University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Pair Programming Web Application

Naral Chalermchaikosol

March 22, 2024

Abstract

The current tools for coding interviews and collaborative sessions are often inadequate, lacking in real-time interaction and comprehensive content. Our web application addresses these gaps by offering real-time code collaboration, voice and audio call, an extensive question pool, and direct GitHub integration. Moreover, research says remote pair programming helps one develop their coding proficiency, further incentivising the use of the application (1). Enhanced by modern web technologies, it provides a seamless, interactive environment for interviewers, candidates, and collaborators.

User evaluations highlight the app's effectiveness in improving the coding interview process and peer programming experiences. Its innovative approach is poised to set a new standard in the field, and we're planning to extend its reach and impact through targeted blog posts and participation in relevant tech forums, sharing insights and gathering further community feedback.

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Signature: Naral Chalermchaikosol Date: 22 March 2024

Contents

1 Introduction

This chapter will introduce the Pair Programming Web Application project, examining the motivation and aim behind the development of the system. Further chapters in this paper will go on to detail specific areas of the project, including its background, design, implementation, and evaluation.

1.1 Motivation

The current landscape of coding platforms offers either a space for honing individual skills through challenges or a separate realm for collaborative development. However, a unified platform that adeptly serves both these purposes is conspicuously absent. While sites like LeetCode, HackerRank, and CodeSignal excel in individual technical assessments, and Replit shines in cooperative coding, there is a clear market void for an integrated environment that combines the rigor of a coding interview platform with the synergy of a collaborative development workspace. This gap hinders the seamless transition between individual practice and collaborative work, which is increasingly vital in a world where remote teamwork and technical competence are paramount. Recognising this deficiency and the educational potential of pair programming according to Hughes (1), our initiative is to merge these two critical aspects into one cohesive and interactive platform which aligns with findings from Hanks (2), who highlighted the benefits of pair programming in education, such as improved learning outcomes and software quality, enhancing the technical interview experience through live, collaborative problem-solving.

1.2 Aim

In response to this gap in the market, our aim is to launch a Pair Programming web application designed to serve dual purposes. It will provide a robust environment for technical interview preparation, equipped with a diverse set of problems in Data Structures and Algorithms (DSA) and Front-End technologies, while also offering an interactive platform for peer programming and project collaboration. This application will support various programming languages and frameworks and will feature GitHub integration for efficient code sharing and version control. Our goal is to create a versatile platform that enhances the technical interview experience through live, collaborative problem-solving, thereby equipping users with a tool that reflects the collaborative essence of modern software development.

1.3 Summary

This chapter outlined both the motivation and aim behind the development of the Pair Programming Web Application to make clear the overall objective of the system, along with how it aims to add benefit to the emergency management sector. The remainder of this paper will discuss how the motivation for software was developed into a working system, and how the aims were met. The paper is structured as follows:

- **Chapter 2** shows research related products to Pair Programming Web Application, identifying their strengths and weaknesses. This chapter also investigates the backend services available for use throughout the project.
- **Chapter 3** outlines the requirements analysis process, from creating user stories and scenarios to developing a complete set of functional and non-functional requirements.
- **Chapter 4** discusses the design process including the system architecture, tools, and technologies to make use of and constructing the user interface, relating each design decision back to the project requirements.
- **Chapter 5** reviews the implementation of Pair Programming Web Application and highlights the software engineering processes involved throughout the project.
- **Chapter 6** provides an evaluation of the system, including automated testing, performance evaluation, user evaluation and requirement validation.
- **Chapter 7** summarises this paper, examines any future work which could be carried out and provides an overall reflection of the project.

2 Background

This chapter will examine the background of the project, first by looking at several related products that have similar goals or functionality as that of Pair Programming Web Application. The chapter will then investigate the pre-existing services which form the foundations the system is built upon.

2.1 Related Products

A range of systems created for commercial purposes have been introduced with a similar goal as that of Pair Programming Web Application in supporting software engineering candidates show their coding and problem-solving skills. This section provides a review of several of these products, identifying any strengths or weaknesses and discussing how each has influenced the design and development of the new platform.

2.1.1 HackerRank

The site has established itself as a leading technical assessment and remote interview solution for hiring developers. It offers a vast array of challenges across various domains, enabling candidates to demonstrate their problem-solving skills. One notable feature is its code playground, which supports a multitude of programming languages, providing a versatile environment for coding tests. However, while HackerRank excels in individual assessments, its collaborative features for real-time coding are not the primary focus, presenting an opportunity for Pair Programming Web Application to enrich the collaborative aspect of coding interviews.

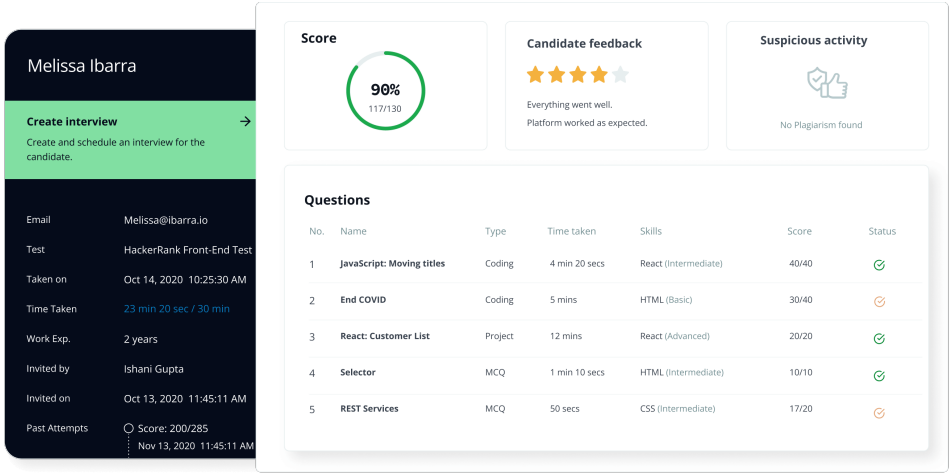


Figure 2.1: *A screenshot of HackerRank.com (3)*

2.1.2 LeetCode

The site is renowned for its extensive problem set for practicing algorithms and data structures interview questions. It has been instrumental in interview preparation for developers, with a focus on individual and learning. Despite its well-structured content and active community, LeetCode's platform is not inherently designed for synchronous collaborative coding sessions, an area where Pair Programming Web Application aims to shine by integrating real-time collaborative tools.

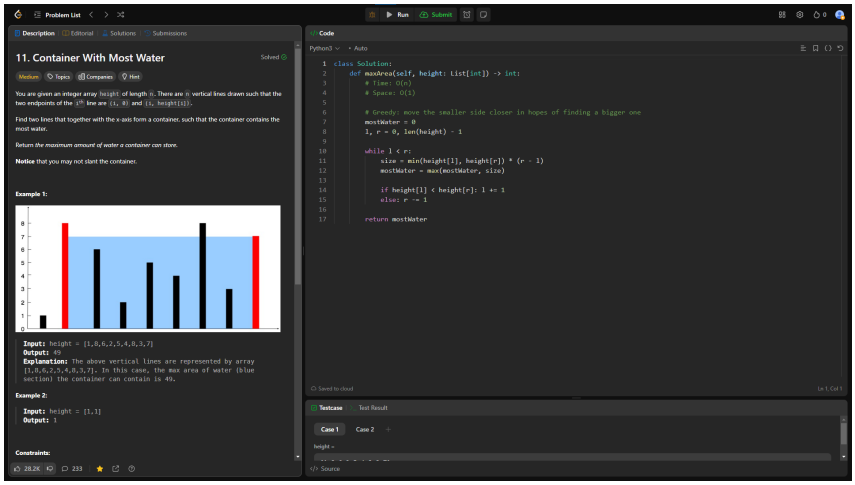


Figure 2.2: *A screenshot of Leetcode.com (4)*

2.1.3 CodeSignal

The site provides a platform that evaluates coding skills using automated, timed coding challenges. It is recognised for its standardised testing framework that helps in reducing bias during the hiring process. The platform's limitation lies in its automated scoring system, which may not capture the nuances of a candidate's thought process as effectively as a collaborative discussion could. Pair Programming Web Application intends to address this by allowing more interactive and nuanced review of coding solutions in real time.

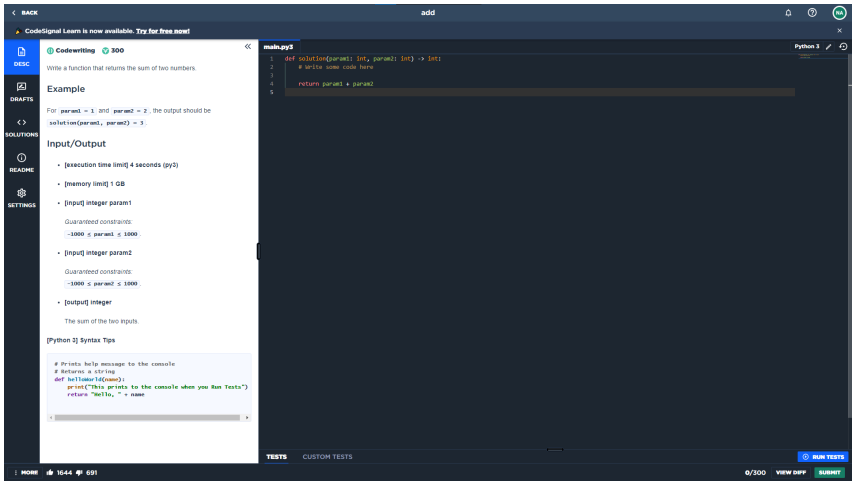


Figure 2.3: *A screenshot of CodeSignal.com (5)*

2.1.4 Replit

The site stands out for its collaborative coding environment, where developers can write, share, and run code within the browser — conducive to education and collaboration. However, its integration of audio and voice call into the technical interview process is not as robust as dedicated interview platforms. Pair Programming Web Application draws inspiration from Replit's collaborative nature while tailoring its features to meet the specific needs of coding interviews.

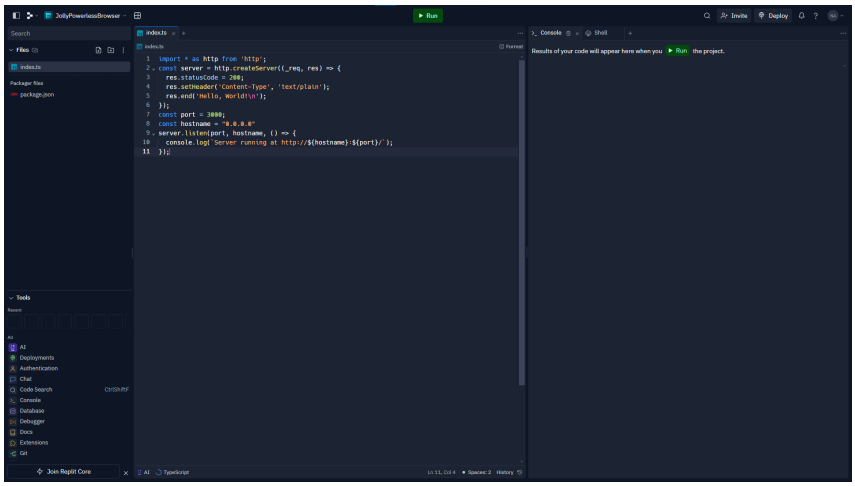


Figure 2.4: *A screenshot of Replit.com (6)*

2.1.5 Codility

The platform provides online coding assessments and tools for evaluating the programming skills of candidates, known for its real-time coding space and comprehensive collection of tasks aimed at effective skill testing. While Codility excels in candidate evaluation, it lacks features that support in-depth collaborative coding beyond interviews. The Pair Programming Web Application aims to address this, creating a more comprehensive solution for both interviewers and candidates.

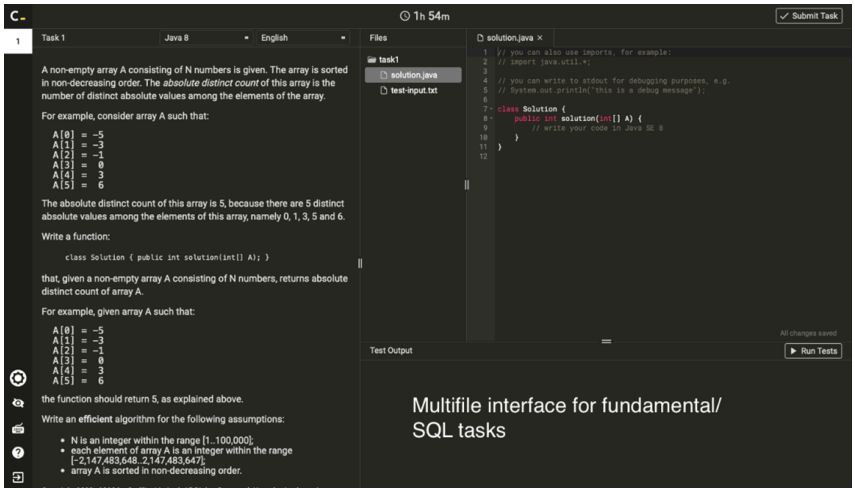


Figure 2.5: *A screenshot of Codility.com (7)*

2.2 Core Concepts

As discussed, Pair Programming Web Application's simple yet robust architecture will allow for seamless interaction between peers — designed to increase the range of capabilities of the system; with the greater goal of improving each other's coding proficiency as discussed by Hanks (2). This section introduces existing technology concepts which were used to craft the software. Their justification and implementation are discussed further down the report.

- **Real Time Collaboration & Communication**: the core functionality of the Pair Programming Web Application centers on real-time collaboration and communication. It ensures users can simultaneously interact with shared editors for code and notes, reflecting changes instantaneously between all parties. Additionally, the app provides real-time voice and audio communication, facilitating clear and immediate dialogue between users for a more interactive and connected coding experience.
- **Text Editors**: the web application allows for two types of programming: front-end and algorithmic, with the former involving in-demand frameworks like React, Vue, and Angular, while the latter involves general purpose languages like Python, C#, and TypeScript. There is also a dedicated note editor for peers to share formatted text (i.e., paragraphs, tables, lists).
- **Code Execution & Preview**: the system allows users to execute, and validate code in real-time, crucial for coding assessments and collaboration, ensuring the code runs as expected across different languages.
- **Authentication & Progress Management**: the platform enhances the user experience by allowing them to keep track of their progress — through uploading code to their own GitHub repositories. Authentication is required.
- **Storing Coding Questions**: to assess software engineers fairly and consistently, it's crucial to have a set of similar questions prepared. Users of the platform would be able to create questions (along with hints and solutions) freely
- **Storing Room Data**: it is essential for users to have the ability to store and retrieve session-specific information such as room configurations, participant roles, and session statuses. This functionality enables the platform to maintain the state of each room..

3 Analysis/Requirements

This section delves into both the functional and non-functional specifications of the project, detailing the approaches employed to derive these requirements. While these requirements were initially set at the start of the project, they were subsequently refined and expanded upon with the progression of the project and the incorporation of new services.

3.1 Example Scenario

Here is a user scenario crafted to illustrate how coders can utilise the platform

- **Jonathan**, a developer preparing for a technical interview, and **Joseph** — a coding interviewer seeking to assess candidates effectively — need a platform that supports a wide array of modern mainstream programming languages and frameworks. This platform should allow for the creation and management of detailed coding assessments, including questions, hints, and solutions, catering to both algorithmic and front-end development challenges. During live coding interviews, it's crucial that they can write, execute, and validate their code in real-time, showcasing their problem-solving skills and coding proficiency. Additionally, the platform should offer real-time video calling to facilitate seamless communication between them, enabling a dynamic and interactive interview process. Integration with GitHub for code submission and authentication enhances the user experience, allowing for a seamless transition between coding exercises and progress management.

3.2 User Stories

Following the conception of the user scenario described earlier, different use cases were decided upon, and user stories were created for each. The results of this are listed below.

- **As a software developer preparing for technical interviews**, I want to practice coding problems in various languages, including Python and JavaScript, so I can improve my problem-solving skills and be well-prepared for any coding challenges I might face during interviews.
- **As an interviewer**, I want to create and administer coding tests within the platform, observe the candidates as they code in real-time, and have the ability to interact with them through a voice chat function, so I can assess their coding abilities and problem-solving approach effectively.
- **As a new developer keen on learning front-end development**, I need a platform that supports HTML, CSS, and JavaScript, and allows me to see the immediate visual output of my code, so I can practice and understand the impact of my coding decisions on web page designs.
- **As an experienced developer**, I need a platform where I can practice using frameworks like React and collaborate in real-time with less-experienced developers, so we can share knowledge, debug together, and enhance our coding interview skills together.

- **As any user**, I want my coding work and progress on the platform to allow for synchronisation with my GitHub repositories, so I can manage my progress with ease, leveraging the platform's integration for authentication and code storage.
- **As any user**, I want to access a library of pre-loaded coding questions, complete with hints and solutions, so I can practice coding, prepare for interviews, or conduct assessments directly within the platform with a peer.

3.3 Requirements

Once the requirement elicitation process was complete, a set of functional requirements could be established. These requirements would then be used as a reference throughout the project, to ensure the system met all of its initial criteria. The MoSCoW method was used, which is a method of prioritisation in which requirements are labelled as one of the following (8):

- **Must-Have**: Essential for the success of the current delivery phase. The absence of any "Must-Have" item means the project incomplete. These requirements form the Minimum Usable Subset and can be reprioritised in agreement with stakeholders if circumstances change.
- **Should-Have**: While important, are not critical for the immediate delivery cycle. They are significant but can be deferred to a later phase if there are time constraints or alternative solutions.
- **Could-Have**: Optional features that enhance user satisfaction or the overall experience with minimal development effort. These are considered for inclusion when resources and time allow.
- **Won't-Have**: Non-essential for the upcoming delivery cycle by stakeholders and are excluded from the immediate schedule. These items may be revisited for future phases and are outside the current scope.

3.4 Functional Requirements

Functional requirements detail the specific behaviors, functions, and operations of a system. They describe what the system should do, including tasks, data processing, and user interactions necessary to fulfill the business requirements. Examples include user authentication, data input and output, transaction processing, and other specific functionalities that users and stakeholders expect from the system.

Must-Haves

- Users must be able to send and receive code from the peer, ensuring synchronisation.
- Users must be able to, when doing both types of coding, have a text editor that properly accommodates the current language.
- Users must be able to, when doing algorithmic coding, compile their code and see the results.
- Users must be able to, when doing front-end coding, see a live preview of their work.
- Users must be able to save the current code and note state to the database, persisting their progress on the server.
- Users must have a shared note editor where they share text separated from the code; this is where a question would go.
- Users must be able to communicate via the provided voice and video call component.
- Users must be able to swap roles (interviewer, interviewee) with their peer; this gives one of them the power to show a question, its hints and solution.

Should-Haves

- Users should be able to perform CRUD (create, read, update, delete) operations on the question pool; this allows them to publicly edit various properties of a question (i.e., title, body, hint, solution)
- Users should be able to create a room — along with specifying its type (front-end or algorithmic) and whether the room comes with voice call.
- Users should, when creating a front-end room, be able to pick which framework they want to code in: React, Angular, Vue.
- Users should be able to join a room by its unique identifier or browse them freely; browsing allows them to see each room's detail (i.e., type, voice call enabled or not, date of creation, etc.).
- Users, when successfully joining a room, should be able to fetch its pre-existing progress from the database or a peer present in the room.
- The system should, under algorithmic coding, allow a room to support multiple languages (i.e., Python, C#, TypeScript) and users to switch freely between them without losing progress.
- The system should indicate where the other peer is currently on the screen by showing their location with a cursor.
- The system should allow users to end a room's interview/collaboration session; this would disallow interaction with both the text and shared note editor.
- The system should allow for authentication via GitHub, and authenticated users would be able to save their room's progress to a GitHub repository on their account.

Could-Haves

- Users, in regards of the voice call system, could be allowed to turn off their microphones and cameras.
- The system could allow users to edit the current room's name.
- The system could allow the shared note editor to share more sophisticated form of media such as formatted text (i.e., headers, tables, etc.), headings, images, and links.

Won't-Haves

- The system would allow authenticated users to see their histories of which room they created, joined, etc.
- The system would allow authenticated users to create their own private pool of questions.
- The system would allow for writing test cases for each question (whether algorithmic or front-end).
- The system would allow for a higher room capacity; this would require re-implementing voice call.

3.5 Non-Functional Requirements

Non-functional requirements, on the other hand, define the quality attributes, performance measures, and operational standards the system must meet. These requirements do not describe specific behaviors but rather outline how the system should perform and the constraints under which it must operate. They encompass aspects such as performance (e.g., response times, throughput rates), reliability, scalability, security, compliance, maintainability, and usability. Non-functional requirements are critical for ensuring the system's reliability, efficiency, and overall user satisfaction.

Must-Haves

- Users, when doing algorithmic coding, must be able to see feedback metrics (i.e., time taken) from compiling their code.
- The system must allow up to two peers in the same room. The current limit is set to two due to how voice call and role swapping works.
- The system must have loading indicators present on buttons when they are making network requests — including peer-to-peer and server-side requests.

Should-Haves

- The system should have an animation for the landing page.
- The system should, on all pages, be responsive on both mobile and desktop view.
- The system should be modular, allowing the addition of coding languages and frameworks in the future.
- The system should work on all modern browsers.
- The system should be intuitive and easy to use for beginner developers.
- The system should perform well, with quick load and response times, allowing for a better user experience.

3.6 Summary

This chapter provided an in-depth look at various use cases for the Pair Programming Web Application, identifying the range of user roles it supports and their related user stories. It discussed the process by which these stories informed the development of both functional and non-functional requirements. Additionally, it highlighted the use of the MoSCoW prioritisation method to categorise these requirements, thereby establishing their relative importance for the project's success (8).

4 Design

In this chapter, the focus will be on the product's design, beginning with the foundational architecture of the system. Attention will be given to the design of the user interface, detailing its various features, and explaining how each fulfills specific requirements. The chapter will conclude by exploring the assortment of tools and technologies evaluated for the project, highlighting those that were ultimately selected and employed during the implementation phase.

4.1 System Architecture

The Pair Programming Web Application follows the three-tiered web application architecture for maximum flexibility, as proven by researchers from the University of Illinois (9); it can be broken down into mainly three parts — client, server, and database — with the first being the intensive one. With the nature of the software, its focus is on the client and how they interact with each other. The software's architecture works as follows:

4.1.1 Web Application

The clients represent the user interface of the platform, browser-based, where the coding interviews or collaboration take place. The web application has support for frameworks like React, Vue, and Angular, as well as styling solutions like vanilla CSS and Tailwind CSS, making the platform a versatile front-end collaboration environment. It also supports compiling mainstream programming languages like TypeScript, Python and C#.

4.1.2 WebSocket & WebRTC

These technologies are the linchpins for real-time communication within the web application. WebSocket facilitates a persistent, full-duplex communication channel, allowing clients to send, receive messages and then update their interface accordingly; it is also a more performant alternative to even HTTP Polling, a battle tested web-based real-time solution as outlined by Pimentel and Nickerson (10). WebRTC, on the other hand, enables real-time voice calls and video chats. The combined use of these technologies indicates a system designed for interactive, immediate, and seamless user collaboration.

4.1.3 Supabase

It functions as the application's back-end infrastructure. It, while packing PostgreSQL under-the-hood, is tasked with storing interview questions and the code produced within the application, suggesting a focus on technical interviews or coding assessments. Additionally, it manages GitHub authentication, which streamlines the sign-in process for users and allows for the ability to link with their GitHub accounts in order to save progress. Lastly, it also acts as a provider for client WebSocket.

4.1.4 Pusher

It serves a critical role, particularly in establishing WebRTC connections. It acts as a signaling server, an essential component in WebRTC's architecture, facilitating the discovery and negotiation process between peers. This process is crucial for setting up a direct peer-to-peer communication channel, enabling real-time audio and video interactions within the application.

4.1.5 GitHub API

This represents a crucial component where users can upload their code to personal repositories; this is handled by Next.js' built-in API routes. Note that this feature is experimental (which would be elaborated later) and requires user authentication via Supabase.

4.1.6 Judge0

Serving as a computational workhorse, Judge0 is an external service that the web application leverages to execute and compile code; this is handled by Next.js' built-in API routes. This is essential for validating code in real-time, a feature that is particularly beneficial for coding assessments or collaborative development environments.

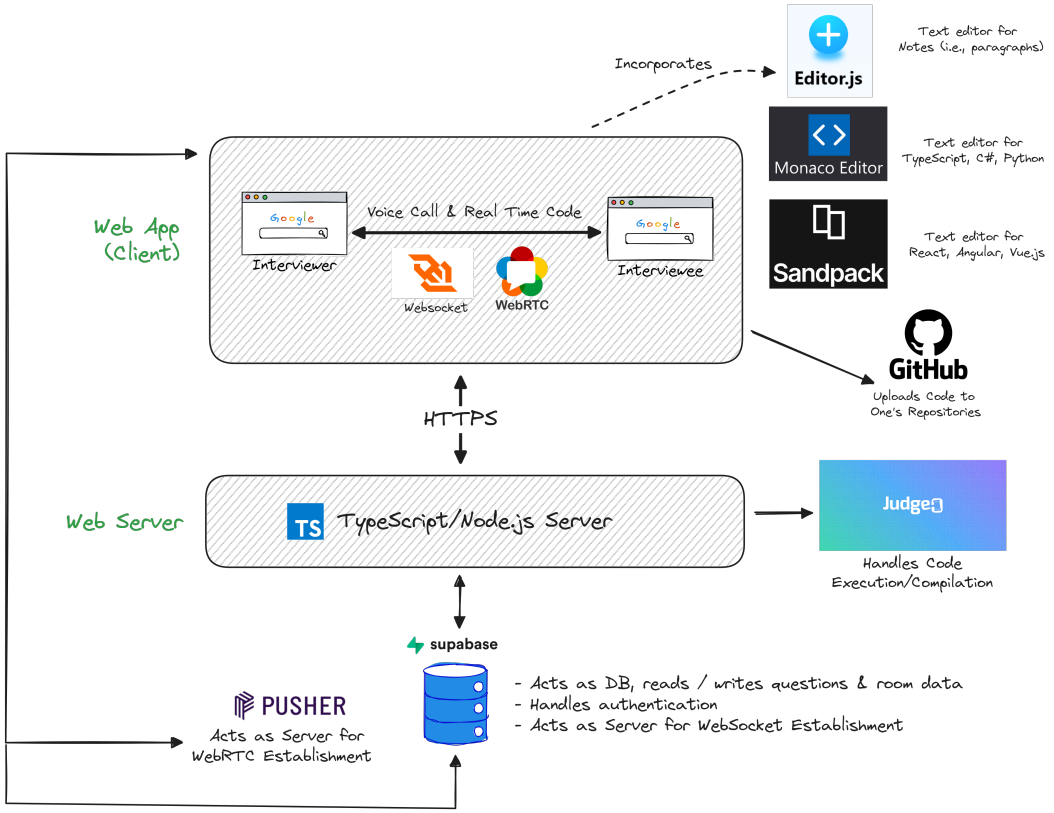


Figure 4.1: *A diagram demonstrating how each part comes together*

4.2 User Interface

The Pair Programming Web Application relies heavily on how peers interact with each other via the application's UI. Here is a breakdown of each notable interface.

4.2.1 Early Prototype

Before moving forward with the implementation and following Karczewski's thought process (11), the coding pages and pop-ups underwent a prototyping stage to understand the objectives and to confirm that all necessary features were included. This process led to the creation of an initial prototype resembling the coding pages and pop-ups, which was chosen to accommodate future module additions seamlessly, in line with one of the non-functional requirements. Although the prototypes did not account for the system's responsiveness on smaller screens, such as mobile phones, where the elements would reduce in size and change their positioning to accommodate for vertical viewing; this would be addressed in the actual implementation. This step proved to be crucial in streamlining the implementation phase, as it provided a clear vision of the desired outcome for each module — maintaining the integrity of the main interface and ensuring user interactions remained intuitive and uncluttered.

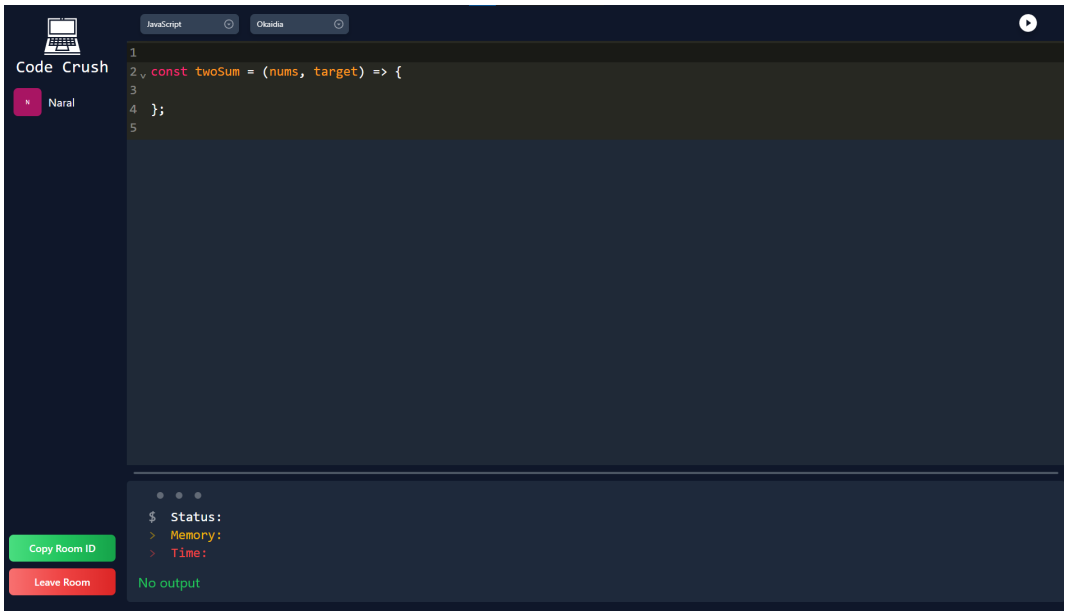


Figure 4.2: An early prototype of the coding pages

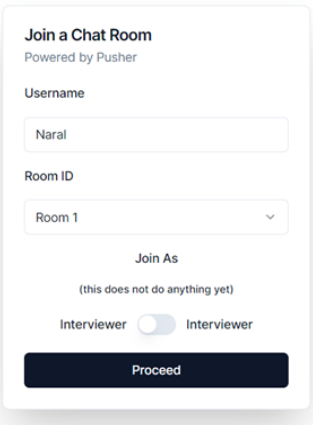


Figure 4.3: An early prototype of the pop-ups

4.2.2 Landing Page

The landing page displayed presents the entry point for Pair Programming Web Application, a platform billed as a key tool for coding practice. It features a background with moving stars, giving it a sleek and modern feel. The central message emphasises the importance of pair programming as leverage for career progression, inviting users to start practicing immediately. The user interface provides several clear options: 'Create a Room,' 'Join a Room by ID,' 'Browse Rooms,' and 'Edit Questions,' allowing for straightforward navigation depending on the user's intention; The presence of these options allows for user-driven actions. At the bottom of the page, there is a link to the project's GitHub repository, further emphasising its open-sourced nature. Overall, the landing page is designed to be user-friendly, providing immediate, accessible options for users to engage with the platform's core features without the need to log in, which could be especially appealing for first-time visitors or those looking to quickly understand the platform's offerings.

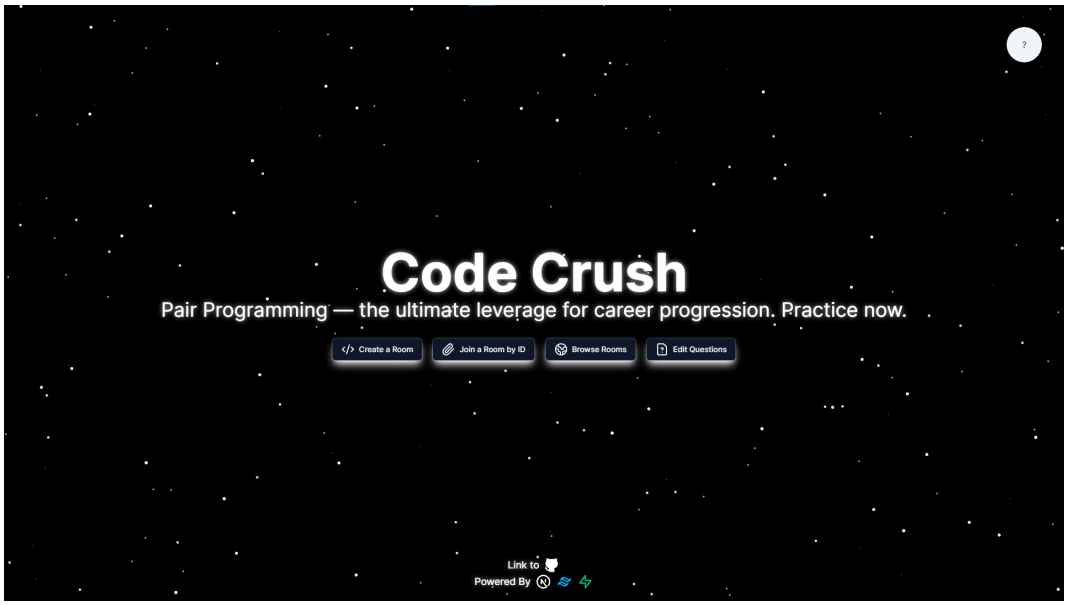


Figure 4.4: *A screenshot of the landing page*

4.2.3 Create Room Pop-Up

The pop-up/modal can be accessed via the "Create a Room" button. This interface is a pop-up/modal for creating an interview room within the web application. The modal provides fields for the user to input the desired public display name for the room and how the user will be displayed to the peers within the room. Dices are used for randomising content in the two aforementioned fields, accommodating users who do not want to go through the hassle of coming up with names. There are also options to specify the focus of the interview room, allowing users to choose between "Front-end" and "Data Structures & Algorithms" (referred to as "algorithmic" in this report). Additionally, there is a feature to enable voice call noted as experimental; this function is intended for one-on-one interactions, like interviews or pair programming sessions mentioned earlier; it comes with an emphasis on real-time interaction as the system is "Powered by Web Sockets."

4.2.4 Join Room by ID Pop-Up

The pop-up/modal can be accessed via the “Join a Room by ID” button. This interface is a pop-up/modal for creating an interview room within the software. The user is prompted to enter a room’s unique identifier needed to access a specific interview room. Below that, there’s a field for ’Your Name’, where participants can enter their display name for others in the room to see — along with a dice for random name generation.

4.2.5 Browse Rooms Pop-Up

The pop-up/modal can be accessed via the “Browse Rooms” button. This interface is a pop-up/modal for browsing rooms and picking the one for joining. Instead of being prompted to enter a room’s unique identifier — users are conveniently shown each room’s details including its name, type, description, date of creation, voice call presence, and so on. The currently selected room would be highlighted in black, have its text color inverted and logo colorised. Like earlier pop-ups, there is a field for the user’s name and a dice to help generate one.

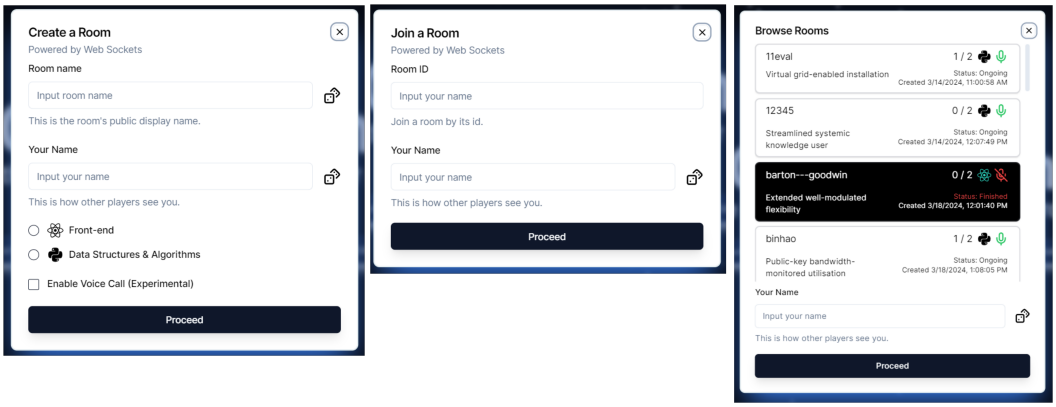


Figure 4.5: A screenshot of the create, join room, and browes rooms pop-up

4.2.6 Login with GitHub

This visual component is located in the top right corner of the landing page and has two different forms depending on the current user’s authentication status. As noted earlier in the report — logging in via GitHub is an experimental feature and only serves one purpose being the ability to upload one’s code to their own repositories. In short, this functionality is achieved by Supabase’s OAuth mechanism, with further explanation coming down the report.

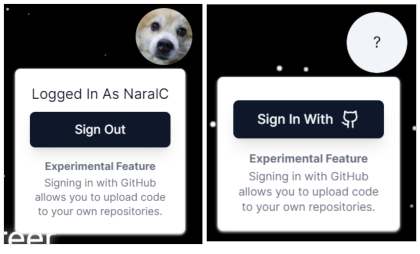


Figure 4.6: A popover for GitHub authentication (left: logged in, right: logged out)

4.2.7 Edit Questions Page

This page can be accessed via the “Edit Questions” button. At the top, users are free to switch to either algorithmics or front-end problems. The questions are displayed in the same manner as how rooms are displayed in the “Browse Rooms” pop-up, with the currently selected one being highlighted and its colors inverted; next to each question lies a delete button that would wipe the question from the database. Multiple input fields are at play here — users can configure a question’s title, hint(s), solution(s), and body; the body uses the same shared note editor from the coding pages, hence formatted texts (i.e., tables, headings, etc.), are fully featured in the question body. Should the user “Create Question” a new question would be inserted into the database, whilst clicking on “Edit Question” would over-write the properties of the currently selected question.

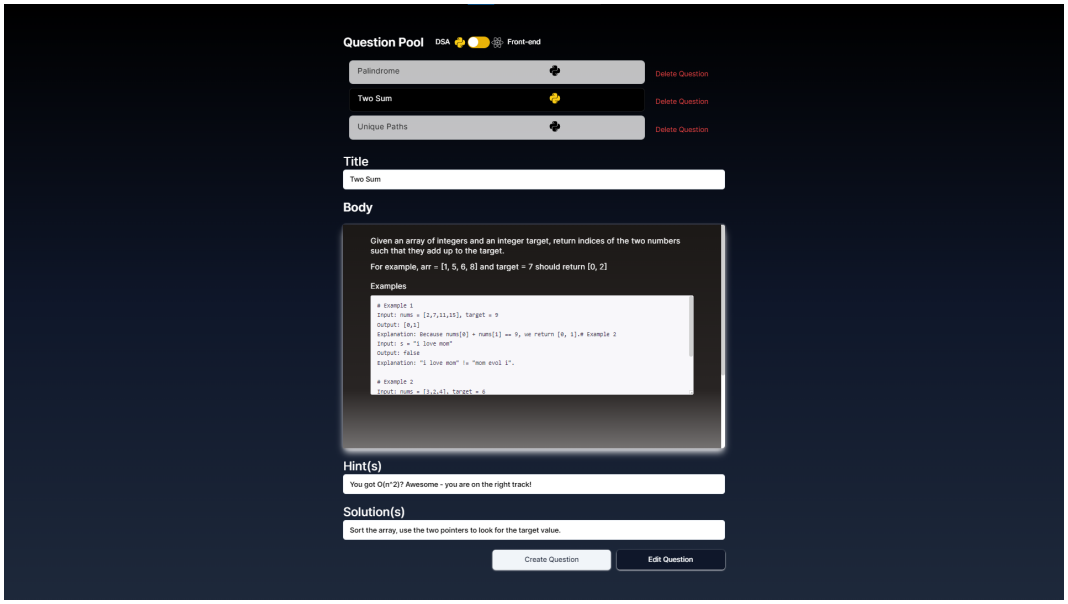


Figure 4.7: *A screenshot of the page for editing questions*

4.2.8 Coding Page

This is the web application’s heart and soul. It is made up of four primary components: code editor, note editor, utility bar, and footer; all of which utilise the real-time functionality that WebSocket provides and are responsive on both mobile and desktop. Note that some components are going to vary in terms of appearance and functionality depending on the room type — either algorithmic or front-end.

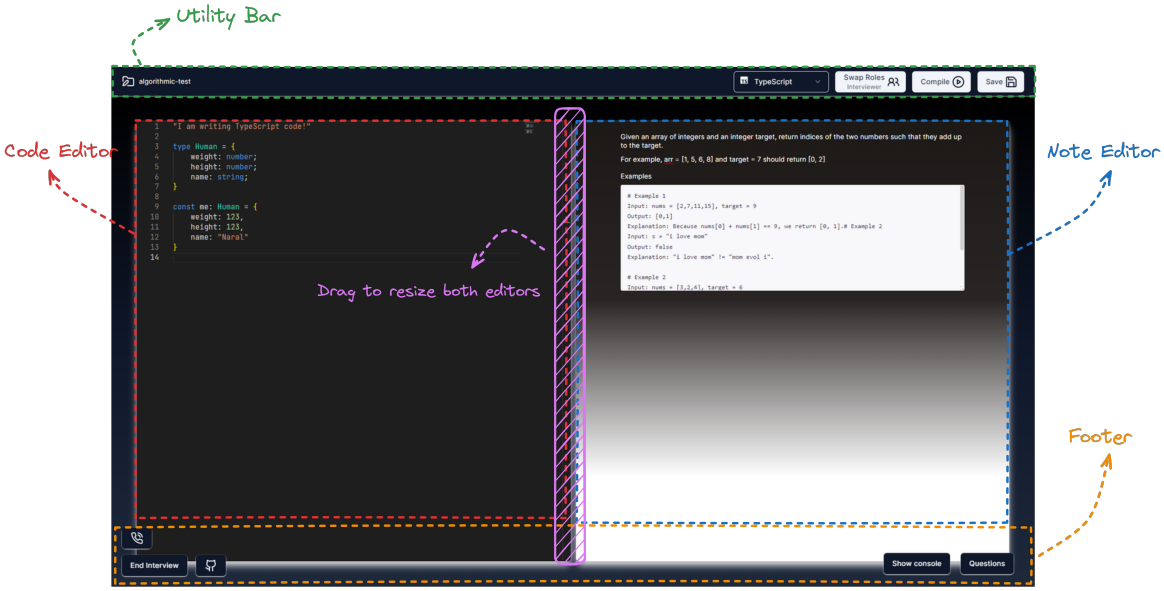


Figure 4.8: A coding page's components; the middle part can be dragged to resise both editors

Utility Bar houses two separate entities being a room name display and a group of buttons.

- **The room name displays** the current room name and allows for the change of it depending on the users' preferences. When clicked — it shows a popover that allows for said functionality, and once the user clicks the arrow or the enter button, their new room name gets formatted to adhere to the GitHub's repository naming standards, which is the kebab-case. Examples include "algorithmic-test", "i-love-coding", "coding-is-awesome". Once the update succeeds, every person in the room sees it reflected on their end. Note that the Utility Bar stays identical for both room types.
- **Button Group** houses multiple clickables capable of various functionalities. If you are in an algorithmic room — you would have the option to, from the dropdown menu, pick the currently available three languages (TypeScript, Python, C#) along with the ability to compile your code with your selected choice of language. The compile button only shows up for the algorithmic type because front-end code can be compiled on the browser right away. "Swap Roles" and "Save" buttons show up in both room types, and they serve the same purpose throughout — respectively switching between the roles of interviewer and interviewee and saving code and notes to the database.

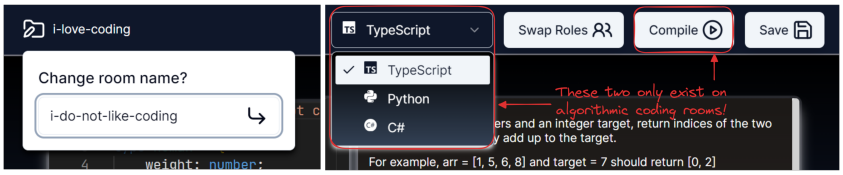


Figure 4.9: A popover for displaying and changing room name (left); Right hand side of the utility bar, composing of buttons of varying functionalities (right)

Text Editors there are three types in the web application:

- **Code Editor (algorithmic)** is powered by the Monaco Editor, the same one that powers Visual Studio Code, a popular integrated development environment from Microsoft. The

editor comes with autocomplete, syntax highlighting, text search and features you can expect from a high-end text editor.

- **Code Editor (front-end)** is powered by the Sandpack Editor, the same one that powers the documentation site of React.js, a mainstream JavaScript framework from Meta. The editor comes with real-time preview, can run and execute three of the most popular JavaScript frameworks (Angular, Vue.js, React.js) and styling solutions (vanilla CSS and Tailwind CSS) right within any modern browser.
- **Note Editor** is where ideas are shared between participants. The editor supports a variety of content blocks, such as paragraphs, headers, lists, and quotes, each provided by a plugin that comes together through the editor's core. This block-style concept offers stability and familiar navigation, unlike traditional WYSIWYG editors that rely on a single content editable element.

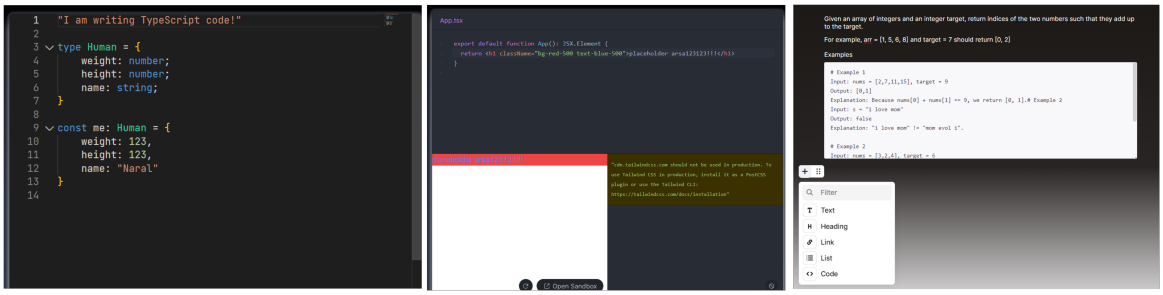


Figure 4.10: *Editors from left to right: algorithmic, front-end, notes*

Footer functions as a utility area, featuring a button to end the interview session, voice-audio call component and GitHub upload button on the left and additional buttons on the right for accessing a code console and a list of questions, providing essential controls for managing the session within the application.

- **Voice-Audio Call** includes both audio and video feed from the peer, powered by WebRTC. Initially, it is minimised as a small button with a phone icon in the middle. When expanded — it shows one self's audio feed along with their peer's. Options to turn off one's microphone's is in future works.
- **End Interview Button** triggers this pop-up/modal when activated. Proceeding with this operation instantaneously results in both code and note editor to become read-only, although code compilation (algorithmic) and code preview (front-end) are still possible. The modal comes with come precautionary text and an outstanding warning button to remind the users that this action is not reversible.

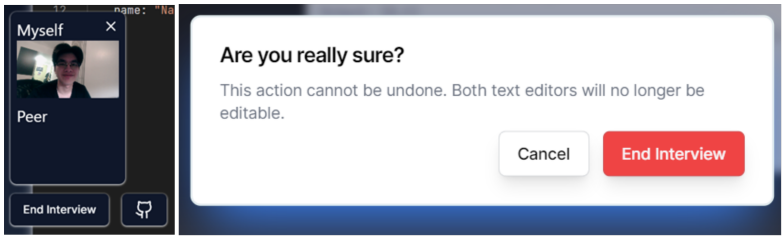


Figure 4.11: *Voice/audio call component (left); Pop-up for ending interviews (right)*

- **GitHub Button** uploads the content in the code editor to one's repositories; the user must be logged in via their GitHub account first for this to happen — else they'd be greeted

with an error message. Provided they are authenticated, the web application will create a new repository for them (according to the room name) then write the code editor content into a markdown file called “README”. On the other hand, should there be a repository with the same name existing, the system would make a commit and overwrite pre-existing content on the user’s behalf.

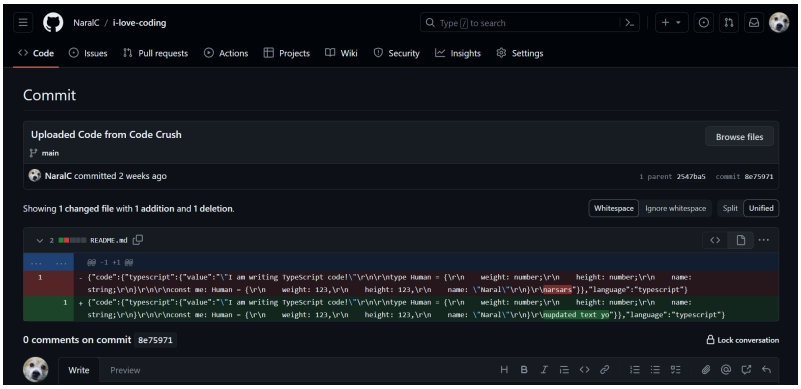


Figure 4.12: *Changes made when we push to the same repository*

- **Show Console Button** triggers output console visibility; note that this only exists on algorithmic rooms since front-end ones already come with built-in previews. It comes in a minimal color schema to provide the best visual cleanliness — primarily black and light slate — and can be either maximised or minimised in size by clicking the expand and cross button. It is equipped with essential metrics for executing code, including memory usage, time taken for execution, final status and most importantly — standard output; these metrics are critical to debugging and optimisation.

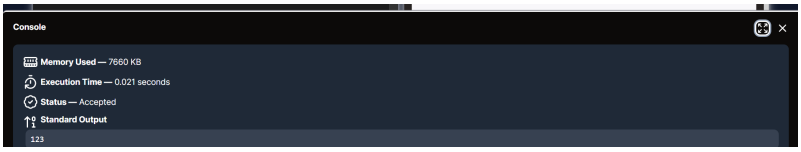


Figure 4.13: *The code console; it pops up when code is compiled*

- **Questions Button** only show up when you are the interviewer, and with it you would be able to put questions into the shared note editor, show hints and solutions of any questions. Clicking the button triggers a popover that displays questions in a vertical order, and showing hints and solutions triggers a popup/modal in the middle of the screen for both participants.

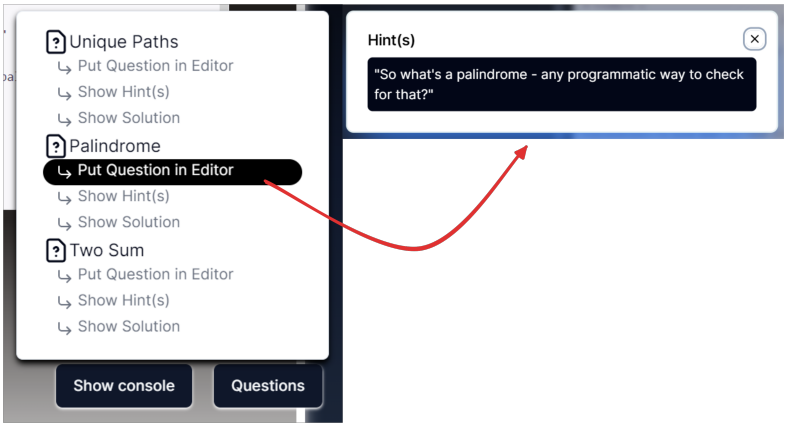


Figure 4.14: *Popover that allows for swapping questions*

4.3 Tools & Technologies

In the world of web technologies — there are countless tools to pick from, which can cause decision fatigue and unnecessary overhead. Since this is a standalone project, modern technologies which are well-maintained and come equipped with diverse functionalities are picked. Here are justifications.

4.3.1 Frontend

- **React**; since this is a client-side heavy project, picking a library that simplifies the development is crucial. Developed by Facebook (or Meta), React is a widely used JavaScript library for building user interfaces, renowned for its huge ecosystem, extensive community support, and straightforward implementation (12). React's component-based architecture facilitates the development of reusable UI components, enhancing development efficiency and maintainability. Its virtual DOM implementation optimises rendering performance, crucial for real-time applications with frequent UI updates (13). Additionally, React's extensive ecosystem, including tools like Redux/Zustand for state management, provides a solid foundation for building complex applications. React's strong ecosystem and abundance of learning resources can also accelerate development and troubleshooting. Given these benefits, React emerges as a compelling choice, balancing efficiency, performance, and scalability for the application's needs.
- **Radix UI** offers a set of low-level, unstyled, accessible components for building high-quality design systems and web applications. When used with React, Radix UI enables developers to create complex, accessible user interfaces without having to start from scratch. This is particularly beneficial for projects requiring a unique brand identity or specific user interaction models, as it saves time while ensuring accessibility standards are met. For example, the component provides buttons, forms, popovers, modals for developers to freely customise.
- **Tailwind CSS** is a utility-first CSS framework that accelerates the development process by allowing developers to style applications directly within HTML markup (or JSX in the case of React). This leads to faster styling processes, reduced CSS bloat, and a more consistent design system across the application. Tailwind's configurability and the vast ecosystem of plugins make it an excellent choice for projects that prioritise rapid development and customisable, responsive designs work well together with Radix UI. It complements React's component-based architecture by facilitating the creation of styled components that can

be reused throughout the application (14).

- **TypeScript**, a superset of JavaScript, introduces static typing to the development process, enhancing code quality and developer productivity. In a complex, client-side heavy application, TypeScript helps in identifying and preventing bugs early in the development cycle, improving maintainability, and making the codebase more readable and easier to refactor. Its integration with React ensures that components and their props are type-checked, reducing runtime errors, and facilitating better collaboration among developers through self-documenting code. TypeScript's support for the latest JavaScript features, combined with its powerful explicit type checking, makes it an essential tool for building large-scale, robust web applications.

4.3.2 Backend / Database

- **Supabase** serves as the backend and database solution, offering a powerful yet easy-to-manage hosted solution to a PostgreSQL database. It provides all the benefits of a relational database along with additional features tailored for modern web applications. Supabase's appeal lies in its WebSocket toolkit, which is integral for real-time features like live code collaboration and instant messaging. This real-time capability ensures that users can interact without noticeable delays, crucial for a smooth and engaging user experience in coding interviews or collaborative sessions. Furthermore, Supabase eliminates the need for extensive backend management since it manages GitHub authentication via a plug-and-play package, allowing focus on the application's functionality rather than on maintaining the infrastructure.
- **Next.js**, a wrapper for React, stands out as an optimal framework for applications prioritising client-side functionality as it offers a lightweight TypeScript server, versatile rendering options through Server-Side Rendering and SSG, and an intuitive built-in routing system based on the file structure. Its serverless architecture (where server-side code are only functions that execute when a request comes in) further enhances its appeal by significantly cutting costs and ensuring scalability — along with the fact that we would not be taking care of a full-blown server (15). This combination of features makes Next.js particularly suitable for developing modern, efficient web applications that require minimal server-side code, seamless page rendering for improved performance and SEO, and a straightforward infrastructure.
- **Judge0** is an open-sourced API, essential for executing code in various mainstream programming languages securely and efficiently. It acts as an external API that evaluates users' code submissions, providing feedback on their output, time taken, memory taken and more. This feature is vital for a coding interview/pair programming platform, as it allows for real-time code execution and validation without the overhead of managing execution environments along with security within the application's infrastructure.
- **Octokit**, the **GitHub** toolkit, is another critical integration for managing GitHub APIs efficiently. It facilitates interactions with GitHub repositories, enabling a GitHub authenticated user to upload code to their repositories. This integration supports a seamless workflow for users who wish to store or share on progress, enhancing the application's appeal to developers who rely on GitHub for version control and collaboration.

4.3.3 Real-time Functionalities

For clarification, these two technologies run on each user's browser, while requiring a server to establish/maintain their connections.

- **WebSocket** is essential for enabling real-time, bidirectional communication between clients, making it perfect for collaborative coding environments and coding interviews. It allows for the instant transmission of messages or code changes, ensuring that all participants

in a session can see updates in real time without needing to refresh their browsers (10). This real-time synchronisation is critical for coding interviews and collaborative sessions, where immediate feedback and interaction are necessary for a productive experience.

- **WebRTC** facilitates direct peer-to-peer communication, enabling features like voice and video calls without the need for external plugins or applications. This technology is key to adding a personal touch to the collaborative experience, allowing participants to communicate verbally as if they were in the same room. (16) This can enhance collaboration by making it easier to discuss ideas, troubleshoot code, and conduct interviews more effectively and naturally — although implementing this from scratch also comes with drawbacks which are discussed in future works.

4.3.4 Testing

- **Cypress** emerges as the ideal testing framework due to its comprehensive capabilities — such as being able to test UI components in isolation and backend integration via end-to-end tests. As highlighted by its documentation, it directly integrates into the browser, offering unprecedented access to the application for real-time reloading, debugging, and testing of complex asynchronous operations — a critical feature for applications with dynamic, interconnected components (17). As Morales and Alfredo suggest, Cypress's snapshot features and visual testing capabilities enable a seamless developer experience (18). It is used to test the expected flows in the application such as creating, joining, and browsing rooms along with reading and writing question.

4.3.5 Summary

This section offered a summary of the architecture of the Pair Programming Web Application, followed by an analysis of the various components that constitute the platform, and the design choices made to fulfill the project's specifications. An evaluation of tools and technologies was conducted, leading to the selection of key technologies such as Monaco Editor for algorithmic coding and additional systems for service integration. The chapter concluded with an exploration of the design methodology, from initial wireframes to the development of the final product.

5 Implementation

5.1 Software Engineering Process

During the course of the project, a variety of software engineering methodologies and practices were implemented, each anticipated to enhance the development process in distinct manners. This approach works well for a solo developer, a straightforward way to tackle the project.

5.1.1 Version Control

A large focus was placed on the adequate use of version control throughout the project's development. GitHub was used to host the repository, ensuring that all code was backed up and would not be lost had any local development issues occur. The use of branching also allowed different features to be implemented separately, ensuring that if anything were to go wrong, a full working system would still always be present on the master branch — and features or bugs would be addressed on the development branch.

5.1.2 Issue Tracking

Throughout the development of the software, simplistic issue tracking was utilised at every phase to maintain clarity on completed tasks and those pending completion. A to-do file written in markdown is kept in the repository, for organising tasks by their importance and value to the project via the MoSCoW methodology (8). Completed tasks are checked off with a check mark.

5.1.3 Continuous Integration & Continuous Deployment

Continuous integration was introduced to the project from the start to ensure fast delivery of new features as this is an important first step outlined by Rangnau and others (19). Vercel was the ultimate choice for deploying the web app since it was compatible with Next.js and worked out of the box with GitHub. The pipeline would trigger a build when code was pushed to any branch (primarily development branch) or a pull request happened (i.e., merging into the master branch), along with an indication of whether the build succeeded. The pipeline was set up to run both tests and linting, to ensure the correctness of the new code and the style used to write it. Each successful build generates a unique link to a fully working version of the application — allowing us to check if anything breaks.

5.2 Front-end

5.2.1 Real-Time Synchronisation with WebSocket

- The bread and butter of the web application. A WebSocket connection is established between the client (the user's browser) and the server. This involves the client sending a WebSocket handshake request to the server to upgrade the HTTP connection to a

WebSocket connection. Once established, this connection remains open, allowing two-way communication without the need for repeated HTTP request-response cycles (10). For collaborative coding sessions, managing users working in the same room involves creating and managing session IDs that group users into the same coding session. When a user joins a session, their client connects to the WebSocket server using the session ID, ensuring that messages are broadcast only within the correct session. With the WebSocket connection established, you can now synchronise code in real-time across all clients in a session. When a user types code into the code editor, the changes are sent to the server via the WebSocket connection. The server then broadcasts these changes to all other clients connected to the same session, ensuring that everyone sees the updates simultaneously. An illustrative example would be when a user executes code, the output can be shared in real-time with all session participants; this is particularly useful during educational sessions or coding interviews, allowing for immediate feedback and collaborative problem-solving. The project's WebSocket server is provisioned by Supabase.

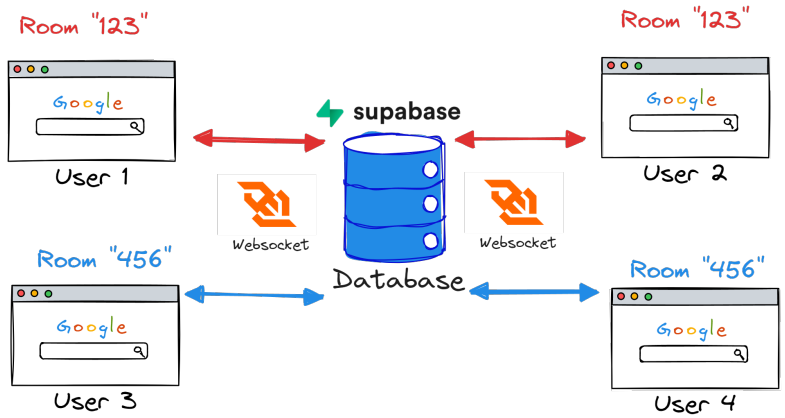


Figure 5.1: *A diagram highlighting changes broadcast by the server to clients*

- Supabase Realtime also comes with another functionality, aptly named "PostgreSQL Changes". This allows the app to listen to changes happening in the database in real-time; changing the room name via the utility bar results in a database update — hence users in the same room and whichever components in the application subscribes to the room data table would receive the change right after the update happens.

5.2.2 State Management with Zustand's Redux-like Reducers

When the application scales, and so do the number of elements that interact with WebSocket grows — it becomes excruciatingly difficult to keep track and manage properly. In such scenarios, adopting a state management solution for React like Zustand, with its Redux-like reducers (elaborated in the following point), can significantly simplify the management of shared state across the application. Zustand provides a minimalistic, yet powerful, way to create global state stores that can be accessed and updated from anywhere in the application — essentially a single source of truth for each corresponding group of components — a huge benefit of this approach as highlighted by McFarlane (20). This approach is particularly beneficial for applications with real-time functionalities, as it allows for a centralised management of the state changes that occur as a result of WebSocket communications and a way to group components by their respective stores (21). For example, pressing the compile button would send a network request to the server — which requires retrieving the current code state from the code editor — and upon receiving the output back, it would be passed to the console for a graceful display.

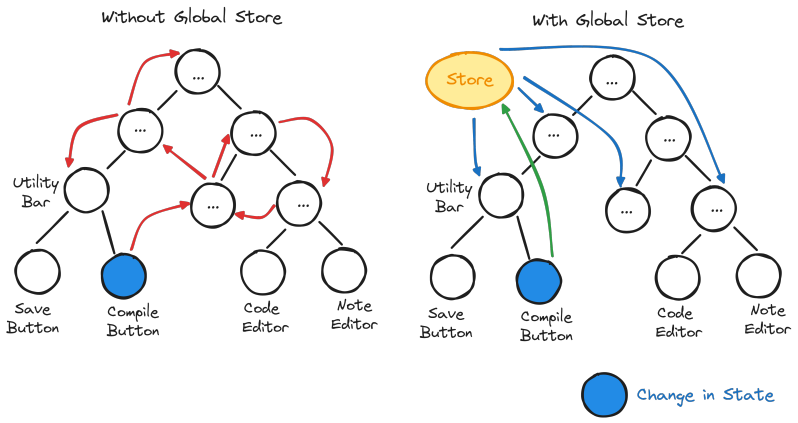


Figure 5.2: *A diagram demonstrating how data is kept in sync between multiple components in a complex front-end-heavy application*

How Zustand's Redux-like reducers works under the hood

- **Actions** are the starting point in the state management flow. They are objects that signal to the store that something has happened. In the context of WebSocket communications, an action might be triggered by receiving a new message, a user joining a collaborative session, or any other event that requires an update to the application's state. Developers dispatch actions to signify user interactions or system events that should result in a state change (21).
- **Reducers** are pure functions that determine how the application's state changes in response to actions. In Zustand, reducers take the current state and an action as arguments, and return a new state based on both. This is where the application's business logic resides, making reducers crucial for maintaining the integrity of the state. With Zustand's Redux-like reducers, the state management logic can be organised and maintained more effectively, even as the application grows in complexity. Reducers ensure that state transitions are predictable and manageable (21).
- **Store** is the central repository of the application's state. Zustand creates multiple stores that hold the application's state tree. A store updates its state by running the reducers in response to actions. The beauty of Zustand lies in its direct mapping of actions to state changes, utilising the pattern to ensure a one-way data flow — avoiding direct data manipulation. The store is what components subscribe to, allowing them to reactively update when the state changes. It acts as the single source of truth for each application's state, making it easier to debug and understand the state at any point in time (21).
- **View**, in our case a React component, reacts to changes in the store's state. When a store updates, components that are subscribed to it re-render with the new state, ensuring the UI is always up-to-date. This reactive nature of the view layer is fundamental in providing a seamless user and developer experience, particularly in real-time applications where the state can change rapidly due to WebSocket interactions (21).

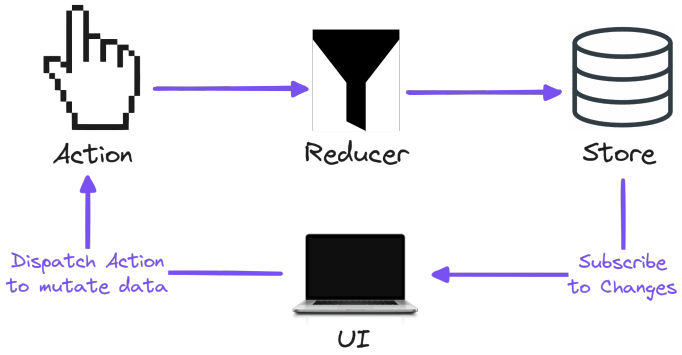


Figure 5.3: *Redux workflow and how it interacts with the UI*

5.2.3 Audio/Video Call with WebRTC

- **Access to microphone and camera**; begins with the web application requesting access to the user's said media input using the browser's getUserMedia() API. This function prompts the user for permission to access these devices. Once the user grants permission, the API returns a media stream object that contains the tracks of the audio and video. This media stream is then used to display the local video on the user's interface and is eventually shared with the remote peer once the connection is established (16).
- **Establishing a connection management instance** is then executed by each user creating an RTCPeerConnection object; it is central to the WebRTC API as it manages the entire connection between peers and is also responsible for handling the negotiation of media and data channels, managing session descriptions, and handling ICE candidates (which are necessary for discovering network interfaces and managing NAT traversal).
- **Finding one's public IP address** can be done via a public STUN server during the ICE (Interactive Connectivity Establishment) process. When the RTCPeerConnection is established, it queries the STUN server to find out the public IP address and port of each peer behind a NAT (Network Address Translation) (16). This information is crucial for establishing the peer-to-peer connection as it allows each peer to know how to reach the other, even if they are behind different types of NATs or firewalls.
- **Signaling** comes next. Before a direct connection can be established, peers need to exchange information such as session descriptions (offers and answers) and ICE candidates. This exchange is done through a signaling process, which does not transmit media but only coordinates the connection. Pusher — another hosted WebSocket solution — acts as the signaling channel (16). It relays messages between the peers, allowing them to exchange the necessary information to find each other and agree on the configuration of the connection. Signaling is an essential step to ensure that both peers are ready to start the communication and are aware of each other's network information and media capabilities.
- **Peer-to-peer** connection can finally be established. Once the signaling process is successfully completed, and both peers have exchanged their offers, answers, and ICE candidates, a direct peer-to-peer connection is established using the RTCPeerConnection — which handles microphone and camera data then streamed directly between peers, allowing for real-time communication without the media having to pass through a server.

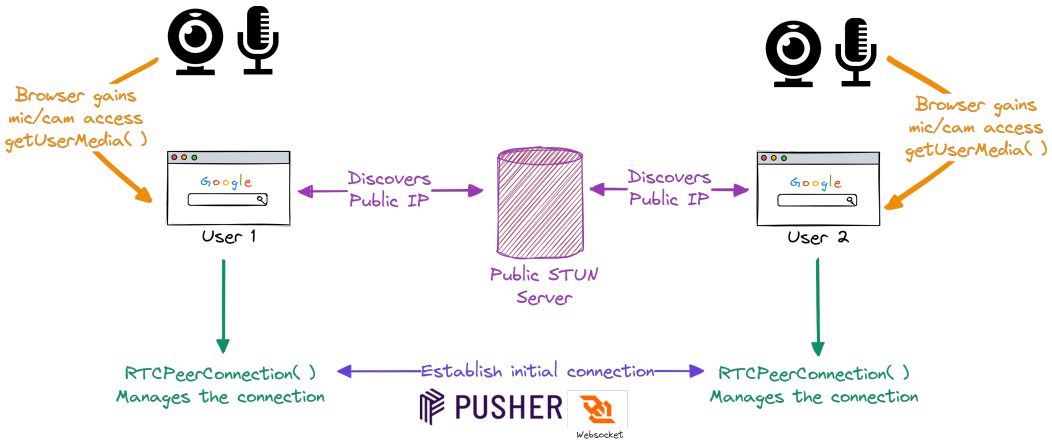


Figure 5.4: A diagram depicting how a WebRTC connection is established on the browser

5.2.4 Handling Input Forms

- Handling input forms on the front-end is another challenge when building a complex web application; in our case — creating, joining, browsing rooms, and editing questions are prime examples of such complex cases.
- **Form or client-side validation** serves as an essential first step in ensuring a positive user experience by immediately identifying and alerting users to invalid data inputs. This prompt feedback allows users to correct errors before the form is submitted, thereby avoiding the delays that occur when the server processes and then returns the data, pointing out errors (22). Once the information entered is corrected — the application would allow data to be submitted to the server and have it handle further business logic there. Most, if not all, popular sites with registration forms provide feedback in the form of "This field is required" or "Invalid date format" and so on. In our case — when creating a room — we require its name, the name of the user creating it, and the room type (along with an optional voice call enablement)

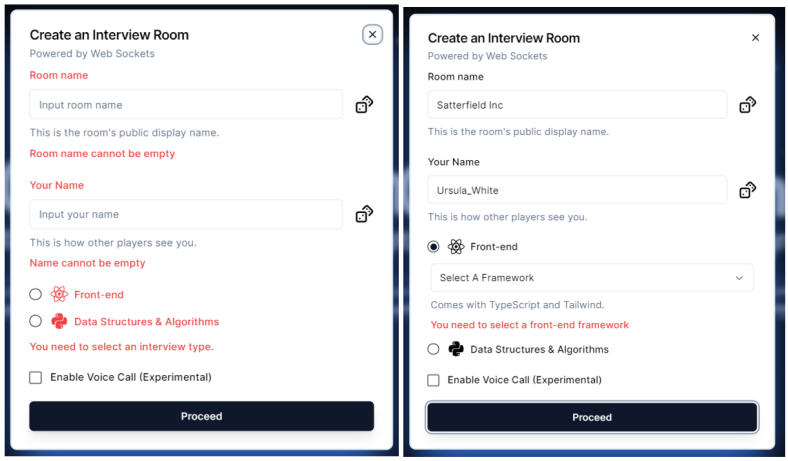


Figure 5.5: Error states and feedback of the "create room" pop-up

- **The anatomy of well-designed HTML forms** are well-structured, semantically correct, easy to use and navigate (with a keyboard). They should also be accessible with ARIA

attributes and proper labels and have support for client and server-side validation; consistent styling with the rest of the application is also a must. Here is an example of such a form — it is semantically correct due to the usage of the right HTML elements for the form and its components ensures that the form is accessible and makes sense to both users and search engines. For instance, <form>, <input>, <label> should be used appropriately to maintain semantic structure (see the image for a common form structure). Since implementing ARIA (Accessible Rich Internet Applications) attributes enhances the form's usability for people with disabilities — a <label> can provide screen reader users with the necessary context for each form element. Proper labeling is also crucial; every input field should have a corresponding <label> element that clearly describes its purpose, enhancing both accessibility and user experience (23). Each component comes with consistent styling and behavior with the usage of Radix UI and Tailwind CSS.

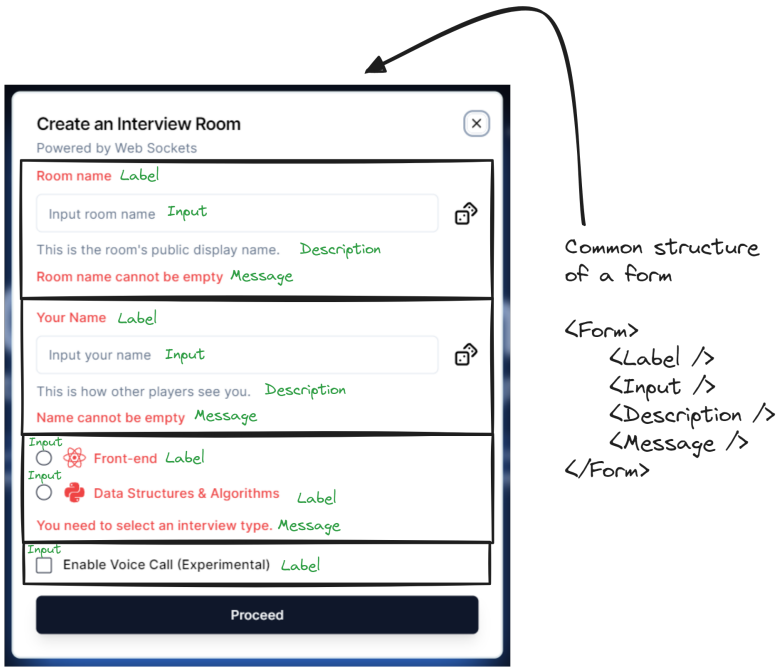


Figure 5.6: *A semantically correct HTML form as said on the Mozilla Web Docs*

- **Input validation** is also used on the server side. It is crucial to remember that client-side validation alone is not sufficient for ensuring the security of your application. Despite its usefulness in improving user experience, client-side validation can be easily circumvented, leaving the server vulnerable to receiving and processing malicious data. Therefore, it is vital to complement client-side checks with robust server-side validation. While the focus here might not be on the specifics of implementing server-side validation, it's important to acknowledge its role in securing the application and safeguarding against potential threats and internal server failures; any data submitted through the client should also be sufficiently verified on the server side. An example from the application would be when a code compilation request comes from a client — the code needs to be validated as a string encoded in base-64 without non-Latin characters, ready to be consumed by an API for compiling code.

5.2.5 Text Editors

- The decision to build these editors from scratch was a serious decision at the start of the project. It began with just using two HTML tags, being <textarea> and <input>, although this approach would fall short when it comes to extensibility. Building autocomplete and syntax highlighting would have been a different project on their own — covering topics such as implementing a compiler/transpiler, syntactic/contextual analysis and native code generation (24). Instead, opting for pre-built editors has proven the better call, and here are explanations of why each was picked, their use cases and capabilities.
- **Code Editor (Algorithmic) — Monaco** is the code editor that powers Visual Studio Code, offering a rich interactive editor that can handle a vast array of languages with advanced features such as syntax highlighting, auto-completion, and real-time error detection. Under the hood, Monaco is built on web technologies (HTML, CSS, JavaScript) and can operate entirely on the browser. It utilises a virtualised rendering technique to manage and render only the parts of the code visible on the screen, ensuring high performance even with large files. In our web application — it is used for accommodating three different languages (Python, C#, TypeScript); said accommodation includes, but is not limited to, syntax highlighting, autocomplete, text search and more. When it comes to code compilation, the Zustand store takes the data from Monaco editor and sends it to the server.
- **Code Editor (Front-end) — Sandpack**, leveraging the power of CodeSandbox, is a comprehensive component toolkit designed to build and share web applications through the browser. It abstracts the complexity of code transpilation and bundling, using workers and sandboxes to execute code securely and efficiently in the browser, with real-time previews that reflect code changes instantly; an editor of the same concept is also used by CS50, a computing course offered by Havard University — demonstrating how strong in-browser code preview can be (25). In our case — it is used for accommodating three different front-end frameworks (React, Angular, Vue); said accommodation includes, but is not limited to, live-preview, text search and an integration with a styling system, Tailwind CSS, which is a less verbose alternative to vanilla CSS.
- **Shared Note Editor — Editor.js** is a block-style, open-sourced editor that provides a clean, modular way of creating and managing content. It separates each piece of content into blocks (e.g., paragraphs, tables, lists), allowing users to focus on one block at a time. This structure facilitates easy content manipulation and outputs a clean in JSON format, which can be rendered on the frontend or stored in a database with ease. Under the hood, Editor.js is also built with web technologies (HTML, CSS, TypeScript) and focuses on extensibility and customisation. In our case, it is for sharing formatted texts (i.e., paragraphs, tables, code blocks).

5.3 Back-end

5.3.1 CRUD (Create, Read, Update, Delete) Operations on Questions

The operations are conveniently handled by Next.js' API routes — which are serverless lambda functions under the hood; they are essentially pure JavaScript/TypeScript functions that execute when a network request comes in, and ceases after it sends out a network response. They automatically scale with the demand, handling increases in traffic without any additional configuration. This serverless architecture also means that a project with a solo developer can focus on writing business logic without worrying about server management, deployment, and scaling issues — taking a huge load of work off since we are not taking care of a full-blown server (15). These API routes are complemented by Supabase's built-in query builder tool which abstracts away raw SQL and presents us with intuitive and chainable JavaScript/TypeScript methods to interact with its PostgreSQL instance and perform CRUD operations — also known as an ORM (Object

Relational Mapper); this makes it significantly easier to manage database interactions without diving deep into raw SQL syntax (26). For example, methods like .insert() can be used to insert a new question into the database — while reading (querying), updating, and deleting records can be done using .select(), .update(), and .delete() methods respectively.

5.3.2 CRUD Operations on Room Data

The operations are conveniently handled by Next.js' API routes — just like the previously mentioned operations on questions. This serverless architecture simplifies the backend structure and enhances scalability and maintainability (15). For instance, when you need to update a room's code and note data, utilising Next.js API routes provides a straightforward and efficient method to interact with your PostgreSQL database through Supabase. In raw PostgreSQL, the equivalent operation to update a room's code and note data would involve executing an SQL statement directly on the database.

Figure 5.7: *Supabase's query tool (left) VS Raw PostgreSQL (right)*

5.3.3 Code Compilation (Judge0)

It is handled through an integration with Judge0, an open-sourced, scalable tool that provides a robust API for compiling and executing code in various programming languages. Judge0 API abstracts the complexity of setting up and managing a compilation and execution environment, allowing the application to focus on user interaction and core functionality. When a user submits code for compilation, the application sends a request to the Judge0 API with the source code and the language identifier. Judge0 processes this code within isolated environments, ensuring security and preventing malicious code from affecting the system. Once the compilation and execution are complete, Judge0 returns the results, which can include the output, any compilation errors, or runtime messages. This integration enables real-time feedback on code correctness and performance, essential for a collaborative coding environment.

5.3.4 GitHub (authentication + repository upload)

It involves leveraging GitHub's OAuth service for secure user authentication, done through Supabase's authentication tooling. Once authenticated, users can link their activities on the platform with their GitHub profiles, enabling features such as repository uploads. For the repo upload functionality, the application utilises the GitHub API (via Octokit, a client library for interacting with GitHub's REST API) to programmatically create repositories and push code. This integration allows users to export their projects or code snippets directly to a new or existing GitHub repository, streamlining the workflow for saving and sharing code. The use of GitHub authentication and API interactions enriches the application's ecosystem, connecting it with a widely used platform for code hosting and collaboration, thus enhancing user engagement and the overall value proposition of the application.

5.4 Hosting/Deployment

5.4.1 Vercel

The default hosting platform for Next.js/React applications. Both are developed by the same organisation, thus Vercel comes out of the box with the support for GitHub CI/CD and the latest Next.js features/updates, including experimental ones that push the limit of Next.js' serverless architecture. Once connected to one's GitHub repository, it will automatically build and deploy the application when code is pushed, regardless of which branch it is — making this an ideal tool to deploy the web application. It also comes with a generous free tier which does not require up-front credit card and provides more than sufficient build time and bandwidth.

5.4.2 Supabase

It provides a fully managed service that removes the overhead of database administration. It offers a PostgreSQL database with real-time capabilities, making it an ideal choice for applications requiring live data updates, like collaborative environments. Supabase enriches its database offering with an array of additional services, including authentication (which supports GitHub OAuth), authorisation, and real-time functionalities through WebSockets, simplifying the development process and reducing the need for additional backend infrastructure.

5.4.3 Pusher

Another WebSocket service, it is used for facilitating the initial handshake/connection in the application's voice and audio call. It provides a robust set of APIs to enable real-time bi-directional communication between clients, abstracting the complexities of direct WebSocket and server-side implementation. This allows the application to implement interactive, real-time features without managing the underlying infrastructure, focusing on user experience and core functionality. Although redundant since Supabase already offers a WebSocket API, migrating away from Pusher to reduce redundancy was not at the top of project requirements/priority list.

5.4.4 Alternatives

These are mentioned in case one feels the need to own the infrastructure and account for scalability in case traffic starts to flood in. The three hosting providers mentioned beforehand are essentially wrappers for more complicated cloud services and libraries — aiming to provide a convenient and smooth developer experience which works well for a solo developer — although racks up bill faster than other complex and barebones services since the better developer experience is what keeps their business model afloat (27).

- **Next.js** applications can be hosted on any virtual private server that supports Node.js, such as AWS, Azure, Google Cloud, or DigitalOcean; all of these come with a more predictable pricing scale as the application sees a spike in usage traffic (27). Particularly on AWS, it can hosted a content distribution network like CloudFront, while routing its API routes to AWS Lambda (28). If we were to swap out Supabase like in the next point — there are countless authentication libraries made for Next.js available in the ecosystem.
- Similarly, **PostgreSQL** can be self-hosted or managed through cloud providers (again AWS and DigitalOcean would work) instead of Supabase, offering more control over the database configuration, security, management, and most importantly — a more affordable price at the cost of a complicated setup (28).
- For real-time communication, self-hosted solutions like deploying your own **WebSocket servers** or using open-source real-time communication frameworks can replace Pusher and Supabase in the long run; running a barebone server in any programming language that supports the WebSocket interface would work too.

- All of these can even be deployed to a single virtual private server with Docker — but would require more time dedicated to system administration, maintenance, and deployment. These alternatives require more setup and maintenance but provide greater control and potential cost optimisations based on specific use cases and scalability needs (28).

6 Evaluation

This chapter outlines the assessment techniques employed in the project, encompassing end-to-end testing, performance assessment, and feedback from user acceptance testing. The outcomes of each method are analysed to thoroughly evaluate the software. Ultimately, the purpose of project evaluation is to confirm completion by the project's conclusion.

6.1 End-to-End Testing

In the development of the web application, Cypress, a browser-based test automation tool, was utilised to create end-to-end tests. This testing framework simulates real-user interactions within the browser, interacting with the DOM to verify the application's behavior across various critical functionalities. By automating actions that mimic user behavior, such as clicking, typing, and navigating, Cypress ensures that the system's front-end and backend components work seamlessly together to deliver the intended user experience (18). These tests extensively covered key features, providing a robust validation framework to ensure each aspect of the application functioned as expected, enhancing reliability and user trust in the platform.

- **Room Creation**: the test covers this functionality completely end-to-end. In order to test that all fields are validated properly and individually, Cypress crawls through each field, input them with randomised data, while deliberately leaving one empty. Subsequently, Cypress proceeds with a case where all fields are properly filled, in which case it would intercept an API call handling the creation of rooms, checks if its response is healthy and as expected. Lastly, to mock a user being re-routed to the room they created — Cypress validates the browser URL once it changes.
- **Room Browsing and Joining**: just like room creation, Cypress covers cases to test that each field is properly validated. On top of that, it also tests that users would be able to join rooms by their id by checking the changed url after re-routing happens; the ability to pick a random a room before joining is also automated by Cypress via simulating a click on a random card from the room list.
- **CRUD Operations on Question Pool**: the application's CRUD operations on the question pool were thoroughly tested with Cypress, ensuring effective interaction with the API for managing both algorithmic and front-end questions. Tests validated the functionalities of fetching, creating, editing, and deleting questions within the pool. For instance and to simulate a user performing CRUD operations, Cypress automatically populates fields like title, description, question body, and hints — then presses the respective submit button; this is followed by checks to confirm their successful addition along with the correct functionality of deleting and editing questions. These tests ensures the API's reliability in question pool management.

While Cypress provided a robust framework for testing the above scenarios, it was acknowledged that certain aspects of the application, specifically real-time coding collaboration, could not be effectively tested using Cypress due to its inability to drive multiple browsers simultaneously (17). Recognising the importance of real-time collaboration features in the application, the decision

was made to exclude these components from Cypress tests. Instead, it was determined that these features would be better validated through hands-on testing with real users. This approach allows for a more authentic assessment of real-time coding functionalities, capturing user interactions and feedback in live scenarios, which are difficult to simulate accurately in automated tests.

6.2 Performance Test

To ascertain if the system meets the non-functional requirement of optimal performance, including swift loading and response times, a comprehensive performance evaluation was conducted using Lighthouse, a tool from Google Chrome. This evaluation tool offers insights into the application's performance, adherence to best practices, and accessibility, providing recommendations for enhancements. The evaluation focused on a desktop navigation context, executing five audit iterations to derive average scores for key metrics such as performance, accessibility, best practices, first contentful paint, largest contentful paint, total blocking time, speed index, and cumulative layout shift. These results, compiled into a table, offer a detailed view of the Pair Programming Web Application's performance across various pages.

Website	Performance	Accessibility	Best Practices	SEO
Landing Page	23	86	93	80
Questions Page	55	87	93	78
Coding Page - Algorithmic	35	93	81	89
Coding Page - Front-end	36	94	74	89
CodeSignal (Coding Page)	41	84	41	91
LeetCode (Coding Page)	46	76	41	100
HackerRank (Coding Page)	72	84	70	92

Table 6.1: *Lighthouse performance tested on all pages of the system and multiple commercial software.*

The performance evaluation of the Pair Programming Web Application revealed it excels in numerous areas, though it falls short in performance metrics. Complex pages, like the feature-rich landing page adorned with animated text and 3D backgrounds, as well as coding environments hosting dual text editors, lag behind simpler pages such as those for question navigation. This performance dip primarily stems from the heavy load times demanded by sophisticated plugins for code and note editing functionalities, leading to a sisable JavaScript bundle. The main recommendations for enhancement focus on reducing the main thread's load and trimming down surplus JavaScript. Interestingly, this performance challenge isn't unique to our application; major platforms like CodeSignal and LeetCode encounter similar obstacles, attributed to their advanced text editors and dynamic user interfaces, which inherently demand more resources than simpler web content. Despite these hurdles, the application still scored commendably across the board in our audits, confirming that it meets the established speed criteria.

6.3 User Acceptance Testing

As the application is designed to be a publicly accessible system, a user evaluation involving two persons was necessary in order to fully examine how effective the system is in achieving its goals — specifically the real-time collaboration part.

6.3.1 Plan

Step 0 - Introduction: The user would be briefed on what the platform is for (3 minutes)

1. The whole process takes 20-30 minutes including a questionnaire, requiring a remote connection from the user's computer. The process is done completely virtually. A GitHub account is also required.
2. The evaluation would be using the web app for coding interviews, to test one's coding skills.
3. The user will undertake the roles of interview and interviewee, switching half-way.
4. The user will have the chance to upload their code/interview progress to their GitHub account.
5. The user will have the chance to interact with the question pool, whether it be creating or deleting questions.
6. The user will be given a post-experiment consisting of questions about the whole platform.
7. Note that in case of the voice/audio call not fully functioning, they would be instructed to move to Microsoft Teams.

Step 1 - Picking a Question (3 minutes)

1. The user navigates to the questions page.
2. The user decides which question type they prefer — either algorithmic or front-end.
3. The user is free to pick one of the pre-made questions (and edit it to their liking). They will have the option to select their selected question in the coding page itself.

Step 2 - Create Room (3 minutes)

1. The user comes back to the web app's landing page - then they authenticate with their GitHub account.
2. They would be instructed to click "Create Room", then create whichever room type they decided upon earlier in Step 1, although they'd be explicitly told to enable voice call for testing purposes.
3. The user would share the room ID with us, so we can join the room to start the process.

Step 3 - Be the Interviewer (5-8 minutes)

1. The user familiarises himself/herself with the coding page's user interface. They are given a quick walkthrough of each UI element.
2. The user starts by selecting the question they picked earlier; the question would automatically go into the shared note editor.
3. A timer of 5 minutes would start.
4. The user can give hints to us for guidance. They can also end the session by showing us the solution.
5. Note that completing the coding question is not the point of this evaluation, but to just prove the platform can facilitate said process.
6. The user exits the room (back to the landing page) after the solution is shown or the timer ends.

Step 3 - Join a Room (3 minutes)

1. We would join a room ahead of time, and provide the user with an ID so they can join.
2. They can also join by browsing for the room instead.

Step 4 - Be the Interviewee (5-8 minutes)

1. The session is started by us dropping a pre-selected question according to the user's preferred type.
2. A timer of 5 minutes would start.
3. The user can request hints from us for guidance. They can also end the session by asking for the solution.

4. Note that completing the coding question is not the point of this evaluation, but to just prove the platform can facilitate said process.
5. The user **does not yet** exit the room after the solution is shown or the timer ends.

Step 6 - Try GitHub Feature (3 minutes)

1. The user would be instructed to click the GitHub button located on the bottom left of the coding page; this would start uploading the code to their GitHub repository.
2. They are then asked to check their GitHub account for the uploaded code.

Step 7 - Questionnaire (10 minutes)

1. The user would be provided a link and kindly requested to complete a post-experiment questionnaire — to rate the platform's usability.

Questionnaire

After completing all tasks, participants will be invited to complete a post-evaluation questionnaire. It is designed to collect comprehensive feedback, encompassing both qualitative and quantitative insights into their user experience. The quantitative part will leverage the System Usability Scale (SUS), a reliable tool to gauge the usability aspect of the system, evaluating ease of use, efficiency, and overall user satisfaction; the methodology is battle tested with the fact that many enterprise level teams are still using it, which has been over two decades since its creation (29). In the qualitative segment, participants will encounter open-ended questions that allow them to offer suggestions for enhancements and express their opinions on whether they perceive the web application as an effective tool when it comes to pair programming.

6.3.2 Results

Based on the questionnaire results, it is revealed that — for the most part — Pair Programming Web Application — is well integrated and facilitates the process of pair programming in a satisfactory manner. Despite this, there are a few caveats.

- Participants universally acknowledged the straightforward navigation and feature accessibility of the web application, praising its well-crafted user interface. However, perspectives diverged on the necessity of technical assistance for initial usage. This variation in user experience emphasises the need for a more adaptive onboarding process to accommodate varying levels of technical understanding.
- The landing page's visual appeal and user-friendly design were highlighted as significant positives, with functionalities like room creation, joining, and browsing, along with GitHub integration, receiving approval for intuitiveness and reliability.
- The code and note editors were well-received for their utility and ease of use, promoting productive coding sessions. However, feedback on the peer cursor functionality indicates a distraction, with a consensus pointing toward the potential benefits of transitioning to an in-editor highlighter for improved focus and less visual clutter.

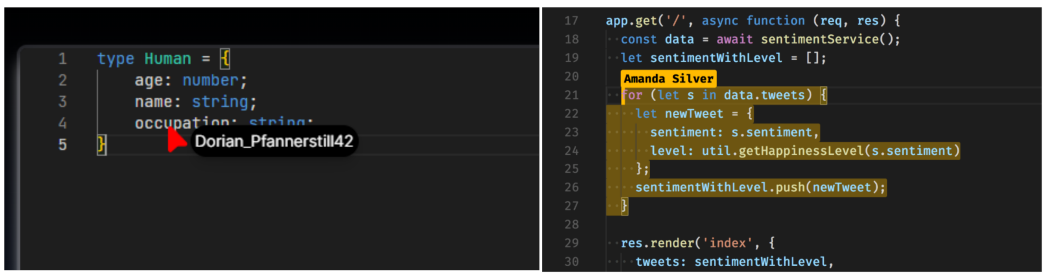


Figure 6.1: *Peer cursor (left) vs in-editor highlighter (right) (30);*

- It was unfortunately found that both parties cannot type on the code editor on different locations at the same time as this would cause the code editor's cursor to jump around, although thankfully this oversight couldn't have been found without the user testing.
- Voice/audio call is another feature favored by the majority of participants, although it doesn't always run smoothly (i.e., not starting at all with some); this could be an issue with network security and browser compatibility in general, since we're implementing our voice/audio the bare-bone vanilla way through basic STUN servers.
- The "upload to GitHub repository" worked for all participants and the majority found it useful. They also found the interface and ease of use of the questions pool page to be satisfactory.
- There are also a few things that could be done more gracefully — such as displaying a helpful message when a compilation error happens, separating the uploaded code into their own respective files (i.e., question-1.py or question-1.ts) instead of a single markdown, popovers being hidden by other elements, and browser extensions that can force dark mode causing visual confusion (see figures below).

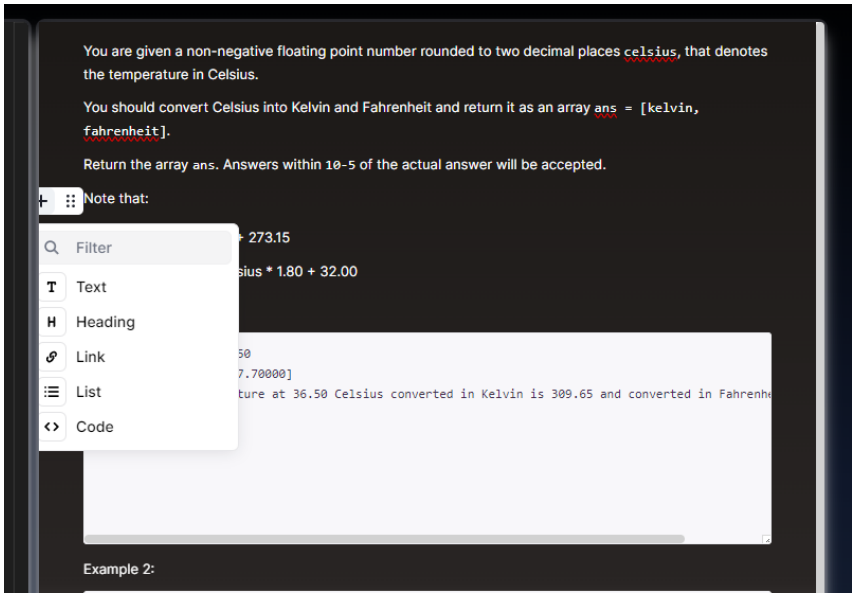


Figure 6.2: *A note editor popover being obstructed by the code editor; this happens on smaller screens or when resising both editors*

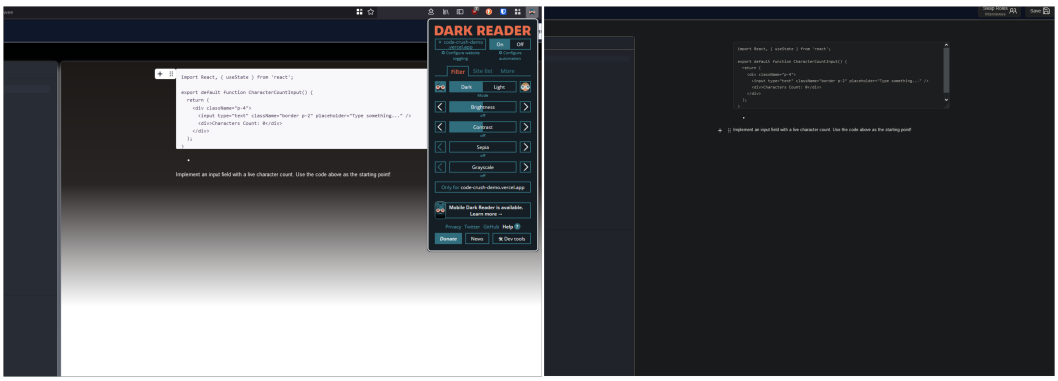


Figure 6.3: *App's default theme (left) vs forced dark mode (right); this causes styling inconsistency and makes it harder to see*

Here are notable features/changes suggestions from the participants.

- Participants expressed a desire for a self-guided portal where they could hone their coding skills independently. This portal could offer a range of exercises and challenges across various difficulty levels and programming languages, allowing users to practice at their own pace without the need for a partner. Incorporating progress tracking and personalised feedback within this portal could further incentivise learning and skill development.
- There's a clear interest in a feature that allows users to review their successful code submissions. A dedicated section to view all passed submissions could serve as a valuable learning tool, enabling users to revisit their code, understand their growth over time, and potentially share their solutions with others.
- The addition of more comprehensive test cases was another popular suggestion. Expanding the test case feature to include more varied and intricate examples could help users better understand problem requirements and edge cases. Furthermore, offering an explanatory breakdown of each test case's purpose and expected outcomes might improve users' debugging skills and enhance their overall coding capabilities.

6.3.3 Summary

This chapter provides an in-depth review of the various evaluation methods applied to Pair Programming Web Application. Initially, the focus was on end-to-end testing, which revealed that it technically couldn't cover the real-time functionalities due to the tool's inherent limitations, attributed to the project's design nature. Subsequently, the system's performance was assessed through Google Chrome's checkup tool — and despite having below average metrics for performance — it is justifiable since this is a trend amongst other enterprise-grade systems of similar functionalities as well. A thorough user evaluation was also conducted, leading to key insights and potential improvements into the application's functionality — which was well-received for its appealing interface and robust yet straightforward functionalities; said key improvements wouldn't have been possible without the user testing.

7 Conclusion

This document has previously explored the rationale and objectives for creating Pair Programming Web Application, followed by detailing the specific requirements necessary to achieve these goals. The architecture and actual development of the system were then explored, leading to an in-depth evaluation. Subsequently, this concluding chapter offers a recapitulation of the entire project and delves into prospective enhancements that could amplify Programming Web Application's utility. To conclude, reflections on the project and the methodologies employed throughout its development will be shared.

7.1 Summary

In summary, the Pair Programming Web Application is a comprehensive and innovative platform designed to facilitate coding interviews and collaborative programming sessions. The application boasts a modular and adaptive architecture, optimised for real-time interactions — powered by WebSocket — and streamlined collaboration across various programming languages and front-end frameworks. It offers key features such as live code editing , preview, and execution, along with an extensive library of interview questions, and seamless integration with GitHub for code sharing and management.

The application also facilitates robust communication channels among users through integrated voice and video calls, powered by WebRTC. Additionally, it provides an intuitive interface for navigating and utilising the question pool, further enriching the user experience.

The evaluation of the application encompassed several methodologies, including end-to-end testing, performance assessment, user experience evaluation, and validation against defined requirements. User testing indicates that the Pair Programming Web Application stands out in fostering collaborative problem-solving, enhancing the preparedness of interviewees, and streamlining the pair programming process for organisations. Its user-friendly design and learnability are likely to decrease the onboarding time and training costs for new users. The application not only supports the technical assessment and interactive learning but also fosters a more connected and interactive community of programmers and interviewers.

7.2 Future Work

The hindsight from writing this report highlights countless potential improvements to Pair Programming Web Application, and so does the user testing; it sheds light on a few technical oversight that, when dealt with correctly, can skyrocket the user experience and ease of use.

- As discussed in alternative deployment options, both Supabase and Pusher offer WebSocket APIs ready for use — therefore opting out of one of them would reduce technical redundancy and overhead. Opting out of both and implementing our own WebSocket server is a viable option too since we wouldn't be relying on third-party services. This also applies to self-hosting a PostgreSQL server instead of letting Supabase handle it.

- Due to the disarray of user feedback when it comes to using the system without the help of a technical person — implementing a tour functionality where it takes first-time users around the web app to for demonstration purposes would prove useful.
- Replacing the peer cursor with an in-editor highlighter for a clearer visual indicator would prove a wise decision as the highlighter would not obstruct other elements present on screen.
- Technical improvements to both the code and note editor include allowing both parties to type at the same time on different lines/position without changing the code editor's cursor location, embedding the note editor with the ability to share links, pictures and videos, and accounting for better visual complexity.
- The voice and audio call functionality is peer-to-peer; in other words, all peers need to connect with one another, making this currently unscalable. To accommodate more than two parties in the room/call — approaches like MCU and SFU which requires implemeting our own server to handle and centralise multiple WebRTC connections come to mind (31).
- Adding test cases to run against emerged as a valuable addition, with the aim of diversifying the range of test cases and helping users understanding the scope of questions better. This could significantly boost the users' ability to debug.
- A solo self-paced learning portal was also suggested by test participants. Such a feature would allow users to practice autonomously, with progress tracking and past successful code submissions to encourage continuous learning and reflect on their progress.

7.3 Final Reflection

Through this project, I've gained substantial insight into orchestrating and executing a major project individually; it has shown me the critical role of project management, spotlighting essential methods and strategies to support it, while also emphasising the necessity of sticking to defined scopes and realistic deadlines. It has also allowed me to dive deeper into the world of programming and application development; things like serverless architecture, front-end complex state management and real-time APIs such as WebSocket and WebRTC are prime examples of things not available in classrooms. Subsequently — the value of user evaluations was not something I was fully aware of, let alone its potential to spot oversights in the project. Ultimately, it was an invaluable learning experience in university.

8 Appendices

8.1 Final Deliverable

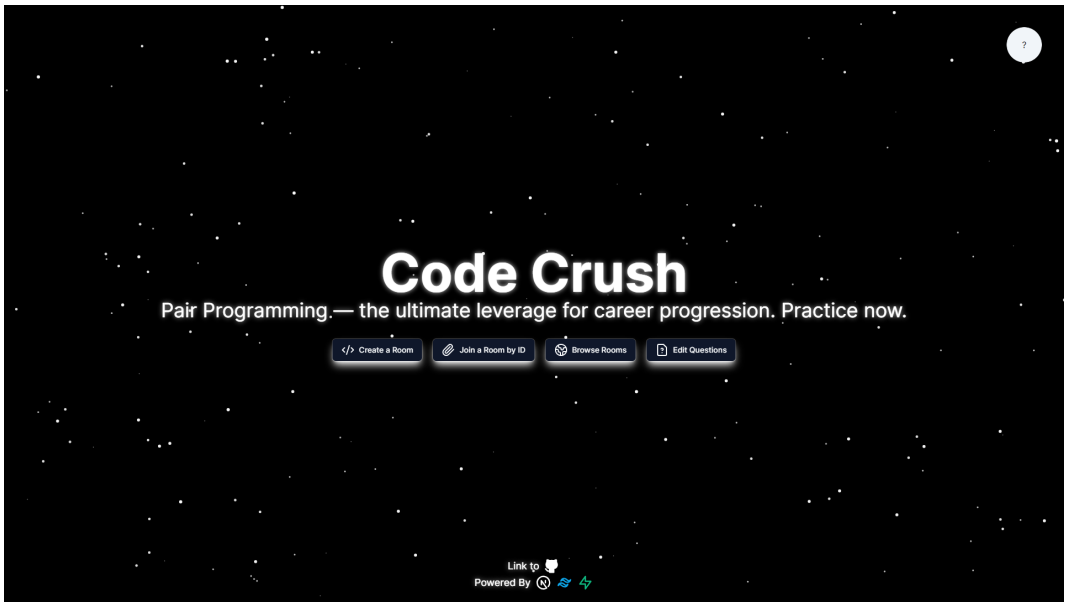


Figure 8.1: Landing page

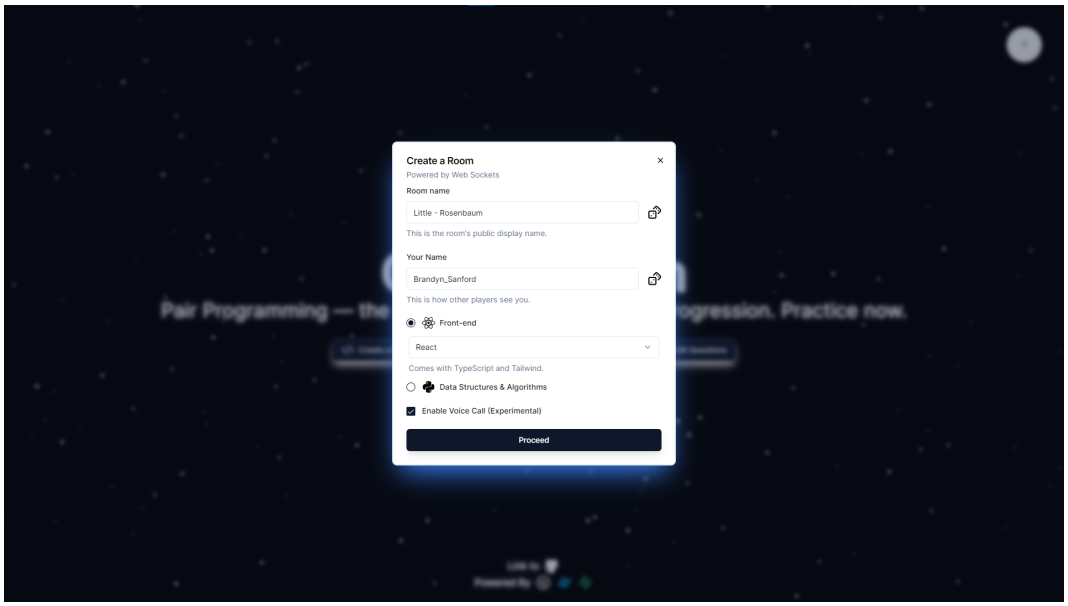


Figure 8.2: Create room pop-up

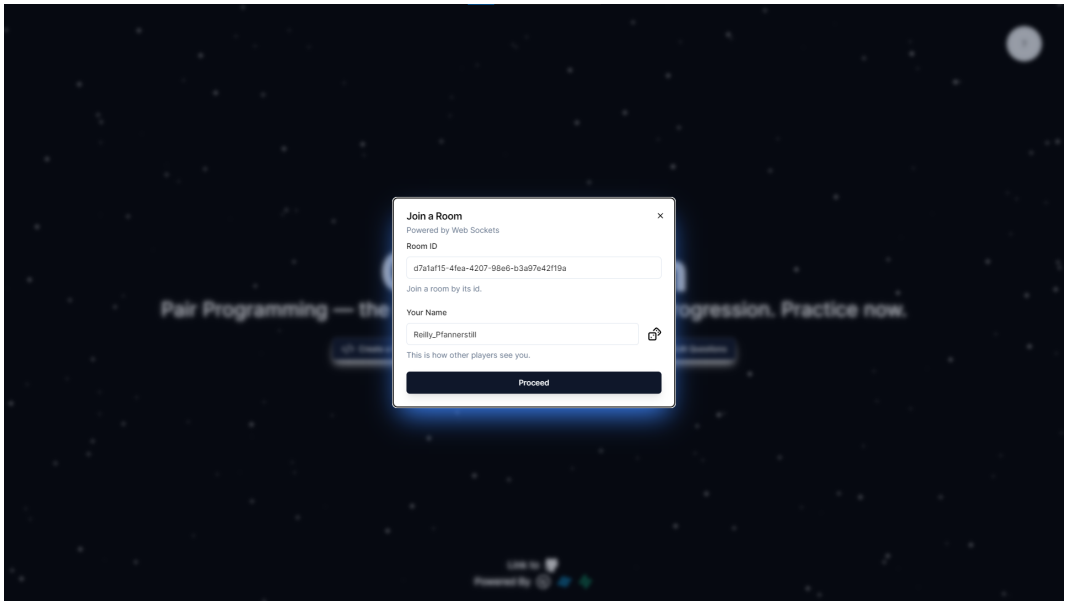


Figure 8.3: Join room by ID pop-up

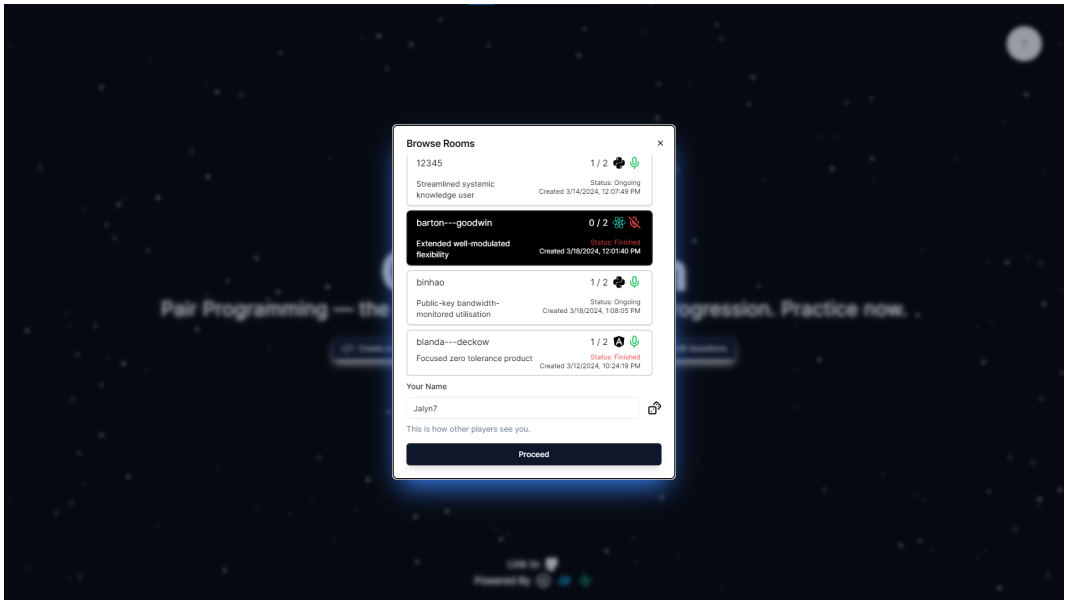


Figure 8.4: Browse rooms pop-up

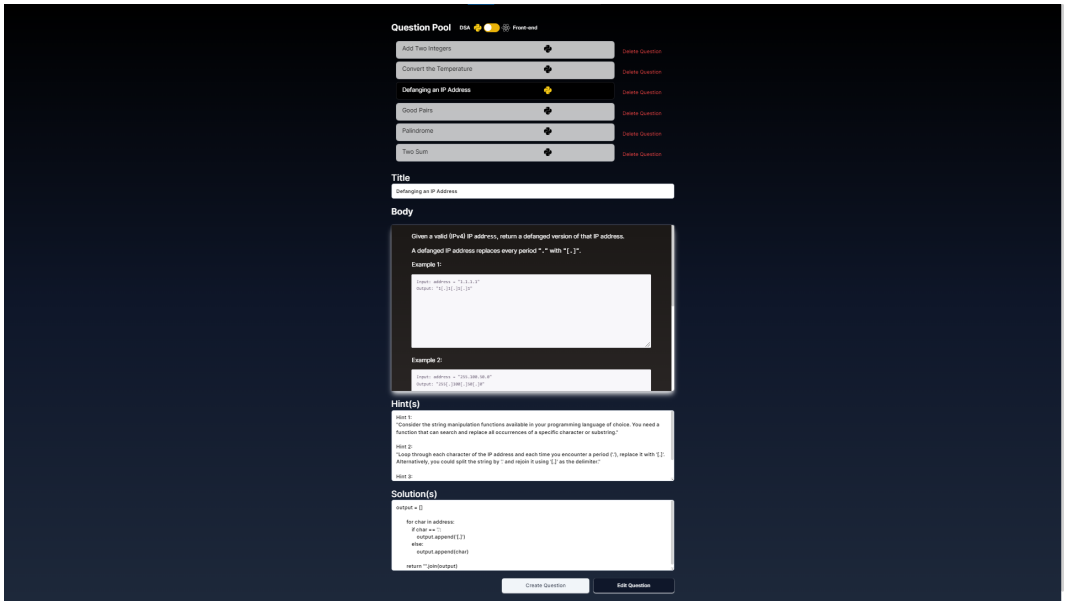


Figure 8.5: *Questions pool page*

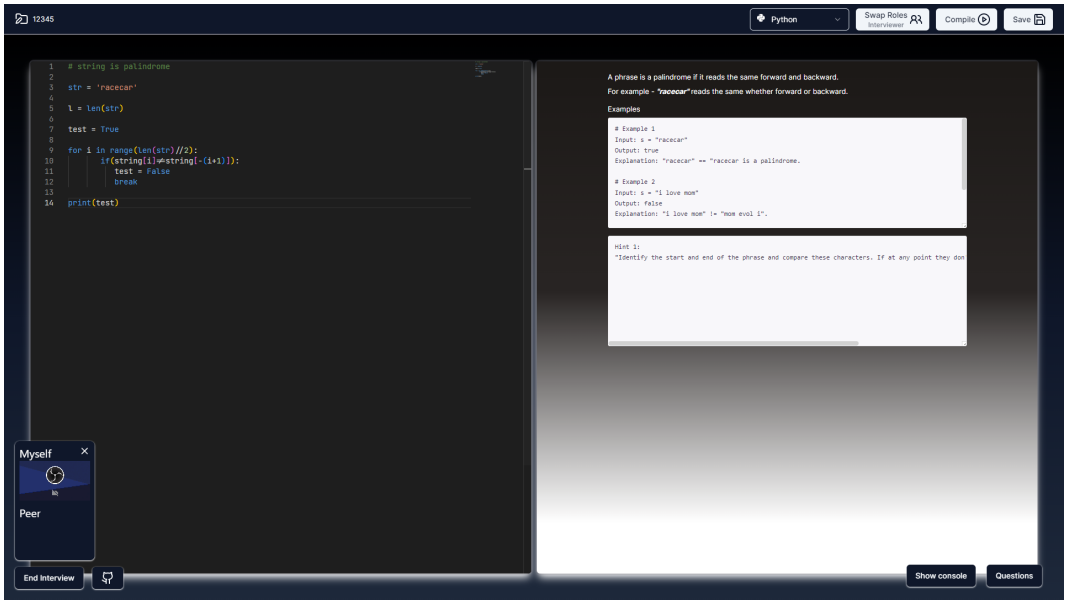


Figure 8.6: *Coding page (algorithmic)*

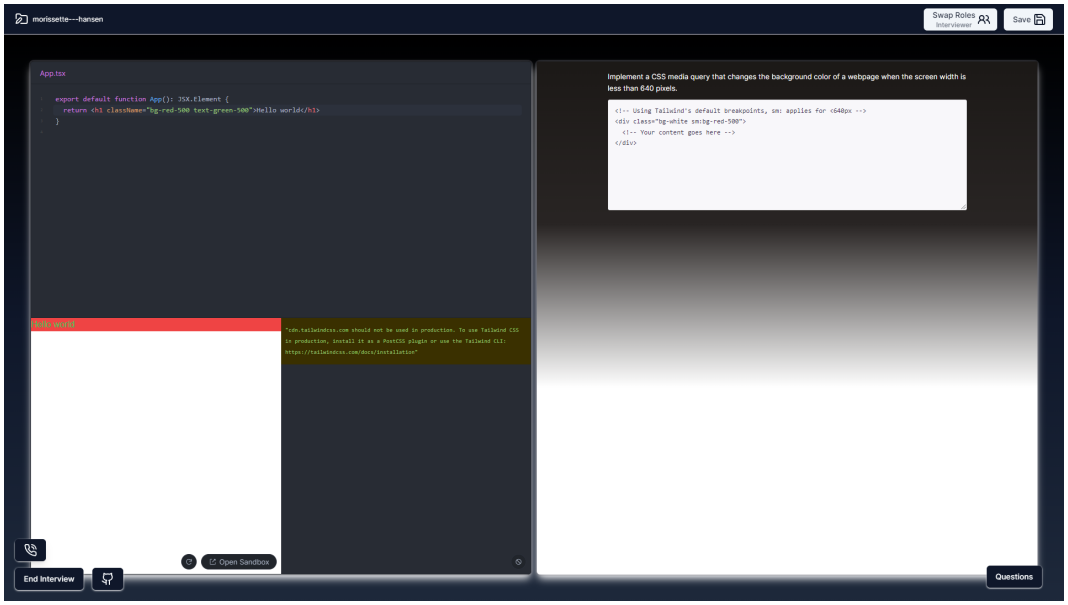


Figure 8.7: Coding page (front-end)

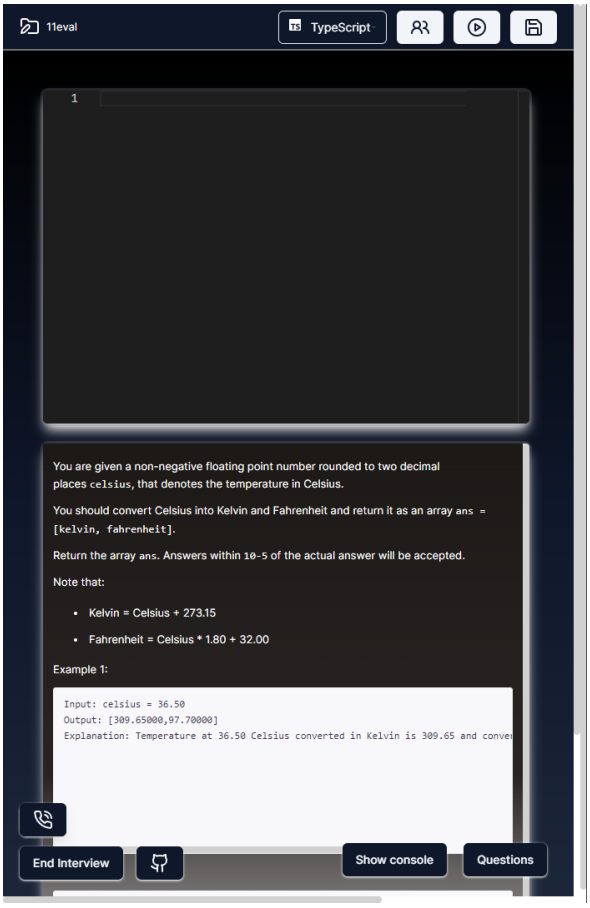


Figure 8.8: Coding page (algorithmic), mobile view

8.2 Ethics Checklist

School of Computing Science
University of Glasgow

Ethics checklist form for assessed exercises (at all levels)

This form is only applicable for assessed exercises that use other people ('participants') for the collection of information, typically in getting comments about a system or a system design, or getting information about how a system could be used, or evaluating a working system.

If no other people have been involved in the collection of information, then you do not need to complete this form.

If your evaluation does not comply with any one or more of the points below, please contact the Chair of the School of Computing Science Ethics Committee (matthew.chalmers@glasgow.ac.uk) for advice.

If your evaluation does comply with all the points below, please sign this form and submit it with your assessed work.

---

1. Participants were not exposed to any risks greater than those encountered in their normal working life.

   Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that occur outside usual laboratory areas, or that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, that use sensory deprivation (e.g. ear plugs or blindfolds), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback

2. The experimental materials were paper-based, or comprised software running on standard hardware.

   Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, laptops, iPads, mobile phones and common hand-held devices is considered non-standard.

3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project.

   If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, or the data is to be published), then signed consent is necessary. A separate consent form should be signed by each participant.

   Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script.

4. No incentives were offered to the participants.

   The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

Figure 8.9: *First half of signed ethics checklist form*

5. No information about the evaluation or materials was intentionally withheld from the participants.
   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

6. No participant was under the age of 16.
   Parental consent is required for participants under the age of 16.

7. No participant has an impairment that may limit their understanding or communication.
   Additional consent is required for participants with impairments.

8. Neither I nor my supervisor is in a position of authority or influence over any of the participants.
   A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. All participants were informed that they could withdraw at any time.
   All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script.

10. All participants have been informed of my contact details.
    All participants must be able to contact the investigator after the investigation. They should be given the details of both student and module co-ordinator or supervisor as part of the debriefing.

11. The evaluation was discussed with all the participants at the end of the session, and all participants had the opportunity to ask questions.
    The student must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation. In cases where remote participants may withdraw from the experiment early and it is not possible to debrief them, the fact that doing so will result in their not being debriefed should be mentioned in the introductory text.

12. All the data collected from the participants is stored in an anonymous form.
    All participant data (hard-copy and soft-copy) should be stored securely, and in anonymous form.

Course and Assessment Name __Level 4 Individual Project (Single Hons)_

Student's Name ___Naral Chalermchaikosol__

Student Number __2783720c_

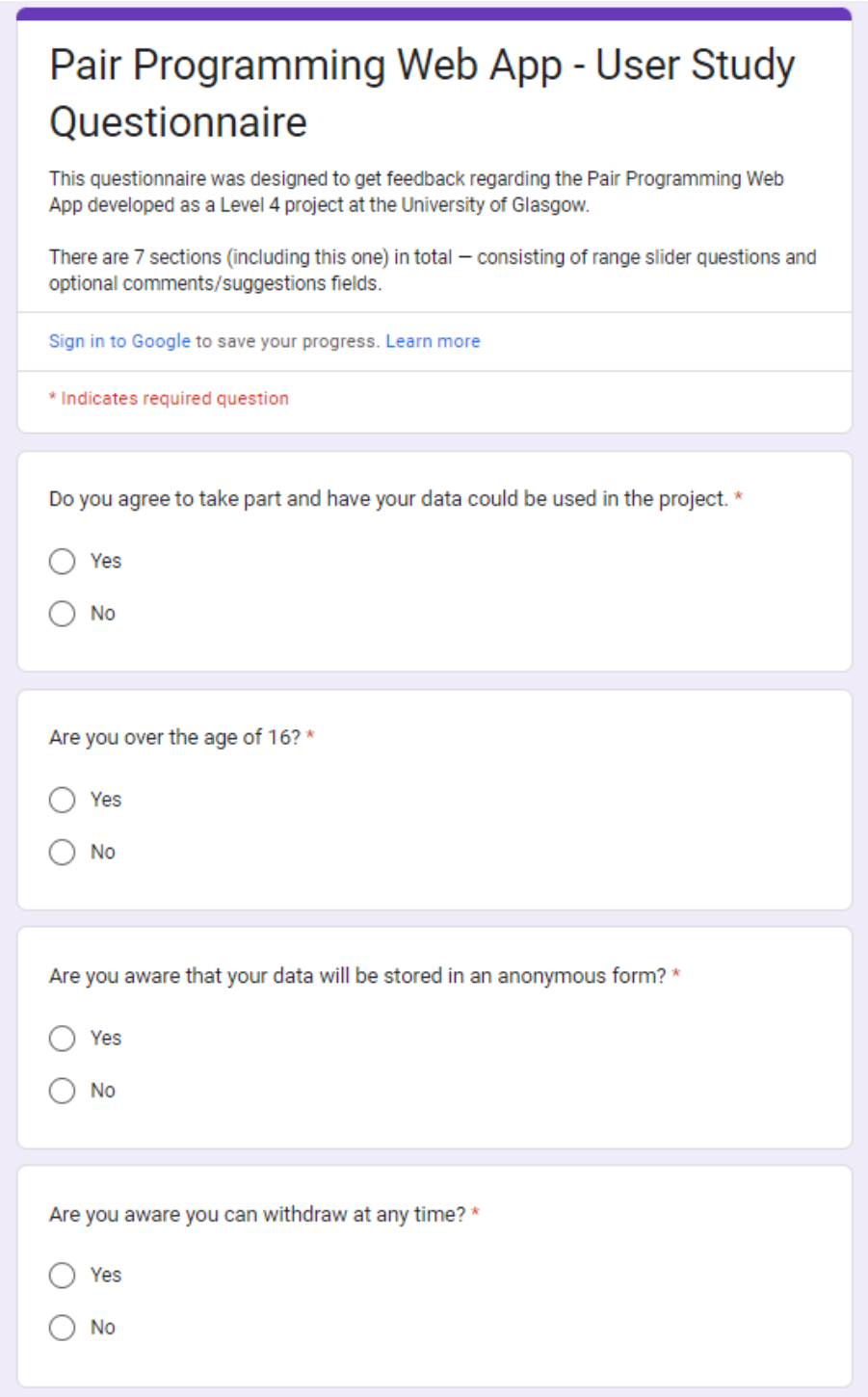
Student's Signature ___Naral C._

Date __18/03/2024___

Ethics checklist for assessed exercises

Figure 8.10: *Second half of signed ethics checklist form*

8.3 User Acceptance Questionnaire



Pair Programming Web App - User Study Questionnaire

This questionnaire was designed to get feedback regarding the Pair Programming Web App developed as a Level 4 project at the University of Glasgow.

There are 7 sections (including this one) in total — consisting of range slider questions and optional comments/suggestions fields.

Sign in to Google to save your progress. Learn more

* Indicates required question

Do you agree to take part and have your data could be used in the project. *

- Yes
- No

Are you over the age of 16? *

- Yes
- No

Are you aware that your data will be stored in an anonymous form? *

- Yes
- No

Are you aware you can withdraw at any time? *

- Yes
- No

Figure 8.11: *Evaluation questionnaire, 1/8*

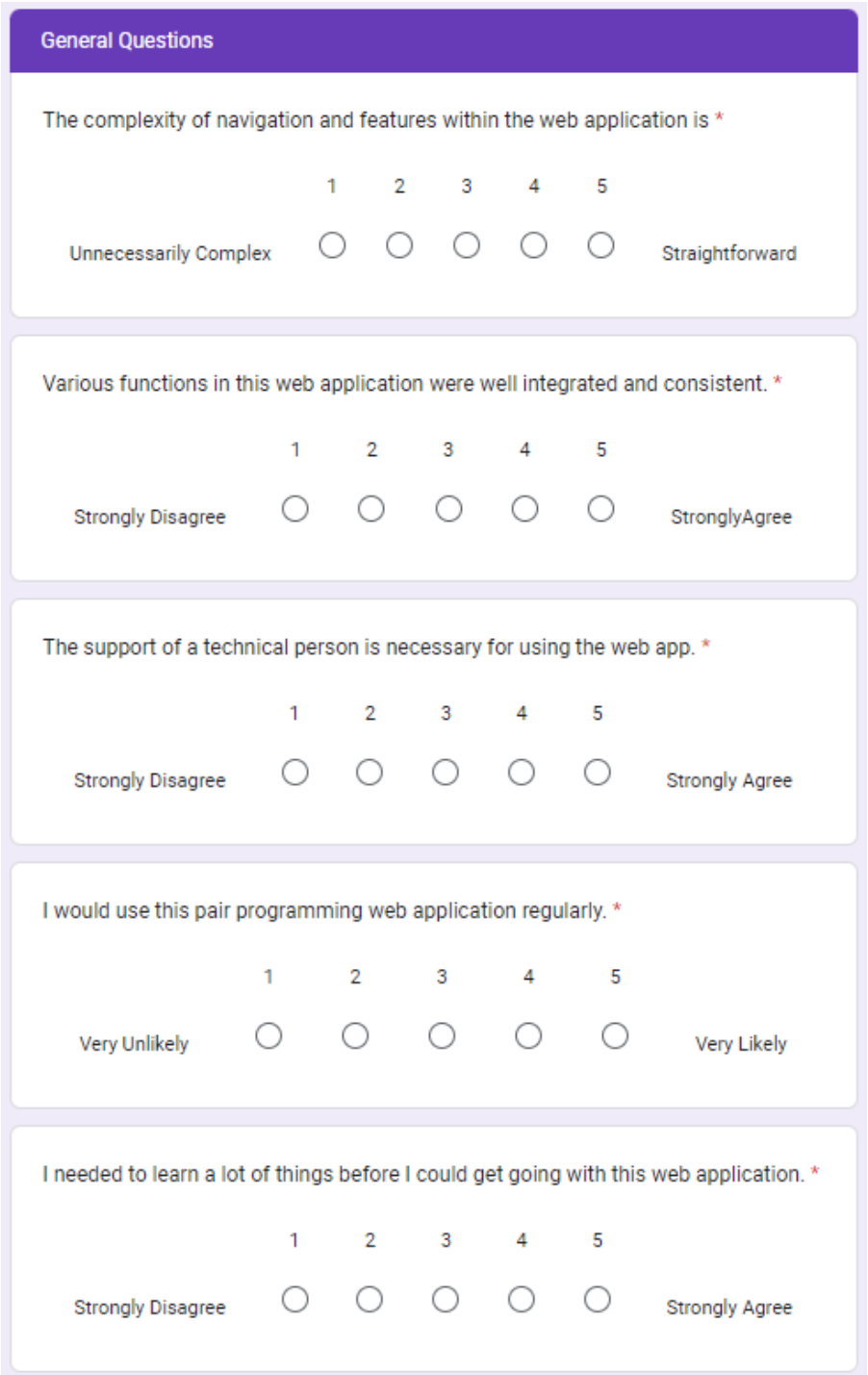


Figure 8.12: *Evaluation questionnaire, 2/8*

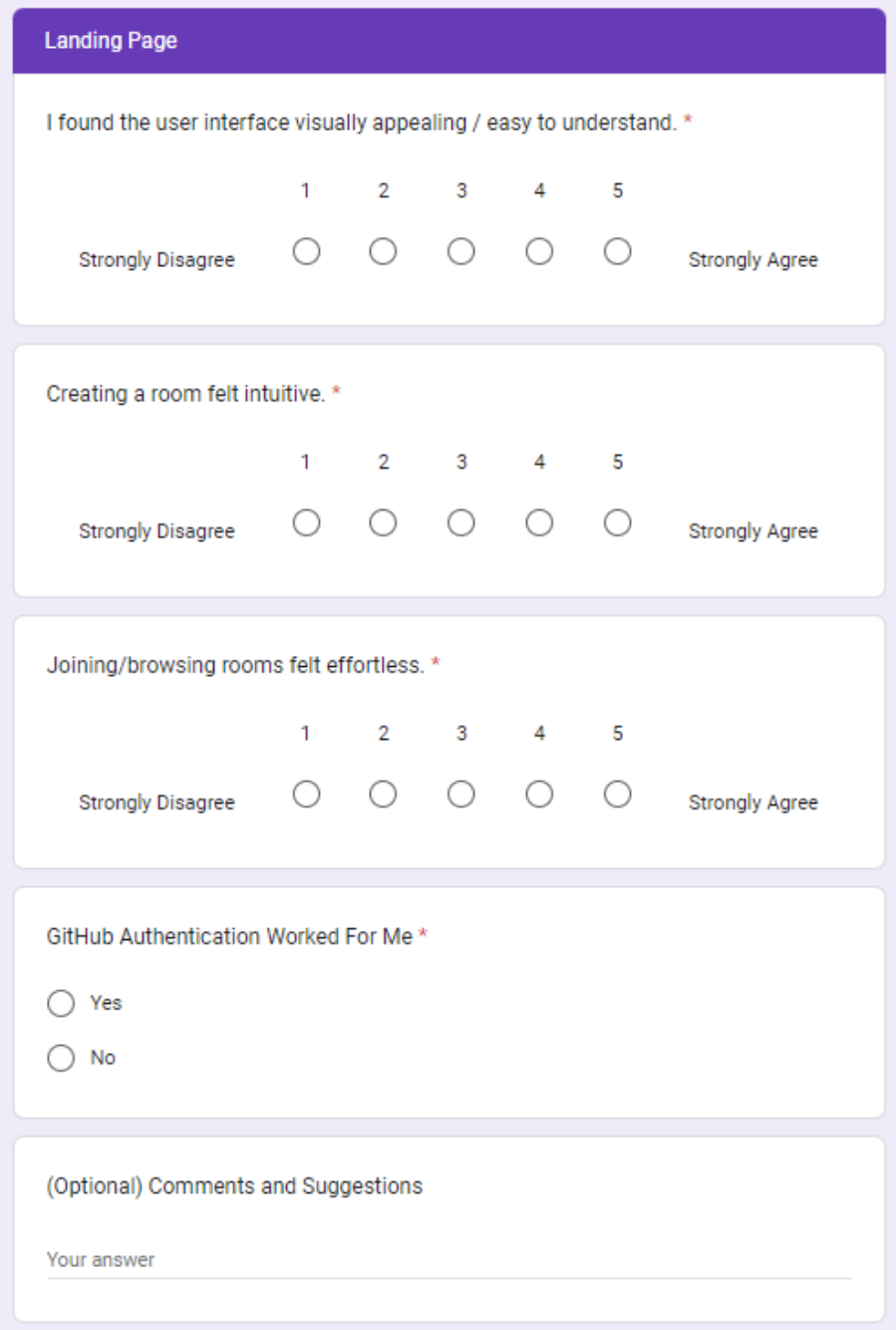


Figure 8.13: *Evaluation questionnaire, 3/8*

Figure 8.14: *Evaluation questionnaire, 4/8*

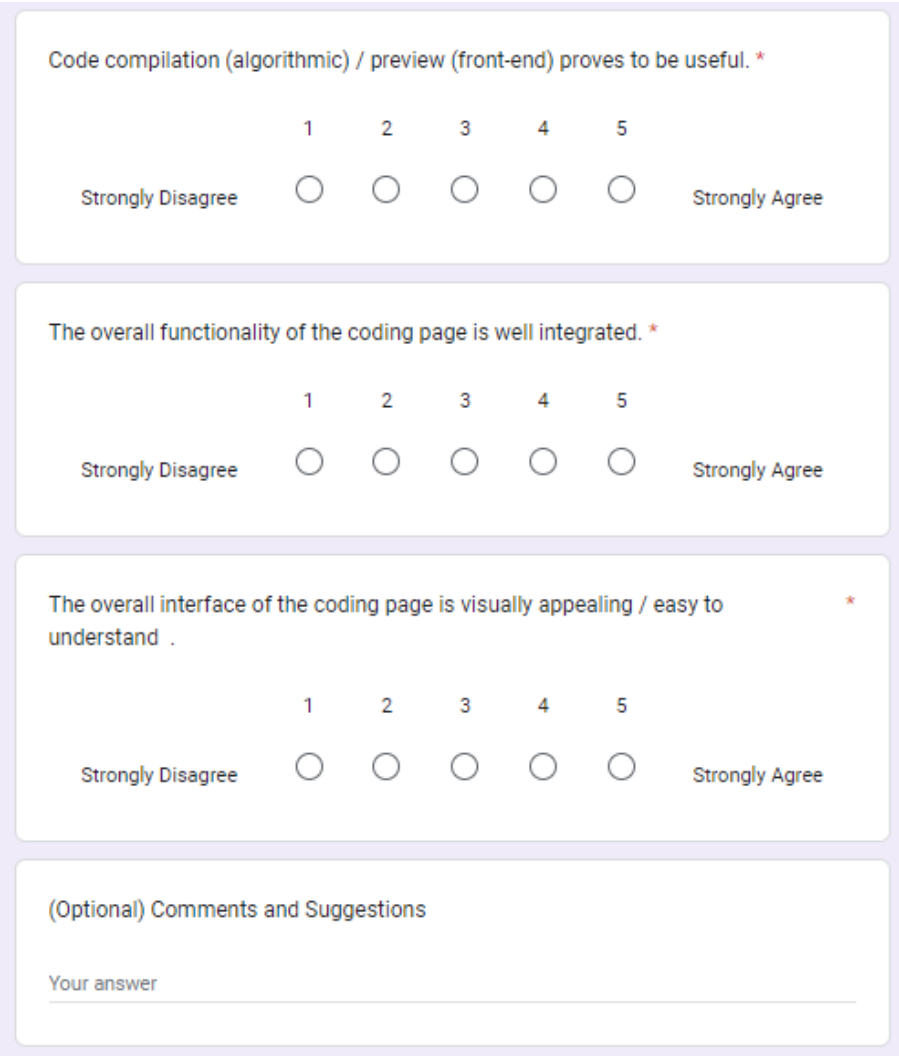


Figure 8.15: *Evaluation questionnaire, 5/8*

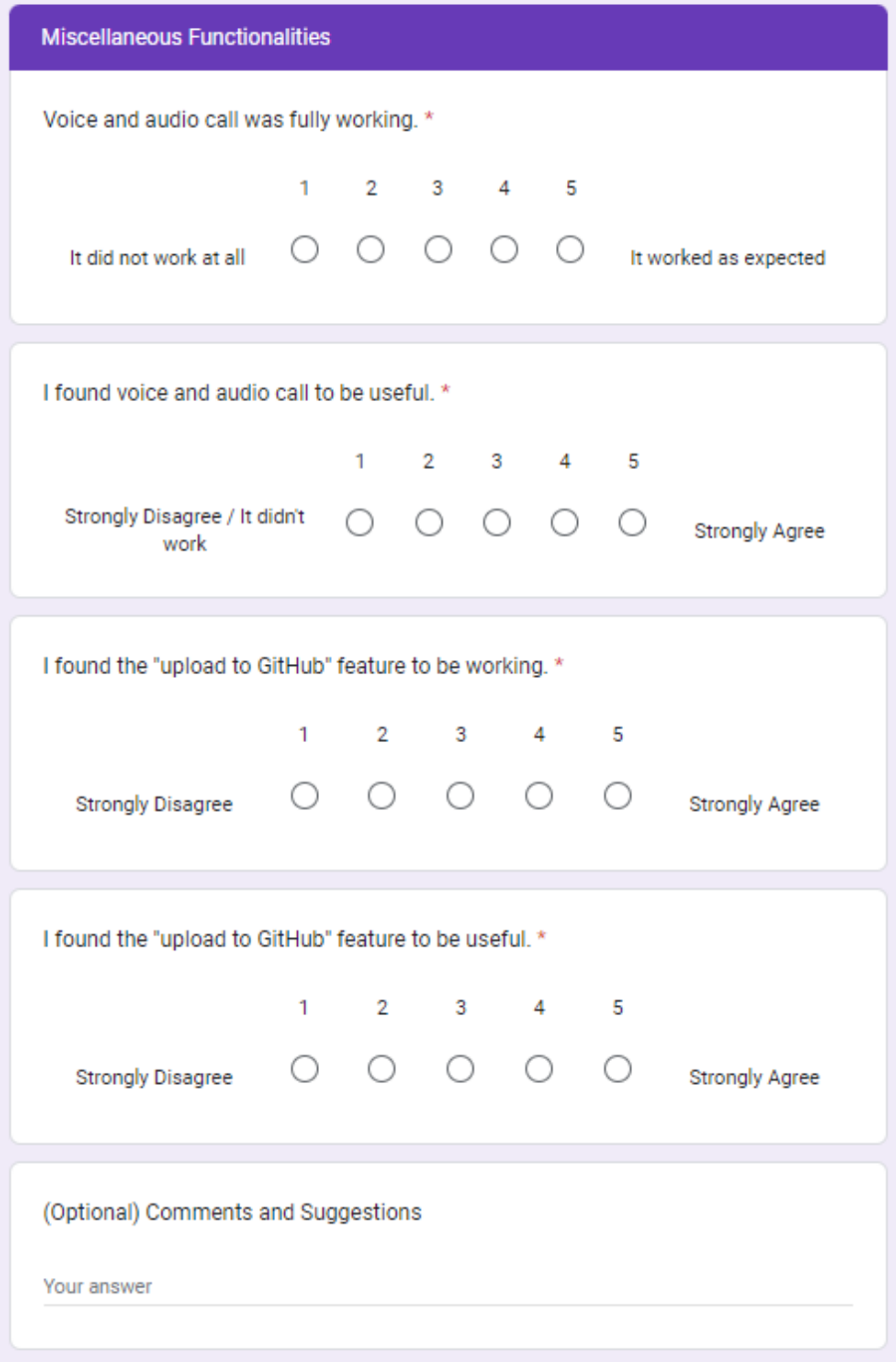


Figure 8.16: *Evaluation questionnaire, 6/8*

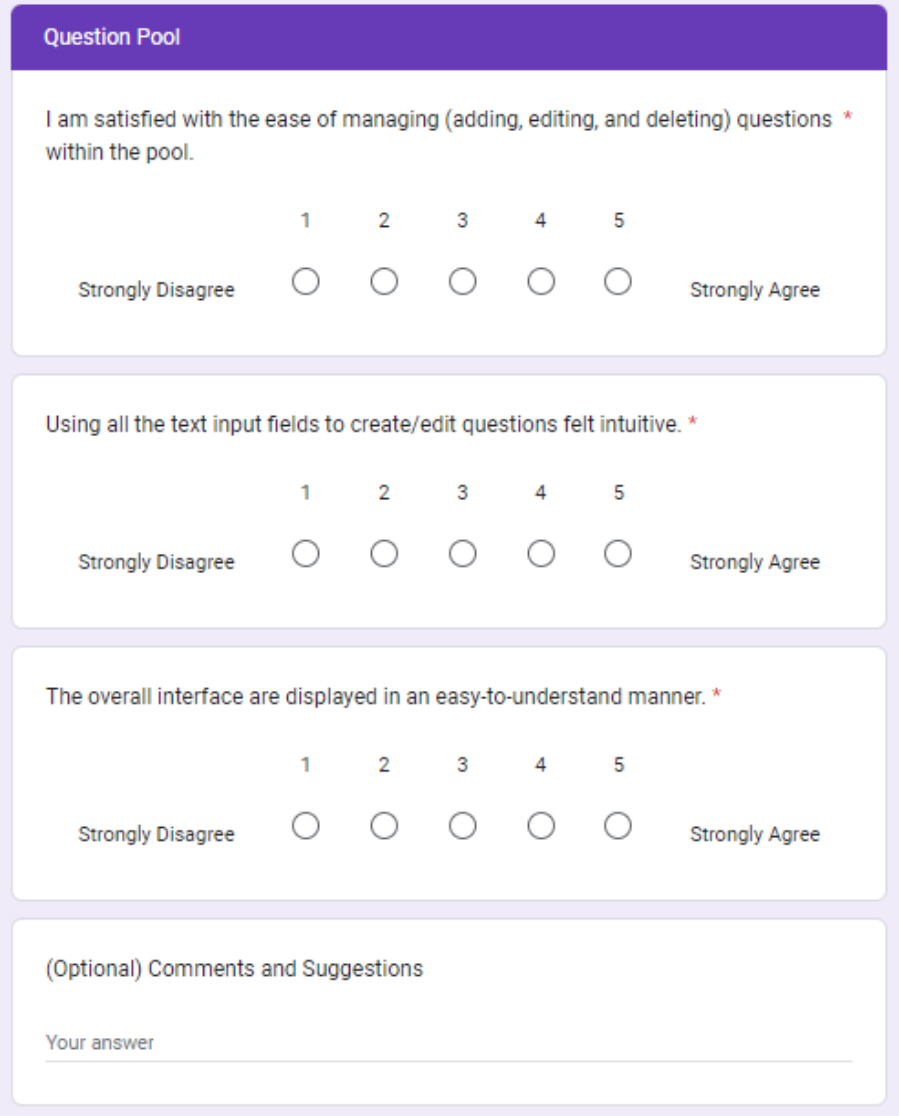


Figure 8.17: *Evaluation questionnaire, 7/8*

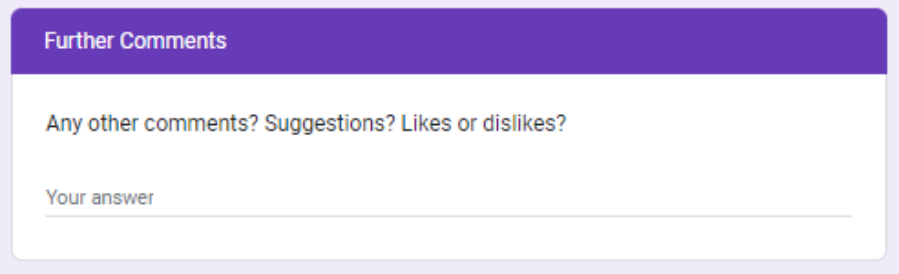


Figure 8.18: *Evaluation questionnaire, 8/8*

8.4 Requirements Satisfaction

Real time func. = Real time functionalities

Importance	Name	Met?	Met By
MH	Send and receive code from the peer	Yes	Real-time func.
MH	Have a text editor for coding languages	Yes	Monaco/Sandpack editors
MH	Compile their code and see the results	Yes	Judge0
MH	See a live preview of their work (front-end)	Yes	Sandpack editor
MH	Save the current code/note state to DB	Yes	CRUD on room data
MH	Have a shared note editor	Yes	Shared note editor
MH	Communicate via provided voice/video call	Yes	WebRTC implementation
MH	Swap roles (interviewer, interviewee) with their peer	Yes	Real-time func.
SH	Create room while specifying type and voice call	Yes	CRUD on room data
SH	Create front-end while specifying its framework	Yes	CRUD on questions pool
SH	Join room by ID	Yes	CRUD on room data
SH	Fetch progress from DB/peer when joining room	Yes	CRUD on room data/real-time
SH	Allow language switching (algorithmic)	Yes	Monaco editor/Zustand state management
SH	Peer cursor	Yes	Real-time func.
SH	End interview/session	Yes	CRUD on room data/real-time
SH	CRUD on questions pool	Yes	CRUD on questions pool
SH	GitHub auth & upload to repo	Yes	GitHub integration/Supabase
CH	Switch off mic/camera in voice call	No	Not met
CH	Edit room name	Yes	CRUD on room data/real-time
CH	Share images, links, videos on note editor	No	Not met
WH	Logged in users can see room history	No	Out of scope
WH	Logged in users create private pool of questions	No	Out of scope
WH	Questions test cases	No	Out of scope
WH	Higher room capacity (with voice call)	No	Out of scope

Table 8.1: *Check list of functional requirements, in the same order as the requirements page*

Importance	Name	Met?	Met By
MH	See feedback/metrics when code is compiled (algorithmic)	Yes	Judge0/UI design
MH	Allow up to two peers in the same room	Yes	WebRTC implementation
MH	Indicators when making network requests	Yes	Real-time func./UI design
SH	Animation for landing page	Yes	UI design
SH	Responsive on desktop and mobile	Yes	UI design
SH	Allow addition of coding languages/frameworks in the future	Yes	Judge0/Sandpack editor
SH	Work on modern browsers	Yes	End-to-end testing
SH	Intuitive for beginners	Yes	User evaluation
SH	Perform well with quick responses	Yes	User evaluation

Table 8.2: *Check list of nonfunctional requirements, in the same order as the requirements page*

8 Bibliography

[1] Bobby Law Brendan Murphy Janet Hughes, Ann Walshe. Remote pair programming. https://researchonline.gcu.ac.uk/en/publications/remote-pair-programming, 2020. Accessed: 2024-03-18.

[2] Brian Hanks and Others. Pair programming in education: A literature review. https://eric.ed.gov/?id=EJ929272, 2011. Accessed: 2024-03-18.

[3] InterviewStreet Inc. Hackerrank website. https://www.hackerrank.com/, 2024. Accessed: 2024-03-18.

[4] LLC. LeetCode. Leetcode website. https://leetcode.com/, 2024. Accessed: 2024-03-18.

[5] CodeSignal Inc. Codesignal website. https://codesignal.com/, 2024. Accessed: 2024-03-18.

[6] Inc. Replit. Replit website. https://replit.com/, 2024. Accessed: 2024-03-18.

[7] CODILITY LIMITED. Codility website. https://www.codility.com/, 2024. Accessed: 2024-03-18.

[8] Hina Tahir Shaista Khan Khadija Sania Ahmad, Nazia Ahmad. A fuzzy based moscow method for the prioritization of software requirements. https://ieeexplore.ieee.org/abstract/document/8342602, 2018. Accessed: 2024-03-18.

[9] L. Sha X. Liu, J. Heo. Modeling 3-tiered web applications. https://ieeexplore.ieee.org/abstract/document/1521145, 2005. Accessed: 2024-03-18.

[10] Bradford G. Nickerson Victoria Pimentel. Communicating and displaying real-time data with websocket. https://ieeexplore.ieee.org/abstract/document/6197172, 2012. Accessed: 2024-03-18.

[11] Dawid Karczewski. The main benefits of prototyping in software engineering. https://www.ideamotive.co/blog/the-main-benefits-of-prototyping-in-software-engineering, 2020. Accessed: 2024-03-18.

[12] State of JS. State of js 2022: Front-end frameworks. https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/, 2022. Accessed: 2024-03-18.

[13] Priyangi Singh Archana Bhalla, Shivangi Garg. Present day web development using react.js. https://www.academia.edu/download/64560056/IRJET-V7I5223.pdf, 2020. Accessed: 2024-03-18.

[14] Marzieh SOMI. User interface development of a modern web application. https://webthesis.biblio.polito.it/secure/30076/1/tesi.pdf, 2021. Accessed: 2024-03-18.

[15] Manoj Kumar. Serverless architectures review, future trend and the solutions to open problems. https://www.researchgate.net/profile/Manoj-Kumar-318/publication/331729868_Serverless_Architectures_Review_Future_Trend_and_the_Solutions_to_Open_Problems/links/5c984797a6fdccd460384edb/Serverless-Architectures-Review-Future-Trend-and-the-Solutions-to-Open-Problems.pdf, 2019. Accessed: 2024-03-18.

[16] Dragan Posarac Branislav Sredojev, Dragan Samardzija. Webrtc technology overview and signaling solution design and implementation. https://ieeexplore.ieee.org/abstract/document/7160422, 2015. Accessed: 2024-03-18.

[17] Cypress.io. Cypress trade-offs. https://docs.cypress.io/guides/references/trade-offs, 2023. Accessed: 2024-03-18.

[18] Alejandro Alfredo Monje Morales. Automated front-end website testing with cypress. https://www.theseus.fi/handle/10024/806779, 2023. Accessed: 2024-03-18.

[19] Frank Fransen Fatih Turkmen Thorsten Rangnau, Remco v. Buijtenen. Continuous security testing: A case study on integrating dynamic security testing tools in ci/cd pipelines. https://ieeexplore.ieee.org/abstract/document/9233212, 2020. Accessed: 2024-03-18.

[20] Timo McFarlane. Managing state in react applications with redux. https://www.theseus.fi/bitstream/handle/10024/265492/McFarlane_Timo.pdf, 2019. Accessed: 2024-03-18.

[21] Matthias Kevin Caspers. Rich internet applications — react and redux. https://uol.de/f/2/dept/informatik/ag/svs/download/reader/reader-seminar-ws2016.pdf#page=14, 2017. Accessed: 2024-03-18.

[22] Mozilla Web Docs. Client-side form validation. https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation, 2024. Accessed: 2024-03-18.

[23] Mozilla Web Docs. How to structure a web form. https://developer.mozilla.org/en-US/docs/Learn/Forms/How_to_structure_a_web_form, 2024. Accessed: 2024-03-18.

[24] Michael J C Gordon. Programming language theory and its implementation. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=e94334412e47eb35ebb418037f7279a61021151a, 1988. Accessed: 2024-03-18.

[25] David J. Malan. Cs50 sandbox: secure execution of untrusted code. https://dl.acm.org/doi/abs/10.1145/2445196.2445242, 2013. Accessed: 2024-03-18.

[26] Marc Prud'hommeaux Patrick Connor Linskey. An in-depth look at the architecture of an object/relational mapper. https://dl.acm.org/doi/abs/10.1145/1247480.1247581, 2007. Accessed: 2024-03-18.

[27] Sushrit Pasupuleti. How, when, and why you should switch from vercel to a different hosting provider (especially for next.js). https://medium.com/@sushrit.pk21/how-when-and-why-you-should-switch-from-vercel-to-a-different-hosting-provider 2022. Accessed: 2024-03-18.

[28] Aishwarya Anand. Managing infrastructure in amazon using ec2, cloudwatch, ebs, iam and cloudfront. https://www.academia.edu/download/59534122/managing-infrastructure-in-amazon-using-ec2-cloudwatch-ebs-iam-and-cloudfront-l pdf, 2017. Accessed: 2024-03-18.

[29] John Brooke. Sus: A retrospective. https://uxpajournal.org/wp-content/uploads/sites/7/pdf/JUS_Brooke_February_2013.pdf, 2013. Accessed: 2024-03-18.

[30] justjavac. Vscode live share . https://zhuanlan.zhihu.com/p/36566121, 2018. Accessed: 2024-03-18.

[31] Coding With Chaim. Going beyond peer to peer with webrtc (mesh, mcu, sfu). https://www.youtube.com/watch?v=V9g4MYtCHkY, 2021. Accessed: 2024-03-18.