Project Documentation
CFG Degree - Fullstack 1, Group 3

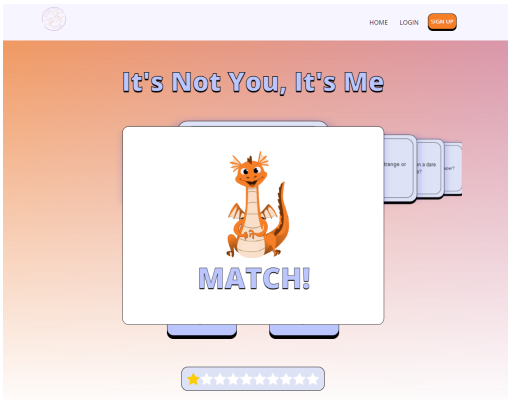
INTRODUCTION

For our final project, we have created a multiplayer game called “It’s Not You, It’s Me!”. We chose this project as it allowed us to demonstrate the skills we learned on the CFG degree, while pushing us beyond our starting skill levels. The aim of the project was to create a fully functioning React application, which satisfies a demand in the market, is scalable, and is cohesive throughout. We aimed to incorporate a connected database, backend server, multiplayer functionality, the use of APIs, strong design principles and testing. With consistent communication and a clear plan, we were able to complete our project within the timeframe and satisfy all of our aims. Flexibility was occasionally necessary, but thanks to detailed planning, our final product is similar to our original concept.

This report sets out the steps we took to bring our idea to fruition. The report explains the background to the game followed by details of the key features and flow of the game. It shares the reasoning behind the design choices and explains our roles within the team, the tools we used, the development process we followed and the testing of the final product.

BACKGROUND

Our application is an online multiplayer game that connects two players on different devices using WebSocket functionality, allowing them to complete a fun interactive quiz together, wherever they are. We have named the game “It’s Not You, It’s Me”. The concept of the game involves two players being asked questions, to which the answer is one of the players - e.g. “Who is more likely to be late for work?” or “Who has the better fashion sense?”. The players must then select their answers, and once both have responded, it is revealed to the players whether they have chosen the same or different answers.



Example of the ‘Match’ and ‘No Match’ pop ups that appear based on the users’ answers

Our game has many possibilities for expansion. For example, if we were to add a chat function it would allow the players to communicate on the same platform whilst they were playing the game. We also believe the game has scalability, as extra question ‘packs’ could be added, allowing users to pick a set of questions that are particularly suitable for their relationship, e.g. a pack specifically for families, or a pack for date night.

We believe our application satisfies user demand for two reasons. Firstly, more people are now looking to connect with friends and families virtually. Secondly, we believe our game stands apart from other similar online quiz games, as the only knowledge it requires is of each other. The personal style of the game fosters conversations and, as a result, brings people closer together.

SPECIFICATIONS

Our application consists of three key features, which are outlined below.

User Authentication:
The authentication process allows players to create an account using their details and a chosen username and password. These credentials are stored within our MySQL database, prioritising normalisation and upholding data integrity.

Dynamic Two-Player Interaction:
Leveraging the capabilities of socket.io, a JavaScript library for real-time communication, our application bridges two players across different web browsers. Socket.io orchestrates real-time data exchange, ensuring synchronised interactions and uninterrupted gameplay.

Interactive Quiz Experience:
Central to our application is the dynamic quiz feature. Players respond to questions sourced from our MySQL database via API calls. This approach ensures a constantly evolving pool of questions, guaranteeing a unique set each time.

Game flow:
Our game follows a clear path, allowing for ease of use. We developed this flow through the use of wireframing - a visual guide that represents the skeletal framework of our application. We created a wireframe, which was then adapted into the final draft after we used it to simplify the flow of the game. The flow of the game is as follows:

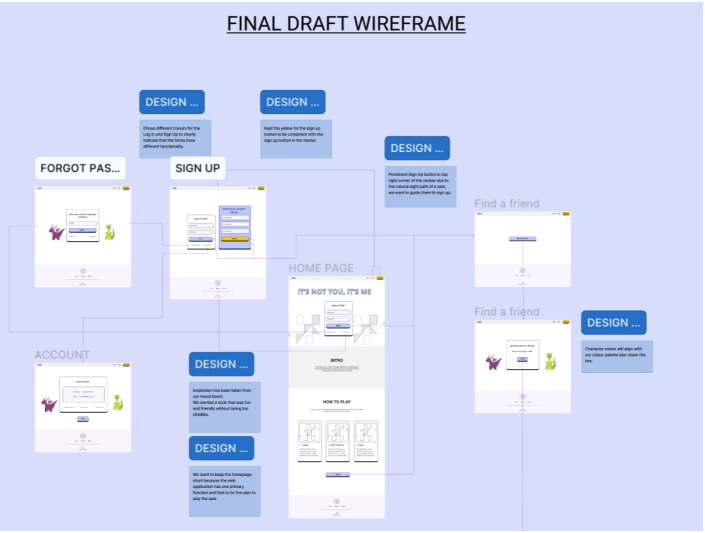
1. Account creation - the player starts by creating an account. The information is stored securely in the database.Login and authentication - the players logs into their account

2. Partner connection - once logged in, the player can select to ‘Start Game’ which then provides them a link to copy and share with a friend

3. Room Assignment - once the second player follows the link, the two players are placed within a new socket.io ‘room’.

4. Quiz - the questions are shown on a carousel one by one.

5. Answering - for each question, the players select between ‘Player 1’ or ‘Player 2’ buttons.



Sub-section of the final wireframe

6. Outcomes - after each question, the players are shown whether their choices were a match or not.

7. Quiz completion - the quiz ends after ten questions with a ‘Show results’ button.

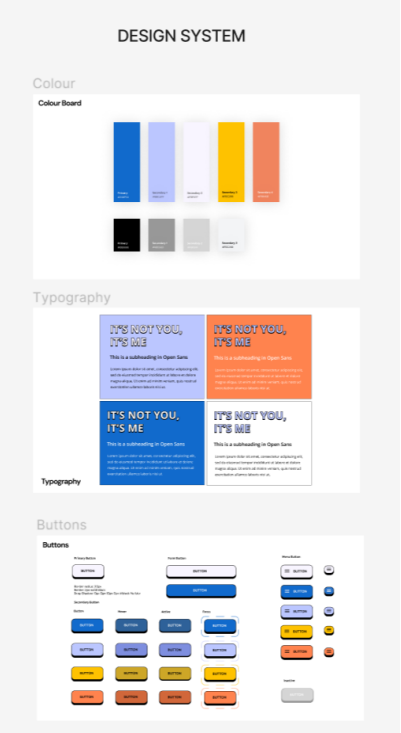
8. Scores - the players are shown their final score. One point is earned for each matched answer.

9. Back home - the players are shown a ‘Back to Homepage’ button.

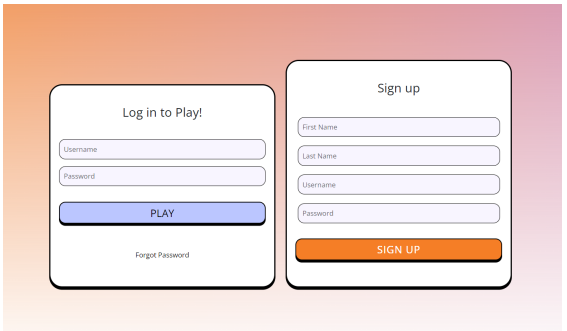
DESIGN

The game’s design is deliberately inclusive, employing a gender-neutral colour palette and icons. Every element features a signature border design, creating a harmonious visual identity that ties the interface together.

Our design philosophy is centred around clarity and user-friendliness, ensuring that the game's mechanics are conveyed concisely. The navigation guides users through the experience, with intuitive cues and pop-ups enhancing the interaction. The content is strategically placed to facilitate natural reading habits, and the quiz questions are presented in an eye-catching carousel feature.



Sub-section of design system



Homepage and Sign up pages of the application, demonstrating the colour palette, icons, and consistent styling of components

IMPLEMENTATION

The first stage in the process was agreeing upon an idea for the project. All of the team were enthusiastic about creating a game, so this was agreed upon quickly. The team was proactive in laying strong foundations. A kickoff meeting was held to understand the different strengths, weaknesses and goals of the team. Each individual also shared their schedule and personal time

constraints, so the workload could be fairly divided and managed. Eleri created a Discord channel and Megan set up recurring twice-weekly Zoom calls. These were held throughout the duration of the project, in addition to extra meetings that were held during class, or on an ad hoc basis. Communication was very strong in the team throughout the project.

Assigning of roles
After the first project meeting, it was agreed each member would be the lead for different areas of the project. This allowed clear points of contact for any issues.

Eleri - GitHub Lead, React Support
Kudzai - Database Lead
Maddy - React Lead
Megan - Project Manager, Design Lead
Niki - Task Management Lead

Megan set up a Google Spreadsheet, which the team populated with an overview, timeline, tasks, meeting notes and other useful information. Eleri set up a Google Drive for important files, assets and recordings of the weekly meetings. During these early meetings, we defined the key features of our application, and everyone was assigned jobs. These lists expanded as tasks were completed:

Eleri - set up Github repository, including a practice repository and homepage and support database construction
Kudzai - prepare the database tables with player information
Maddy - set up the initial project files and structure, and begin work on the gameplay
Megan - design the wireframe and moodboard, and set up the API for the questions
Niki - set up Jira and user authentication pages

Tools and libraries
It was initially agreed that React would be used for the frontend and Flask for the backend. However, after Node was demonstrated during classes, we switched the backend to Node (Express).

Main tools:
- Jira for project management
- Github for managing code and simultaneous working
- VSCode for writing code on our local machines
- Figma for design schema and wireframe

Libraries:
- React - building our interface
- Express - creating our server
- Axios - to allow us to make HTTP requests from the browser or Node
- Bootstrap - library of pre-designed components and styles for our webpage
- Cors - allows our client and server to communicate
- MySql12 - allows our server to use information from our SQL database
- Nodemon - updates code as you make changes to see updates in real-time
- Socket.io - allows real-time communication between client and server
- Bcrypt - for password hashing
- Redux - for managing the state of our App
- Jest - for testing our app
- Dotenv - for storing private information

EXECUTION

We developed our own agile workflow that worked best for our project and also accommodated the team's lives outside of the course.

Product Backlog- we used Jira to list our features, user stories and tasks that needed to be implemented. This was built in the first week so everyone had a clear understanding of the expectations and jobs assigned to them.

Sprint Planning- we spent the first week planning the project. This included setting up chats, meetings, Google Docs, Google Drive, Jira project, Github and the main file structure and setup for the project. We did not give specific deadlines for each task, but an overall deadline of 2 weeks for the writing of code. We assigned everyone on the team at least one major job to do first and all smaller jobs were left until last and completed in order of priority. We allowed the last week for merging branches, troubleshooting, testing, final edits, finesse elements and documentation.

Sprint- All major components were completed mid-way through the final week. The team constantly communicated details of which components were working well and which were not.

Daily Standup- we did not do dailies because of the team's external commitments, instead checking in daily in class and on our Discord channel. We had team meetings on Sundays and Wednesdays in addition to meetings in class and ad hoc.

Development- this was continuous throughout the project. We were constantly working together and sharing work on Github to make sure different components were working together. Once a branch was ready to be merged, a PR would be made and Eleri as GitHub lead would then check for conflicts. She was solely responsible for merging branches to the main, thus minimising confusion with merges and ensuring conflicts were resolved promptly.

Reviewing- we did reviews and demonstrations in our weekly team calls when components were ready. In the final week, Eleri, Maddy and Meg met in person to do the final review of the main bulk of the application. A checklist and timeline for the last few days before project delivery was shared amongst the team.

Adaptation- we adapted regularly on the project where needed. When beginning the project, we tried to make all major decisions about the game, but as we worked, we often came up with better solutions. In our initial wireframe, we had our game going from homepage → login→ find friend → connect with friend → choose character → play. We realised this was too long for us to complete in the timeframe, so it became homepage → login → link to friend → play instead. We initially thought we would use Flask and React, but in the second week we realised that it would be easier to use Node and React, so we adapted the project accordingly. Originally, we had planned on storing the questions in an API, but we changed this to storing them on our database instead, as this is best practice. We also added additional components to the homepage to make it more visually engaging.

Incremental Development- because we needed each other's code to check if our own components were working as expected, we shared all components incrementally.

Embracing Change- as outlined in the 'Adaptation' section, the team was readily able to embrace change for the good of the project.

TESTING

It was imperative for us to outline clear testing objectives. We wanted to integrate unit testing into the project to guarantee optimal performance. A significant goal was ensuring the game's cross-device compatibility post-deployment.

We inserted tests within our code elements and concentrated on crafting tests for individual React components to ensure they rendered and operated correctly. We also included tests for database functions. Additionally, we integrated tests into our Socket.io implementation to catch and correct any errors that might arise during gameplay.

Throughout the project, the team ensured that all components functioned as intended. In our meetings, we demonstrated the user experience. For example, when crafting the login and registration features, we thoroughly tested them to confirm new members could be added and then logged in. The team also regularly inserted console.log statements to track the flow of the application.

To safeguard our project's integrity, team members were advised to create separate branches for new components, utilising GitHub for version control. We adopted the saying: "Protect the Main". This approach encouraged the team to pull fresh branches from the main when tackling new tasks.

During user testing, a range of people from ages 24-70 tried the game and provided feedback to inform future developments, helping to refine and evolve the project.

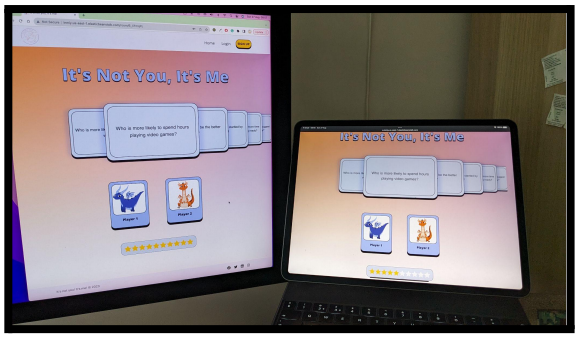
The feedback showed:

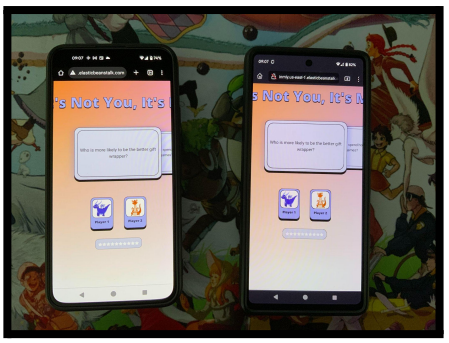
- Users enjoyed the changing questions each game session.
- The instructions were clear and easy to understand.
- Users found the game enjoyable, sparking fun discussions.
- The design was described as modern with engaging colours and characters.
- Buttons made navigation simple, guiding users through each step.
- Many liked that the game could be played on various devices, one user mentioning the game offered "a new way to connect with loved ones."

Areas to improve:

- The 'Finding a Friend' feature could be easier, especially for mobile users. A universal share button was suggested for easier game-sharing.
- While the logo's design was praised, it could be more visible with bolder lines.

One final goal was deploying the project. We used AWS Beanstalk, and the outcome was positive. Owing to tight timescales, the project was deployed with some unfinished elements leading to a few bugs. Nonetheless, we achieved cross-device functionality for the game which helped further testing.





EVALUATION

Looking back at our development and completion of the project, we were able to identify key achievements, challenges, and changes that were made during the process.

Achievements

- Team-working - communication, time management and decision-making
- Use of new software - Socket.io, Jira, building APIs from scratch
- Learning and implementing React and Node
- Sticking to deadlines
- Reaching submission with a fully functioning web application

Challenges

- Merging code on GitHub and ensuring there were no conflicts
- Ensuring the randomised quiz questions were the same set for both players
- User authentication- Login and Registration
- Connecting to the database
- Using a single server
- Ensuring the correct, up-to-date result is displayed on the results page
- Balancing other work with project work
- Utilising Bootstrap across all components of the project

Changes

- Changed questions that were once stored in the server and fetched by API, to be stored in the database.
- Changed the results page to be a pop-up, made visible with the use of state in React
- Added a new About Us section with avatars
- Linking by username was discarded and we used a link to be copied and sent to a friend instead.

There were other changes we discussed throughout the project, but we agreed to set them aside to ensure we reached a fully functioning application within the deadline. For example, in future we would ensure both players had to log in, so that the buttons within the game could show their usernames rather than ‘Player 1’ and ‘Player 2’.

CONCLUSION

Our aim was to create a React application, which satisfied a demand in the market, demonstrated the skills we learned during the course, and was cohesive throughout. Our application is fully functioning and includes all of the features we set out to incorporate: a React application with Node (Express) on the backend, user authentication, WebSocket functionality, a connected database storing key information and calls to an API. The flow of the game is intuitive and the design is clean and consistent throughout. As well as the research we undertook and the hours we put into the development of this application, the consistent and effective communication between members of the team was paramount to the project’s success.