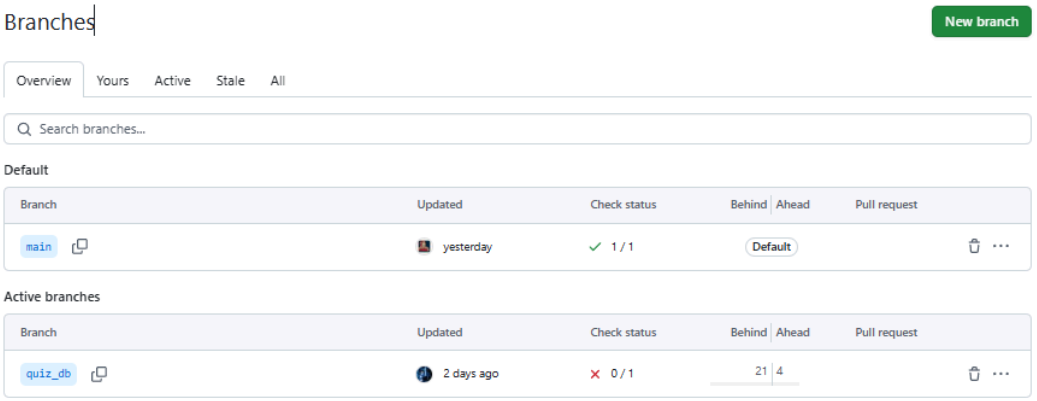


Fig 24. Branches Page on Github

During the development of BrainBrawl, branching was used to organize work on new features, bug fixes, and experiments without affecting the main codebase. Each new feature or fix was developed in its own branch, allowing us to work independently and in parallel. This approach made it easier to manage changes and test new functionality while avoiding conflicts, ensuring that the main branch always remained stable and deployable.

Pull Requests

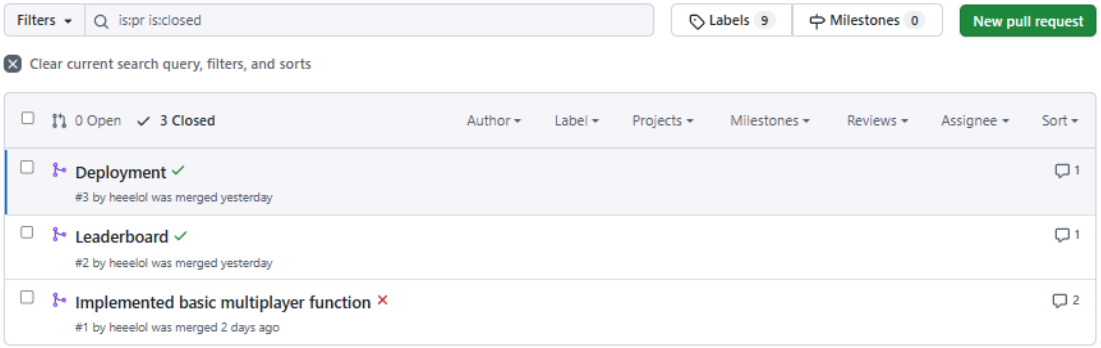


Fig 25. Pull Requests Page on Github

Pull requests allowed us to review and discuss changes before adding them to the main code. This helped catch mistakes early and encouraged collaboration and knowledge sharing among us. This also helped keep a clear history of what was

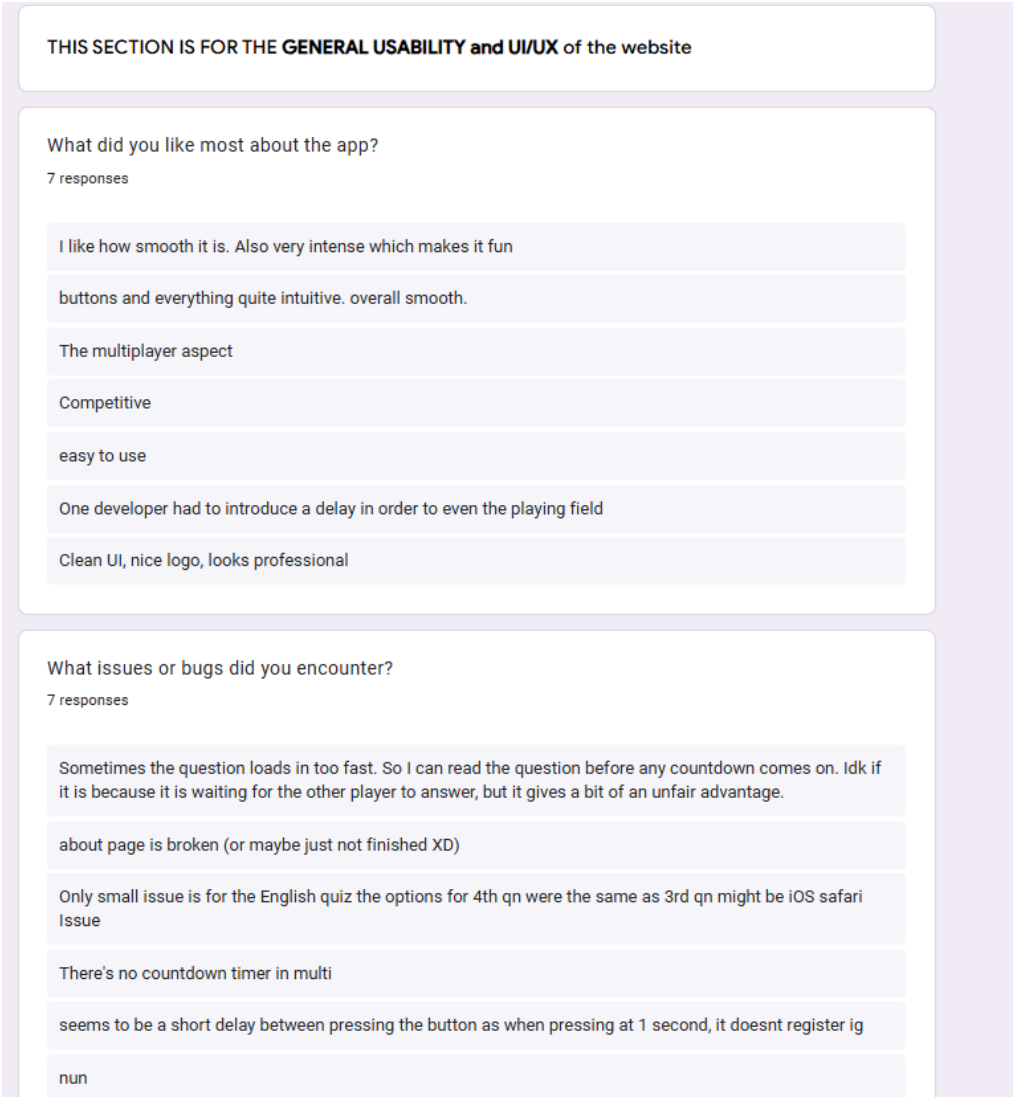
changed and why.

Security Measures

We keep user data safe by hashing passwords with bcrypt before saving them in the database, so plain-text passwords are never stored. Authentication uses secure HTTP-only cookies and JWT tokens, which help protect against attacks. Only non-sensitive information, like usernames and points, is shared in public routes such as the leaderboard. All user inputs are checked and errors are handled to prevent unauthorized access.

Quality Control

User Testing



THIS SECTION IS FOR THE **GENERAL USABILITY and UI/UX** of the website

What did you like most about the app?

7 responses

I like how smooth it is. Also very intense which makes it fun

buttons and everything quite intuitive. overall smooth.

The multiplayer aspect

Competitive

easy to use

One developer had to introduce a delay in order to even the playing field

Clean UI, nice logo, looks professional

What issues or bugs did you encounter?

7 responses

Sometimes the question loads in too fast. So I can read the question before any countdown comes on. Idk if it is because it is waiting for the other player to answer, but it gives a bit of an unfair advantage.

about page is broken (or maybe just not finished XD)

Only small issue is for the English quiz the options for 4th qn were the same as 3rd qn might be iOS safari Issue

There's no countdown timer in multi

seems to be a short delay between pressing the button as when pressing at 1 second, it doesnt register ig

nun

Fig 26. Open-Ended Constructive Feedback Segment

We asked real users to try out BrainBrawl by registering, logging in, taking quizzes, and checking the leaderboard. Their feedback showed us what worked well and what needed fixing. This helped us find bugs, make the app easier to use, and improve things like error messages. (The link to the feedback form can be found in the appendix.)

PostmanTesting

We used Postman to manually test all backend API endpoints during development. With Postman, we could send HTTP requests to routes to make sure each one worked as expected. For example, we tested user registration by sending a POST request with a sample username, email, and password, and checked the response for success or error messages.

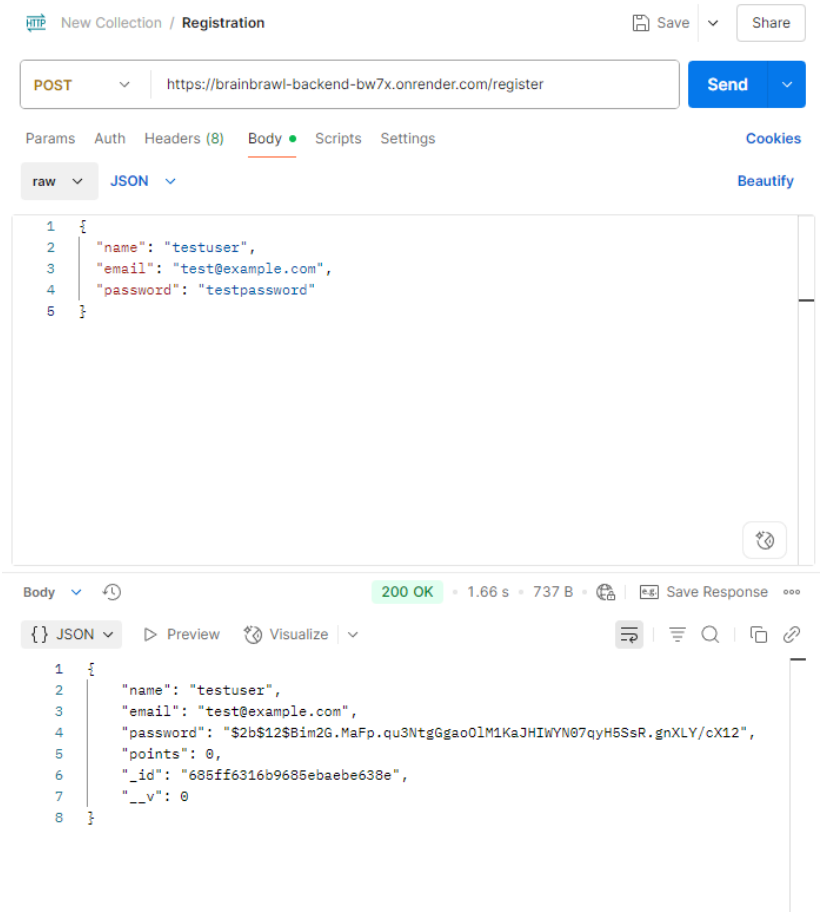


Fig 27. Registration Postman Test

We also tested error cases, like trying to register with an existing email

or logging in with a wrong password, to confirm that the backend handled errors properly.

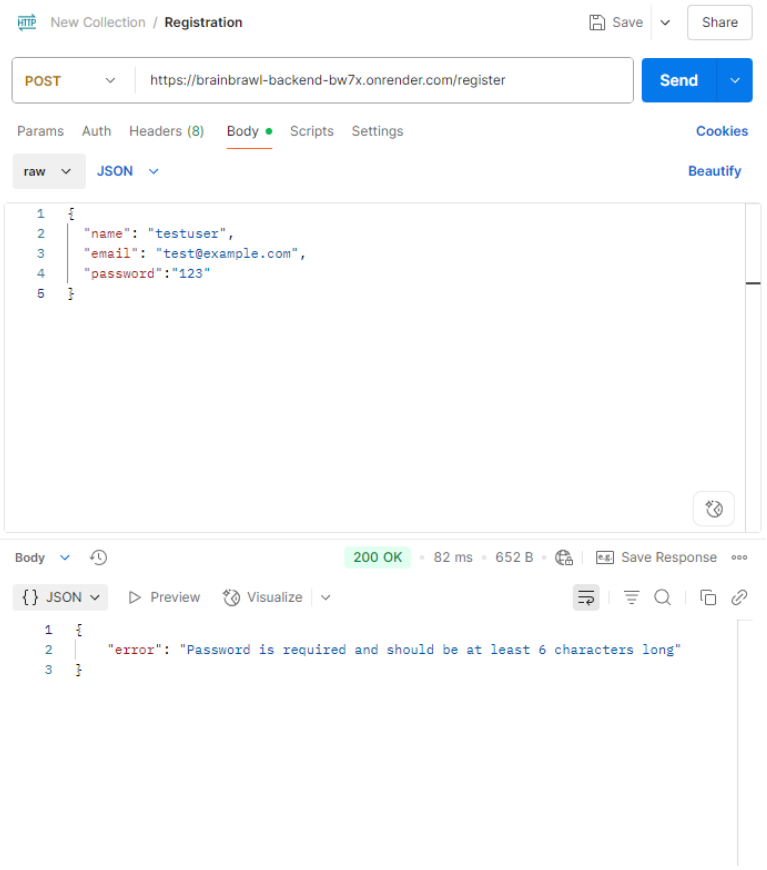


Fig 28. Postman Registration Weak Password Test

This process helped us quickly find and fix bugs, and ensured that our API was reliable before connecting it to the frontend.

End-to-end (E2E) Testing

We used Cypress on a separate testing branch to perform end-to-end (E2E) testing on key user flows in the frontend. This allowed us to simulate how a real user would interact with the application, from loading pages to making purchases in the shop.

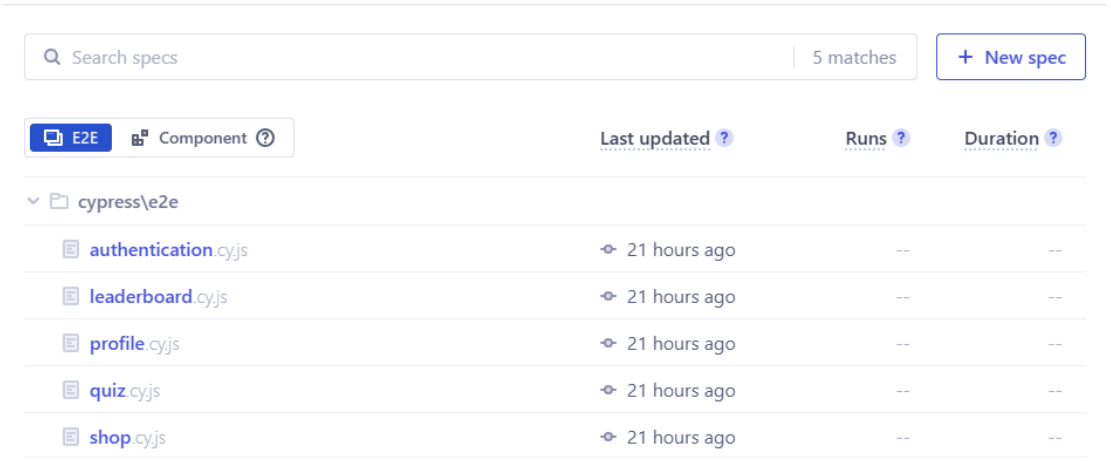


Fig 29. List of Cypress Key Web-App Functionality Tests

For example, we tested the Cosmetic Shop feature by simulating a user visiting the shop, checking their coin balance, and redeeming an item. Cypress verified whether the correct buttons were enabled or disabled based on the user's coin amount and ownership status.

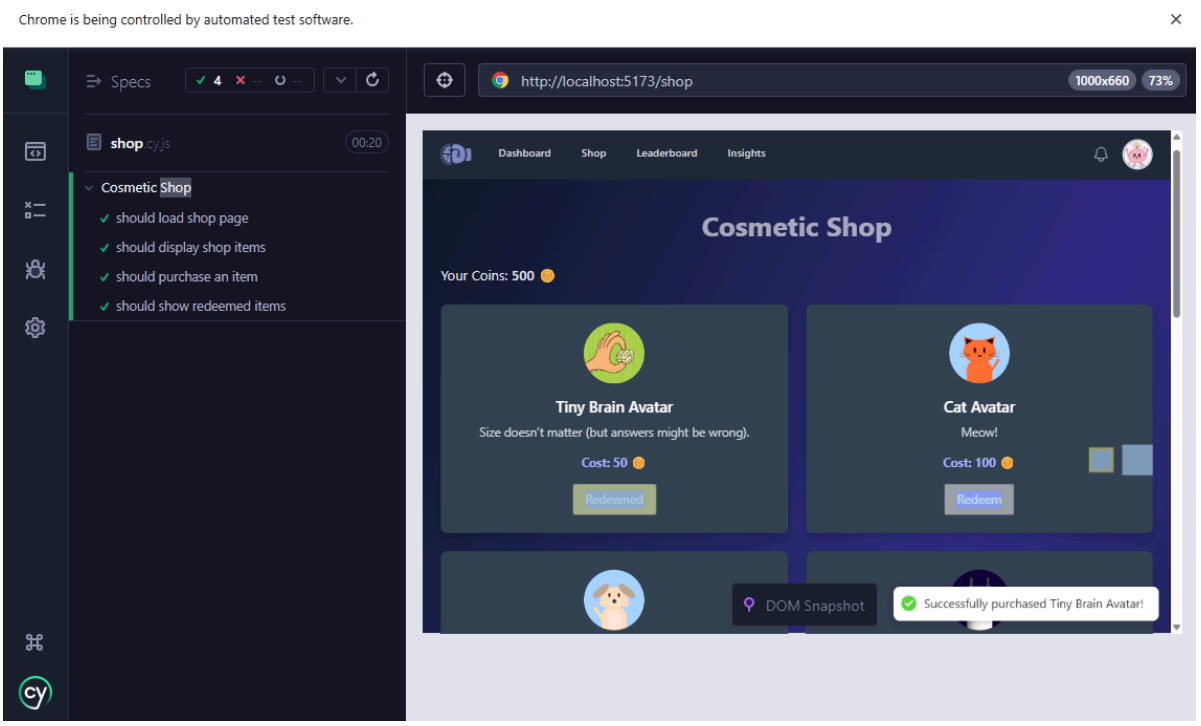


Fig 30. Cypress Shop Functionality Test

Timeline and Development Plan

S/N	Task Title	Status	Description	In-Charge	Date Range
1	Ideation	Completed	- Brainstormed quiz app concept - Researched QuizUp, Kahoot, Duolingo - Designed Poster and Video - Sketched wireframes using Draw.io/Canva	Jia Wei Kuan Yi	15 May - 18 May
Liftoff					
2	Main Screens	Completed	- Do up Home, Login and Registration Page	Kuan Yi	28 May
3	User Account Authentication	Completed	- Use Bcrypt for hashing, JWT for token authentication and connect to MongoDB	Jia Wei	28 May
4	Dashboard Page	Completed	- Design static content accessible only when users are logged in	Jia Wei	28 May - 29 May
5	Navigation Bars	Completed	- Add navigation bars to both home and dashboard pages	Kuan Yi	28 May - 29 May
6	Logout Functionality	Completed	- Implement logout logic (clearing of JWT tokens)	Jia Wei	30 May
7	Basic Quiz Functionality	Completed	- Design the basic quiz mode as a core functionality	Kuan Yi	30 May
Milestone 1 — Ideation					
8	Not Found Page	Completed	- Add an error 404 page for pages not yet implemented or does not exist	Jia Wei	10 Jun

9	Countdown Functionality	Completed	- Countdown clock for the quiz game modes	Jia Wei	10 Jun
10	Multiplayer Game Mode	Completed	- Create the 1v1 game mode using Socket.io to allow interactivity between 2 players	Jia Wei Kuan Yi	13 Jun - 25 Jun
11	Quiz data transfer to database	Completed	- Transfer the question/answer bank into MongoDB	Kuan Yi	25 Jun
12	Leaderboard	Completed	- Display ranking of all players based on points earned from game modes	Jia Wei	26 Jun
13	Prototype Deployment	Completed	- Deploy Frontend on Vercel and Backend on Render	Jia Wei	27 Jun
Milestone 2 — Prototype					
14	Power Up System	Completed	- Add an ability mechanism in the 1v1 quiz game mode to make it more interesting	Jia Wei	30 Jun - 4 Jul
16	Quiz Variety	Completed	- Add categories to both singleplayer and multiplayer quizzes	Kuan Yi	30 Jun - 4 Jul
17	XP System and Level Progression	Completed	- Integrate XP logic with quiz	Jia Wei	7 Jul - 11 jul
18	Custom Avatars and Profiles	Completed	- Create Profile and Settings page - Design custom avatars for users to equip	Kuan Yi Jia Wei	7 jul - 11 jul
19	Learning Insights	Completed	- Collect and analyse user data and provide personalised feedback	Jia Wei	14 jul - 18 jul
20	Cosmetic Shop	Completed	- Create in-game currency to be earned from quizzes	Kuan Yi	14 jul - 18 jul

			- Currency can be used to purchase items from the shop		
21	Refinement, Testing and Debugging	Completed	- Refine user interface - Test the system via user testing and automation extensively - Fix any bugs or issues	Jia Wei Kuan Yi	21 jul - 25 jul
Milestone 3 — Extension					

Challenges Faced

CORS difficulties

CORS issues caused a lot of unexpected bugs and made deployment especially frustrating. Even after setting what seemed like the correct configuration, requests would sometimes fail silently or cookies wouldn't be sent, breaking authentication and user sessions. The website would work perfectly on localhost, but as soon as it was deployed, things like login or fetching user data would stop working because of subtle differences in domain names (e.g. adding a "/" at the back of the Vercel url").It felt like every fix introduced a new problem, and getting both the frontend and backend to communicate smoothly took a lot of trial and error. This made the deployment process more stressful than expected.

Appendix

Work Log

- Refer to spreadsheet: Orbital Project Log.xlsx

User Testing Feedback Form

- Link to form: https://forms.gle/fPtdtggTptGmpj6g6\