Fullstack-6 Project Report

Introduction

Triviago brings the lively spirit of a pub quiz into a digital format, offering users a fun and interactive quiz experience. Players can test their general knowledge or choose from 20 specialist subjects, with a leaderboard bringing the competitive edge. During this project, considerations have been made to create a user-friendly, entertaining, and accessible React app.

This report outlines the inspiration for our project and how it is valuable for our target audience, followed by an overview of the design process. The implementation section deep-dives into the steps undertaken to build the app, including our development approach and execution, highlighting challenges we faced and successes we accomplished as a team. The final section provides insight on our testing and evaluation process. The report is concluded by what we learned and how we would further develop the app in future.

Background

Our idea was born from our team members' love of pub quizzes. The vision for the app is to provide trivia enthusiasts and pub quizzers alike a fun and accessible way to play from home or on the go. Our target audience includes individuals who enjoy testing their knowledge, groups looking for an engaging activity, and educators seeking a fun way to conduct quizzes.

Whilst researching possibilities, we encountered the Open Trivia Database, a free to use quiz question database, which allowed us to bring our vision to life. Our aim was to use this API to provide both general knowledge questions and trivia related to specific categories. We wanted the app to feel fun, like a game, while still providing interesting and broad quizzing opportunities.

In order to adequately consider user experience (UX), we agreed that before beginning the build we would create wireframes and map out user journeys to ensure the structure of the app was intuitive, clear and easy to navigate. To help with this, each stage of the application was designed to have only one or two calls to action, so that it would be clear to the user on how to use the app. We also made accessibility considerations, for example for users with screen readers, and to consider aspects such as contrast so that text would be legible.

It was also acknowledged that in order to create a consistent, cohesive user interface (UI), we needed to produce a refined and clear design, which all team members could refer to throughout the development process. As such, we spent time during the planning phase to agree exactly what we were aiming for in terms of layout, aesthetics, colour scheme and typography. This allowed for a coherent design language throughout the application, and also meant that we could maximise the use of reusable components in React to efficiently create building blocks that will be used in all pages.

Specifics and Design

Requirements

When planning and designing our app, we decided to outline a minimum viable product to aim for before adding any additional features:

Minimum viable product (MVP)

- Home page with welcome message, instructions, and a ‘generate quiz’ button
- Quiz page with multiple choice questions and checkboxes for answers
    - Using hard-coded questions/answers at this stage to ensure correct quiz logic
- User's score is calculated and displayed attend of quiz
- User-friendly and accessible interface
- Unit testing of individual components is implemented

Second iteration

In the second iteration of the app, we intended to connect to the third party API in order to fetch questions and answers. We viewed this as a ‘should have’ requirement. We planned to fetch ten general knowledge questions as a standard. We also planned to implement the scoreboard in this iteration, which connects to a database of high scores.

Further development

Beyond the second iteration of the app, we brainstormed further ‘nice to haves’ which could be picked up if time allowed, these included:

- Option for users to choose different categories of questions (with responsive CSS)
- Option for users to choose the number of questions in the quiz
- Functionality to share quiz sessions with friends (research detailed in this document)
- Login and authentication
- Timed questions with visual timer
- Mobile friendly design

Design and architecture

Themes and ideas were discussed among the group, and it was agreed that we wanted the app to be fun and engaging, using bold, bright colours while still thinking about accessibility for the user. We visualised our ideas in a Figma design.

Arcades/gaming was the inspiration for our design. Darker colours were used to simulate an arcade atmosphere; although we made a conscious decision to avoid a true black for both depth and accessibility reasons. The navbar is designed with neon navigation links to further reflect the feel of an arcade. Headings and buttons were also designed in the ‘classic’ arcade game font, including blinking arrows.

In general, the headings of each page are at the top and the navigation buttons are at the bottom right of the screen, giving the pages a consistent layout throughout for an intuitive UI. We ensured not to overload each page, limiting the calls to action on each page, whilst still engaging the user through the design.

The page sequence follows a simple journey from the landing page > instructions > quiz questions > scoreboard. Screenshots from our initial Figma design are shown in Figure 1.

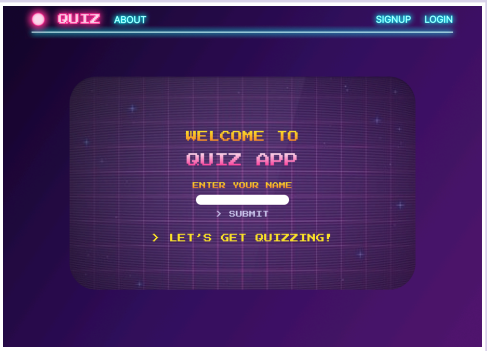
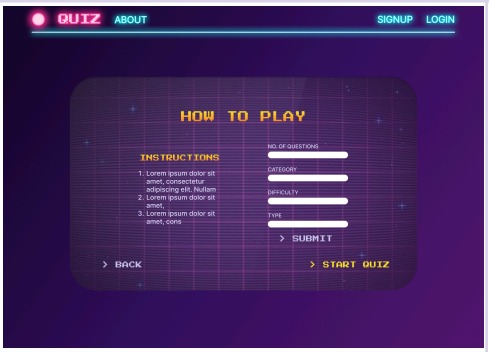
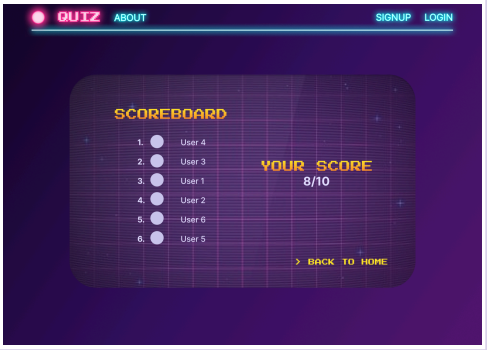
	**Landing page:** Navbar with links to Home, log-in, sign-up and 'about us' pages. Input box for player to enter name (linked to database). 'Let's get quizzing' button to lead to the next page.
	**Instructions page:** Step by step Instructions on how the quiz works. Form for users to choose number, category, difficulty and type of questions. Start quiz button to begin the quiz.
	**Quiz page:** The quiz page then displays one question at a time alongside multiple choice answer buttons which the user can select. Progress bar (bottom-left) visualises how many questions have been answered.Next button in the corner to progress to the next question.
	**Scoreboard:** Displays the top scores fetched from the database alongside the users score. Button in the right-hand corner to navigate back to the homepage where the user can start a new quiz.

Figure 1: Design wireframes and page sequence

Implementation and execution

Development approach

At the beginning of the project, each team member completed a SWOT analysis to highlight their strengths, weaknesses and interests, helping us to distribute tasks efficiently. Table 1 provides a general overview of team member responsibilities.

	Jacquelyn	Paola	Phoebe	Rania	Rayah	Soo-Jin	Olivia
Standups & code reviews	X	X	X	X	X	X	X
Ideation and research	X	X	X	X	X	X	X
Design & wireframes						X	
Component/page creation	X	X	X	X	X	X	X
Styling (CSS)	X	X	X	X	X	X	X
Quiz logic	X					X	
3rd party API connection			X				X
Database implementation	X	X					X
Testing and evaluation	X		X				
README compilation	X						
Project document	X					X	X
Project presentation		X					

Table 1: Division of responsibilities across the team

We opted for an agile approach using Kanban principles. Daily standups were conducted after lessons for planning and reviews. A Jira Kanban board was used to manage stories as ‘To do’, ‘In progress’ and ‘Done’. We frequently reviewed the board as a team and agreed on which members would pick up particular stories. We decided on this flexible approach as opposed to assigning scrum roles due to the fairly short length of the project and the ease at which we could pick up stories and continuously develop.

We decided on a feature branching way of working, each team member worked on a particular feature at a time, for example, ‘create heading component’. Each development branch is named consistently in the format “feature/jiraCode-yourName-jiraTitle” to link the branches to the board. Branch protection rules were implemented on the staging branch (two reviewers required per pull request and all conversations resolved) to prevent accidental merging. We outlined our ways of working in a shared document on Slack.

Implementation process

Our work was broadly split in to 1-2 week sprints, goals were set for what we aimed to achieve during each:

- Week 1&2: Planning, designing
- Week 3&4: Building MVP
- Week 5: Testing & implementing nice to haves

Our iterative approach to development allowed us to create a working app, then gradually implement new features, refactor and improve functionality. Each member continuously

contributed to the code base, communicated via Slack and provided code demonstrations where necessary. Figure 2 shows our step-by-step process of building the app.

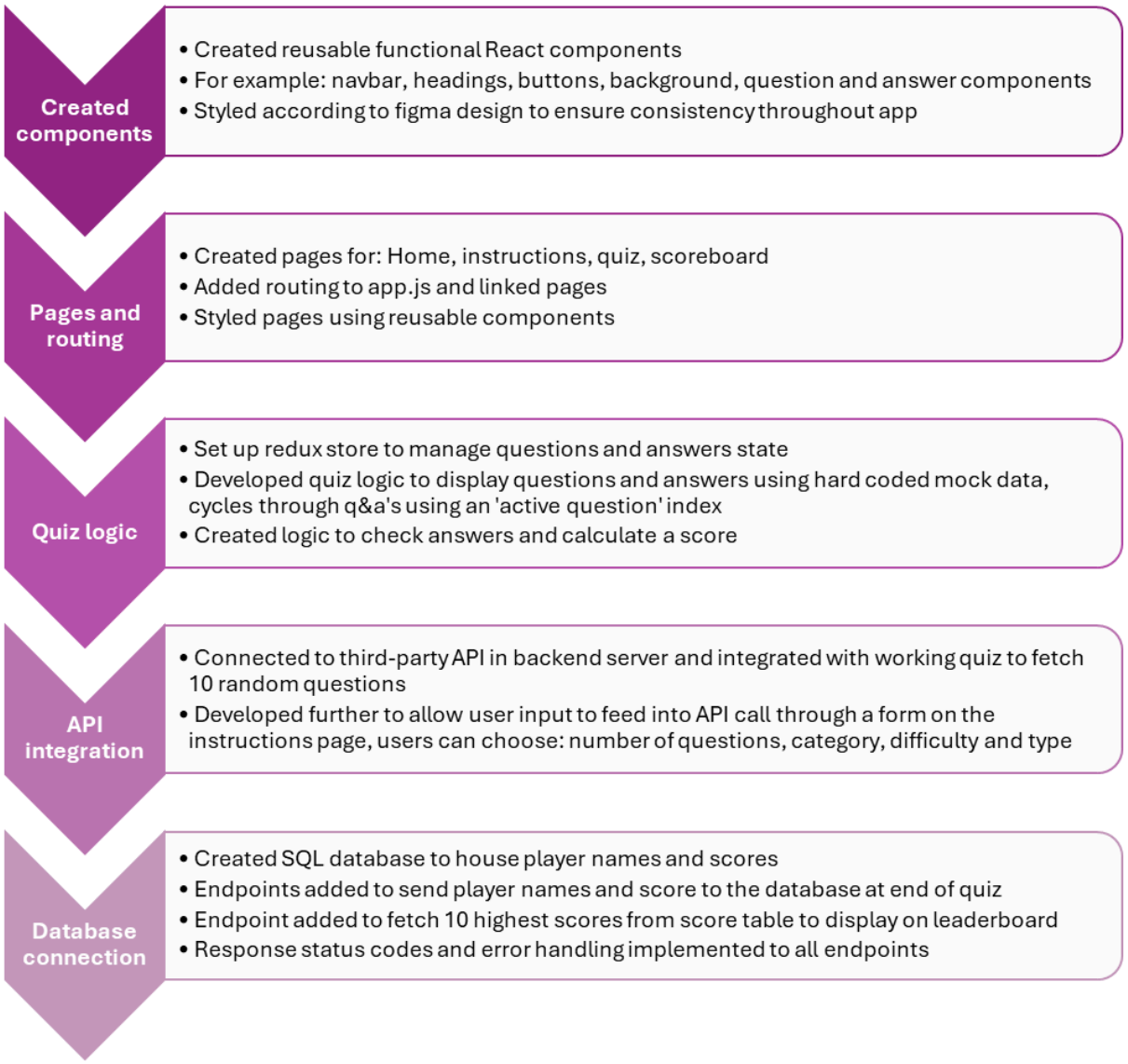


Figure 2: Overview of implementation process

Some notable achievements we are proud of include creating a design layout based on agreed functionality to create a concise application from the outset, this ensured no superfluous pages or components were created. The design, alongside the work put in at the beginning of the build to create small, reusable components, made creating a cohesive and user-friendly app more successful. We are really proud that we could execute the design set out during the planning phase.

We were also very pleased to implement a redux store to manage the state of our quiz questions and answer options. Centralising state management made it much easier to share state across different components, having a ‘single source of truth’ and avoided having to pass props between multiple components (prop drilling) which may have become complex as questions and answers changed with every new API call.

We were also able to implement some of the ‘nice to haves’ outlined in our planning phase:

- Enabling the user to create a bespoke quiz by feeding user input into the API call
- Connecting to the database to store player names and scores, and fetching the appropriate data to the frontend for the leaderboard

We faced some challenges along the way, for example negotiating and managing each team member’s schedule and time commitments. We dealt with this through regular communication via slack and daily standups, which helped us work well as a team. We were also new to collaborating on a shared git repository, dealing with multiple development branches and merge conflicts. Our organised approach to branch naming linked to the Kanban board, as well as descriptive pull requests, helped to mitigate any issues. Thirdly, assessing which user stories were essential and prioritising tasks was something we had to be mindful of. Our team was ambitious to implement exciting functionality on this app, but we made sure to achieve the minimum viable product first before picking up stories related to additional features.

As a team, we made some key decisions during implementation, one of which was deciding not to attempt log-in and authentication; ultimately we agreed this was not an *essential* feature of our app and would have been very time-intensive. However, this is something we would like to add to the app in future. Secondly, we decided to connect to the third-party API on the backend server as opposed to through the react app itself. Partially to gain experience of programming like this as it is more common in real-world applications, and also as it allowed us to define the shape of data sent to the FE from the API call. For example, before sending the data to the FE, we combined and shuffled the answers into an ‘answer options’ array (hiding the correct answer), gave each question an ID, and decoded html entities, making the data more manageable.

Tools and libraries leveraged

- Jira for Kanban board management
- Slack for regular communication and sharing links to pull requests
- Zoom for daily standups
- Prettier & ESLINT - for consistent code formatting
- Axios - for third-party API calls
- React modal - used for modals on our about us page
- Redux - state management
- React Router
- He library - for HTML entity decoding
- Mysql2 and express.js - connecting to the database
- Jest and React Testing Library - testing

Testing and Evaluation

Automated testing

We implemented unit tests to cover frontend components using React Testing Library (RSL) to ensure components rendered as and when required. More complex tests were implemented for components with functionality, for example:

- Answer buttons: used the fire event function to simulate a user clicking an answer button, the tests ensure the answer is rightly assigned as correct or incorrect.
- Progress bar: Jest mocking used to simulate the app mid-way through the quiz, tests check that the progress bar is the correct size depending on question number.

We used Jest framework to cover all backend server endpoints (requests to/from the database and between FE/BE), testing that each scenario returns the expected status code and request body. We implemented Jest mocking for requests involving the third-party API:

- For fetching questions from the API, Jest mocks the 'axios' library rather than making real requests, allowing us to solely test our code handling the different responses.
- When sending data from BE to FE, we mocked the third party API response data to focus on testing the action of sending data from BE to FE.

Manual functional testing

We tested the app manually to check all functionality (button clicks, form submissions, scoring) worked as intended. We used logging to ensure the correct data was sent between our back and front end. We noticed that a user could diverge from the intended user journey by navigating straight to the /quiz path, causing issues for functionality. We safeguarded against this by adding logic to redirect the user to the home page if certain criteria were unfulfilled.

A system limitation we noted during manual testing was that the third party API couldn't handle too many calls in quick succession, as such we implemented a timer on the submit button to prevent this.

We investigated our vulnerability to HTML and JS injection, we identified our name input field on the landing page where free text is submitted. React inherently protects against injection for the most part, however we implemented custom regex rules to further mitigate risk and ensure a valid name is inputted. We protected against SQL injection through parameterising queries in our BE.

Conclusion

We have successfully developed an interactive quiz app that allows users to learn and test their trivia knowledge in a fun and entertaining way. Our design echoes the feel of retro arcade games and users are able to create bespoke quizzes depending on their area of interest. Learnings from the project include understanding the benefit of an agile approach, the importance of regular communication, and the benefit of pair programming.

If we were to continue developing the app, implementing login and authentication and improving responsiveness would be our next steps. We would also like to develop functionality so groups could share quiz sessions, allowing friends to enjoy the app together or open the door to use in more educational settings.