

Higher Diploma
In
Computer Science



Final Project:

“Data-Heist”

Gamified Revision Tool for Cybersecurity
using a full stack web application.

By

Seán Kervick

Final Project

Academic Year 2025

South-East Technological University,
Ireland.

Declaration of Ownership

I, the author of this document, hereby declare that the information provided in this report is my own work and does not contain any plagiarism. Any content, illustrations, or material that has been reproduced or copied is clearly referenced by italicizing the text and including the corresponding reference number. More detail of the referenced information can be found in the bibliography at the end of this document.

Contents

1. Introduction.....	5
1.1 Objectives.....	5
1.2 Motivation.....	5
2. Research.....	5
2.1 Existing cybersecurity learning tools.....	5
2.2 Technology Choices	6
2.2.1 Frontend.....	6
2.2.2 Backend.....	6
2.2.3 Database	7
2.2.4 Deployment	7
3. Analysis & Design.....	8
3.1 Game Design Overview	8
3.1.1 Option 1 - Storyline:.....	8
3.1.2 Option 2 - Quiz:	8
3.1.3 Option 3 - CTF:.....	8
3.2 User Flow & Features	8
3.3 System Architecure	9
3.3.1 Frontend:.....	9
3.3.2 Backend:.....	10
3.4 Easy Mode Challenges	10
3.4.1 Password Cracker	10
3.4.2 Spot the Phish	11
3.4.3 Quiz Round.....	11
3.5 Hard Mode Challenges	11
3.5.1 Code Exposed.....	11
3.5.2 Cert Inspector	11
3.5.3 The Port Watcher.....	11
4. Plan – Project Timeline:.....	12
5. Development.....	12
5.1 Backend & MongoDB Setup.....	12
5.2 User Authentication	13
5.3 Vite + React Frontend Setup	14
5.4 Challenges	15
5.4.1 Password Cracker	15
5.4.2 Spot the Phish	17

5.4.3 Quiz Round.....	17
5.4.4 Code Exposed.....	19
5.4.5 Cert Inspector.....	19
5.4.6 The Port Watcher.....	20
5.5 Timer System	21
5.6 Points Allocation, Leaderboard & Hints	22
5.7 Deployment.....	24
6. Testing.....	24
7. Conclusion.....	25
8. AI Usage	26
8. Bibliography.....	27
9. Appendix.....	28
A – backend/src/connect.js	28
B – backend/src/utils/generateToken.js.....	28
C – backend/src/accountsController.js.....	29
D – backend/src/middleware/jwt.js.....	30
E – client/src/components/PrivateRoute.tsx	31
F – client/src/pages/SignUpPage.tsx	31
G – client/src/pages/challenges/SpotThePhishPage.tsx	32
H – client/src/components/QuizQuestions.tsx.....	32
I – client/src/pages/challenges/QuizRoundPage.tsx.....	33
J – client/src/pages/challenges/ThePortWatcherPage.tsx	34
K – client/src/components/Timer.tsx	35
L – backend/src/scoreController.js	36
M – client/src/api/scoreController.tsx	37

1. Introduction

“Data Heist” is a gamified learning/revision tool for cybersecurity, using a full stack web application. In this gamified application, players use their knowledge of cybersecurity through practical and realistic challenges to progress, gain points, and earn a place on the leaderboard.

1.1 Objectives

- Develop a fun learning tool for the fundamentals of cyber security.
- Gain experience of multiple modules such as Full-Stack Web Development, Databases, and Security.
- Combine aspects of existing cybersecurity tools to create a fun, interactive user experience for both beginners and IT enthusiasts.

1.2 Motivation

The motivation behind the project came from a newly developed interest in cybersecurity since completing the security module of the HDip in Computer Science. By using a full stack web application, many of the skills learned over the past two years will be incorporated into the project and by basing the theme of the application on cybersecurity, it will be an enjoyable challenge.

2. Research

During the proposal meeting it was advised that a call with Jimmy, one of the security module lecturers, would be beneficial for brainstorming ideas for the game design and challenges. During that call, on 29/11/24, Jimmy suggested the ‘Capture the Flag’ concept and also provided some deliberating vulnerable web applications. This began the research phase, and the following findings were the basis of inspiration for the finalized game design and technologies used.

2.1 Existing cybersecurity learning tools

- [HackTheBox.com](https://www.hackthebox.com) – At a first glance seems quite complex, simulating real-world problems using virtual machines with vulnerabilities as well as challenges that explore various penetration testing techniques. A more intermediate to advanced learning tool for cybersecurity.
- [TryHackMe.com](https://www.tryhackme.com) – Definitely a more beginner-friendly tool with introductions to operating systems and step by step exercises in every lesson, similar style to that of [freeCodeCamp.org](https://www.freecodecamp.org), for a hands on approach to learning cybersecurity from scratch. Something similar to the structure of these exercises may be an option for the project.
- [GameOfCybersecurity](https://github.com/0x00sec/0x00sec) (GitHub) – ‘*The project consists in developing an online game, with an open-source code, based on a traditional board game, with the objective of improving the level of maturity of the players in cybersecurity.*’¹ A quiz-like game that presents

players with cybersecurity based questions that they must answer within a time limit.

- [DVWA: Damn Vulnerable Web Application](#) (GitHub) – A deliberately insecure application designed for ethical hackers or cybersecurity enthusiasts to practice penetration testing and learn about common vulnerabilities. A very interesting Repo and appears to be a great learning tool that may provide some inspiration for some challenge ideas.

2.2 Technology Choices

2.2.1 Frontend

React.js has been chosen mainly due to the past experience gained during the full-stack-2 module. Its large community of developers was another key factor as it should be relatively easier to find resources for help and support, making the development process less difficult.

HTML/CSS was considered but it's static-like nature is not deemed suitable for this project's interactive UI.

Vue.js was a new technology explored that did show potential. It may be a suitable option but was not selected due to the lack of experience and smaller community, which would potentially impact the speed of development.

	React.js	HTML/CSS	Vue.js
Pros	<ul style="list-style-type: none"> • Component-based • Widely used (strong community) • Suitable for interactive UI • Makes interface building more "seamless".² • Scalable³ 	<ul style="list-style-type: none"> • Easy to use & understand • Great for static websites 	<ul style="list-style-type: none"> • Simple syntax • Flexible & lightweight • Easy to learn with simple integration⁵
Cons	<ul style="list-style-type: none"> • Users must be familiar with JavaScript • Can be a difficult to library to grasp for beginners. 	<ul style="list-style-type: none"> • Not suitable for scaling • 'Cross-Browser Compatibility Issues'⁴ • Not designed for dynamic updates 	<ul style="list-style-type: none"> • Smaller community compared to React. • Lack of plugins & scalability⁶

2.2.2 Backend

It was decided that **Node.js** alongside **Express.js** would be the doing the back-end heavy lifting rather than Node & Hapy because, although with prior experience of using both, Express seemed to be the more user friendly framework to work with and appears to have a larger community which should make facing challenges easier.

With an interest to further explore Docker after the security module, it was heavily considered for its library of pre-built images which may help to quickly build up the technology stack, but it was deemed that it might add complexity to the learning curve, and with the given time frame, it was not selected.

Django was a newly explored technology, 'a free and open-source, Python-based web framework that runs on a web server.'⁷ Again, without the desire to increase the learning curve, Django

was not selected due to the lack of prior experience with the Python language. However, it does seem to have suitable attributes such as scalability, and built in tools for authentication, routing and security.

	Node.js & Express.js	Docker	Django
Pros	<ul style="list-style-type: none"> • Large npm ecosystem & community support. • Hassle free integration with front-end (React.js) 	<ul style="list-style-type: none"> • Large library of pre-built images 	<ul style="list-style-type: none"> • <i>ORM layer for handling database access, session, routing.</i>¹⁰ • Scalability
Cons	<ul style="list-style-type: none"> • <i>'Single threaded nature'</i>⁸ making it less suitable for tasks heavy on CPU 	<ul style="list-style-type: none"> • <i>Requires additional resources and configuration to run</i>⁹ • May add complexity 	<ul style="list-style-type: none"> • Added complexity • Seems like the framework would be excessive for this smaller project.

2.2.3 Database

With prior experience designing databases using MySQL workbench, a MySQL database was considered for the project. It would be interesting to design and build the database using the workbench app, generate scripts and potentially use them in the program.

MongoDB seems like the simplest option, in conjunction with Mongoose, a Node.js library, it's possible to create flexible schemas and easily structure the database for the project in JSON files. This application won't necessarily need a huge database with lots of strict data or relational schemas like in MySQL. A more flexible and dynamic database like MongoDB, which seamlessly ties in with the selected back-end framework Node.js, should be a good option for storing game related data like usernames and scoreboards.

Firebase was researched as another option for the database and could potentially be used, but one of its main use case advantages of multiple API access methods won't be a requirement for the app. It seems like it also could be costly in the event of the app scaling, which is unexpected, but this is a deterrent.

	MongoDB	Firebase	MySQL
Pros	<ul style="list-style-type: none"> • <i>Flexible document schemas</i>¹¹ • Seamless tie-in with Node.js 	<ul style="list-style-type: none"> • API support 	<ul style="list-style-type: none"> • Relational schemas • Secure
Cons	<ul style="list-style-type: none"> • Less suited to highly relational data • <i>'High reliance on good indexing'</i>¹² 	<ul style="list-style-type: none"> • Costly as app grows 	<ul style="list-style-type: none"> • Requires fixed schema – less flexible • Issues handling large query traffic

2.2.4 Deployment

'Vercel' was mentioned in the full-stack-2 module for deploying frontend web applications. While it wasn't utilized in the assignment by me personally at the time, some classmates stated that it is a straightforward and efficient option, that works well with React.

Since Vercel only seems to support *'backend logic via serverless function'*¹³, it is understood that the Node.js & Express backend will have to be deployed separately. With previous experience using '**Render**', it was decided that this will be the chosen solution for the backend.

3. Analysis & Design

3.1 Game Design Overview

Three options came to mind for the type of game:

3.1.1 Option 1 - Storyline:

The player takes on the role of a hacker character and attempts to breach an organisation's IT defence to steal sensitive data. This is done by completing a series of challenges presented to the player. The UI being cartoon pictures of characters, buildings, and rooms where images can be clicked on to find clues or complete challenges to advance to the next levels.

3.1.2 Option 2 - Quiz:

A quiz like format where the player is asked cybersecurity questions and must select the correct answers under time constraints to gain points and earn a spot on the leaderboard.

3.1.3 Option 3 - CTF:

Small individual 'Capture the Flag' challenges or hacking simulations without the cartoon UI, hacking character, or storyline. A revision tool with a relatively basic but interactive UI consisting of a mainly text-based interface along with some images where players must find vulnerabilities, exploit them, and/or use tools to complete tasks.

It's been decided that **option 3** will be best suited for the overall design of the project, keeping the UI simple while incorporating different types of interactive challenges for players to learn about cybersecurity.

3.2 User Flow & Features

Firstly, the player is presented with a home page, that has a hacker/matrix design of an all-black background, bright green text and a hacker-like font. The homepage will have options to sign up and create an account, login, or to play the game as a guest.

Having the option to play the game as guest gives players the ability to try it out without signing up, but their access to some extra features will be limited.

If the user creates an account and plays the game as a logged in player, they will be able to utilize more features such as saving game progress and may earn a place on the leaderboard.

Whether playing as a guest or a logged in player, users will have the two difficulty options:

- **Easy Mode:** For people that use computers at work or for personal reasons.
- **Hard Mode:** For students or 'tech savvy' individuals well-versed in cybersecurity or software development.

Once the difficulty mode is chosen, the player is presented with the first challenge. A dialog box pops up explaining the challenge and once the 'OK' button is pressed, they have a certain amount of time to complete it. The faster the challenge gets completed, the more points that will be allocated to the player.

At the end of each task, another dialog box will open, outlining the result and number of points received. An educational message that explains the reason behind the challenge will also appear to bring awareness of cybersecurity and the different types of attacks.

Initially, there will be 3 challenges for each difficulty level. At the end of the game, all players will be presented with their overall score and logged in players will be able to see where they stand on the leaderboard.

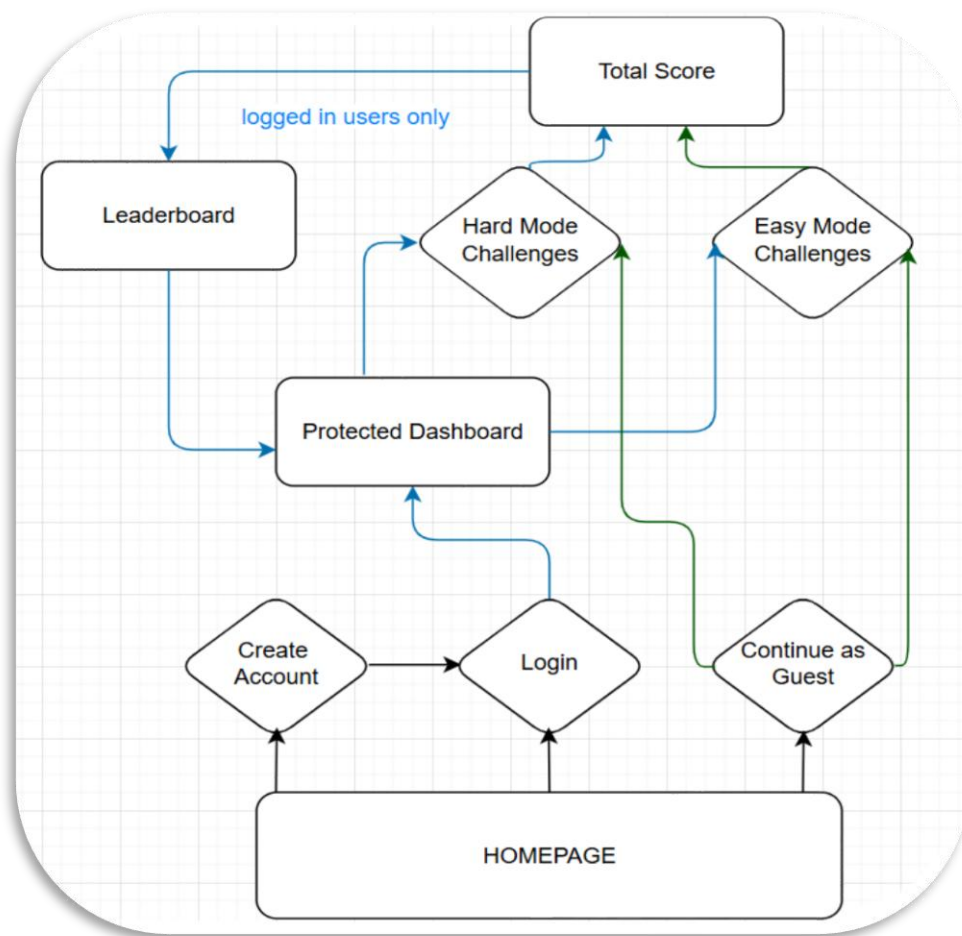


Figure 1: User Flow Diagram.

3.3 System Architecture

The finalised architecture designed for the project is as follows:

3.3.1 Frontend:

- **React.js**; a component based frontend framework, that will allow for a friendly development process for the interactive UI.
- **Vite**; 'a blazing fast frontend build tool'¹⁴ for the frontend development environment.
- **Material UI**; a React component library that provides pre-built UI components to speed up the process of building a modern-looking UI.
- **Typescript**; the chosen frontend programming language with interfaces and static type definitions for good code reliability.
- **React Router**; a React library for managing the routing and navigation between pages. It allows for a fast navigation without the need to refresh or re-loading pages.
- **Axios**; a JavaScript library simplifying the API communication between the frontend and backend.
- **Vercel**; is the chosen solution for frontend deployment.

3.3.2 Backend:

- **Node.js**; a JavaScript runtime environment to build a backend that handles API requests and to interact with the Mongo database.
- **Express.js**; a Node.js framework to handle the backend routing and middleware.
- **JSON Web Token (JWT) Authentication**; for user authentication and session management to authorize users and protect API routes giving logged in users access to more features than users playing as a guest.
- **Bcrypt**; a library for hashing & salting passwords to store user credentials securely.
- **Mongoose**; a Node library for MongoDB that simplifies database interactions with easily structured schemas for storing user credentials, and gaming data such as final scores, an overall leaderboard and saving progress.
- **MongoDB**; where all the above mentioned data will be persistently stored.
- **Render**; is the chosen solution to deploy the backend.

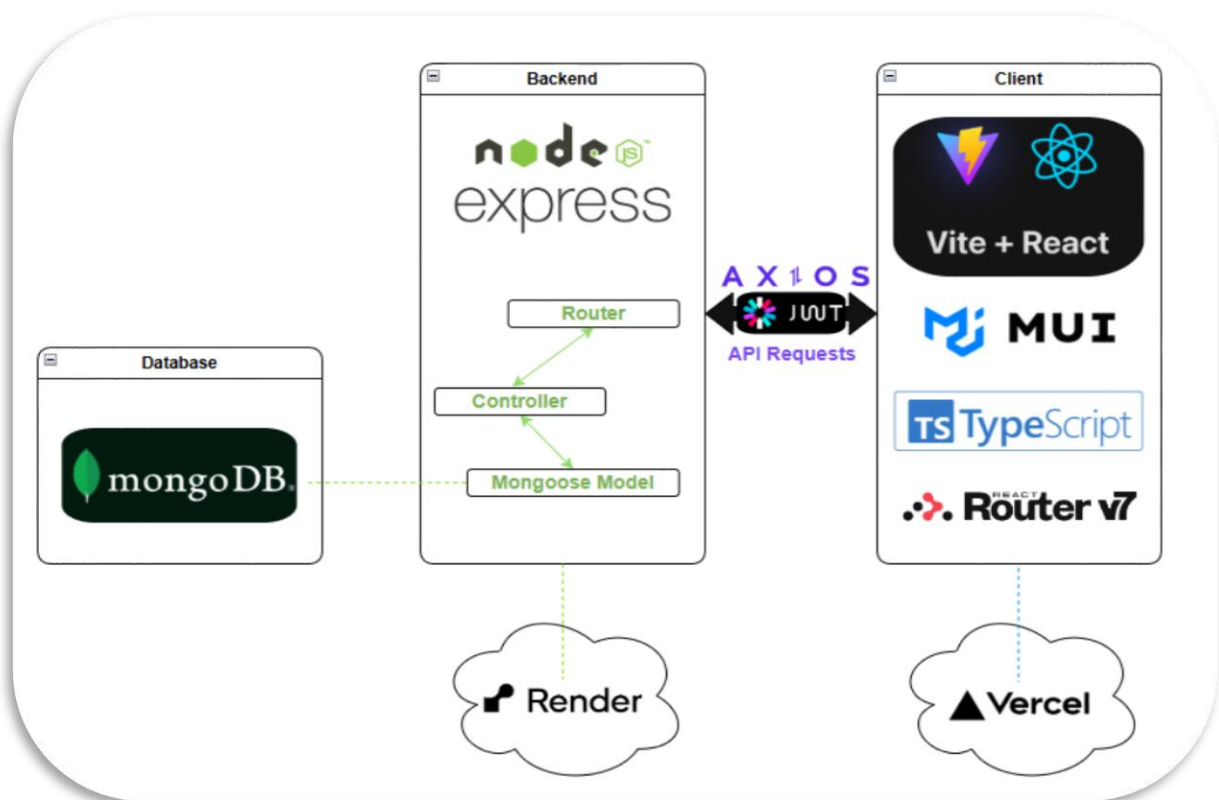


Figure 2: System Architecture Diagram.

3.4 Easy Mode Challenges

3.4.1 Password Cracker

The player is presented with a fake social media page with a picture and a bio. Clues to crack the password are within the information given in the profile. The player must guess the correct password before the clock runs out to move on to the next challenge.

This challenge will be relatively easy but highlights the importance of password strength and sharing sensitive information on social media.

3.4.2 Spot the Phish

The player is presented with a phishing email and have to find 3 ‘red flags’ that indicate the email is suspicious. The UI will consist of a replicated email that contains common phishing tactics such as urgent descriptions, links, and slight spelling mistakes in sender addresses. Once the 3 red flags are found, the challenge is over, and the number of points allocated depends on how fast it was completed. The task teaches different phishing tactics used by attackers and helps to recognize these types of threats online.

3.4.3 Quiz Round

A series of multiple choice, theory-based questions that test the player’s knowledge of cybersecurity fundamentals. Players must select the correct answer earning points for speed and knowledge. The challenge can highlight many different cybersecurity concepts in a fun and interactive manner with the results and educational information being showed at the end.

3.5 Hard Mode Challenges

3.5.1 Code Exposed

For this task, players must go through an application’s directory to identify hardcoded sensitive information, such as API keys or credentials, that could lead to a security breach. They will then have to provide a solution to mitigate the security flaws such as environment variables. The challenge brings awareness to the importance of secure programming practices in developing software.

3.5.2 Cert Inspector

In this challenge players will examine the digital certificate of the ‘Data-Heist’ application. As a main task, they can use their web browser by clicking the padlock icon in the address bar to inspect the certificate and find details like the issuer, validity period and public key algorithm.

OpenSSL or online tools may also be incorporated to get a deeper analysis of the digital certificate for bonus points. For hints, links can be provided to the tutor’s lab (for students) or the official OpenSSL documentation where they can find commands to extract additional info about the cert. This challenge reinforces the value in checking websites for their encryption in online communications.

3.5.3 The Port Watcher

An Nmap-based challenge where the player will be brought through the process of a simulated vulnerability scan on a network. They’ll be asked to identify open ports, detect services or software running, and check for known vulnerabilities. This challenge let’s players act as an ethical hacker by typing in Nmap commands to scan a mock network that outputs emulated scan results.

4. Plan – Project Timeline:

Phase	Sprint	End Date	Sprint Weeks	Phase Weeks
1. RAMP	(Research, Analysis & design, Methodology, Plan)	19/01/2025		2
2. Develop (Basic)		16/02/2025		4
	Backend: Node.js, MongoDB, & JWT Authentication	26/01/2025	1	
	Frontend: React/MUI, login/signup/dashboard	02/02/2025	1	
	Features: Implement 3 easy mode challenges	16/02/2025	2	
	Features: Timer, points allocation, final score	23/02/2025	1	
3. Deployment	Frontend: Vercel Backend: Render	02/03/2025		1
4. Testing & Develop (Enhanced)	Testing: Ask people to provide feedback	06/04/2025		4
	Features: Implement hard mode challenges & Admin	16/03/2025	2	
	Features: Leaderboard & progress saving	30/03/2025	2	
	Backend unit testing	06/04/2025	1	
5. Final Review/Submission	Finish Report & Documentation	13/04/2025		1

5. Development

5.1 Backend & MongoDB Setup

The development process began, focusing on the Node & Express backend by initializing the Node environment using `'npm init'` and installing dependencies. Express, dotenv, and nodemon were among the first installed, as well as eslint and prettier for formatting the code in VScode.

Then, mongoose was installed, and it was time to connect the backend to the MongoDB database. This was done without too much delay, similarly to how it was done during the full-stack module, by connecting to a local instance set up with Studio 3T via a URI connection. The connection was implemented in `/backend/src/connect.js` using the

following function: `Mongoose.connect(process.env.db);` To view the full connect.js file see **Appendix A** at the end of this document.

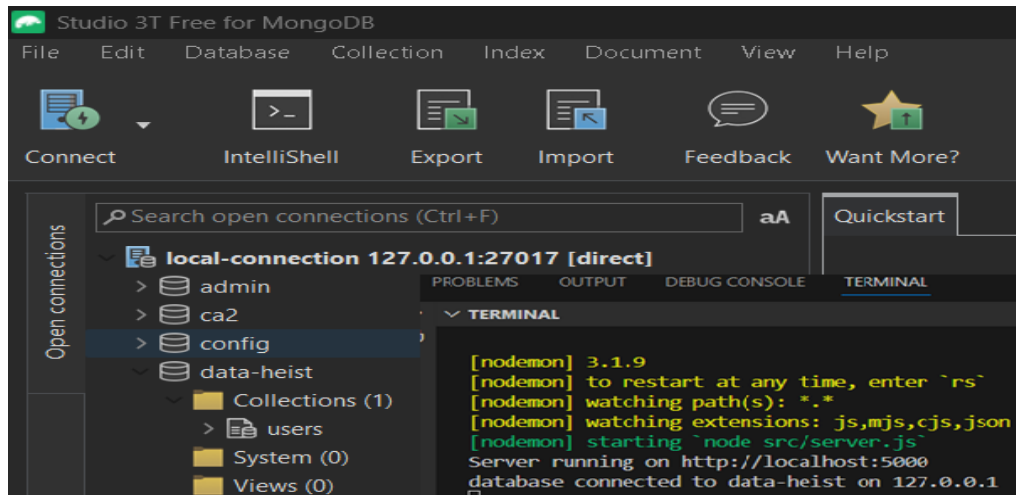


Figure 3: MongoDB connected to local instance via URI connection and Studio 3T.

5.2 User Authentication

Once the database connected successfully, the next step was to work on user authentication. The 'jsonwebtoken' library was installed, and a function to generate tokens was implemented in `/backend/src/utlis/generateToken.js`¹⁵. The function passes a user's ID as a parameter and uses the `jwt.sign()` method to generate a token which is digitally signed using a secret key stored in the `JWT_SECRET` environment variable. The token is set to expire in 1 hour for added security. The full function can be viewed in **Appendix B**.

The accounts controller that manages user signups, logins/logouts, and password hashing & salting was largely based on a previous project, HikeplaceV2, developed during the full-stack-1 module. The signup and login functions in the accounts controller can be found in **Appendix C**.

The JWT middleware that verifies tokens was then imported from another project, previously worked on in the security module. This `verifyToken()` function checks if a token is passed in the API request from the frontend and blocks the user if not. If a token is present, the function uses the `JWT_SECRET` to verify it. Once verified, the decoded token is saved in `req.user`, allowing the user to access private routes (see **Appendix D**).

To test the implemented user authentication and backend API endpoints, the dev tool 'Postman' was used. It allowed for creating, saving, and sending HTTP POST, GET, and DELETE requests manually, without the need for a frontend. Headers, body content and tokens could be included in the requests which made it great for testing the signup and login functions in the accounts controller. The screenshot below shows a GET request to the backend server (localhost:5000) using an authorization token:

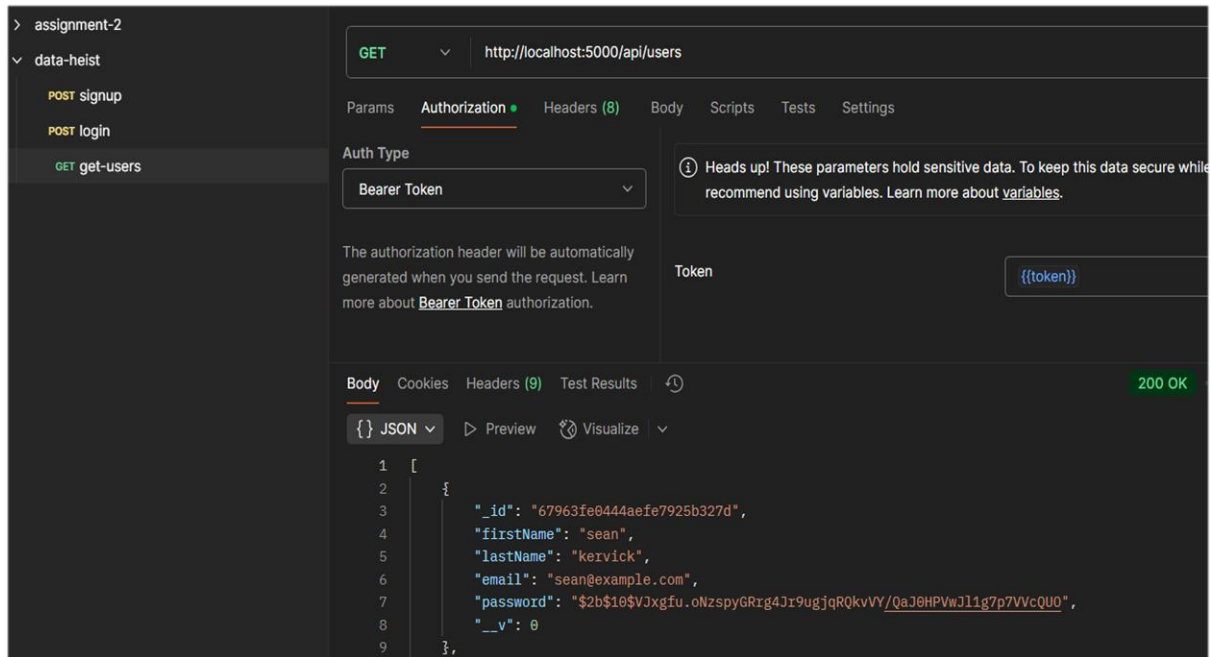


Figure 4: Postman for authentication testing.

5.3 Vite + React Frontend Setup

The Vite + React frontend was installed in a separate 'client' directory using 'npm create vite@latest':



Figure 5: Vite + React Template.

Next, the Material UI (MUI) library was installed to style the user interface for the application. It provided many pre-styled, customizable React components such as, text-fields, buttons, alerts, dialog-boxes, tables, and avatars which allowed for consistent styling across the app's pages with a small amount of effort.

A simple homepage was created to welcome users to the app, and an overall hacker/matrix-like theme was applied using MUI's 'ThemeProvider' ¹⁶:

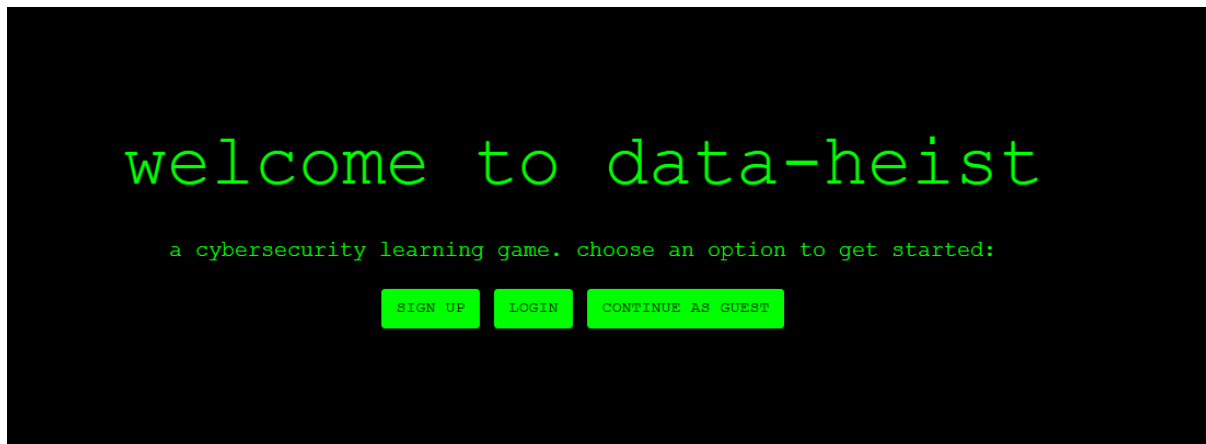


Figure 6: Data-heist homepage.

Basic user signup and login pages were then created, as well as a private dashboard route to restrict access to logged in users only. In `client/src/components/PrivateRoute.tsx` a function `PrivateRoute = () => {}` checks if a user is authenticated (logged in). The variable `isAuthenticated` becomes true if a token exists in the browser's local storage. The function then returns the protected route to allow access if the `isAuthenticated` variable is true and redirects to the login page if false. In `index.tsx`, where the routes are handled, all routes that require authentication are wrapped in `<Route element={<PrivateRoute/>}>`. See **Appendix E** for the full private route function.

'Axios' was then used to send API requests to the backend directly from the UI. After already testing the API endpoints using 'Postman' this was done without too much difficulty. Using the `handleSubmit()` function in the `SignUpPage.tsx` as an example, once the submit button is pressed, the following line: `const response = await axios.post(`${import.meta.env.VITE_API_URL}/api/signup`, formData);` uses the Axios library to send the data, that was entered into the form by the user, to the backend URL and specifically the route `/api/signup` which calls the function that handles user signup. See **Appendix F** for the full `handleSubmit()` function in `SignUpPage.tsx`.

With that, the overall structure or foundation of the app was complete. The backend was set up to handle API requests directly from the frontend UI for user signups and logins with JWT authentication and persistent storage to a local MongoDB instance.

5.4 Challenges

5.4.1 Password Cracker

This was the first of the challenges to be developed, as it had the simplest logic and a small amount of styling. Without much delay, a fake social media page was created by seeding personal information such as name, username, bio, profilePic, and correctPassword in a profile object. This was done in a separate file saved in the components folder:

`/client/src/components/Profiles.tsx` and was imported it into the `PasswordCrackerPage.tsx` file which rendered the page as seen below in figure 7:

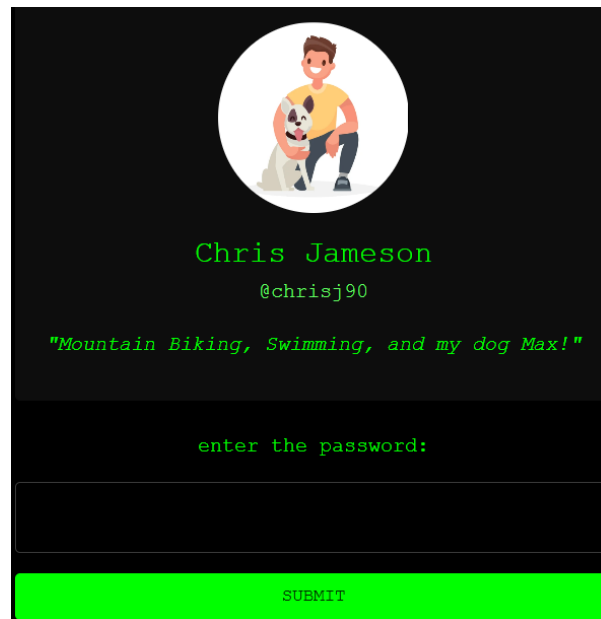


Figure 7: 'Password Cracker' challenge UI.

To pass this challenge, users were to guess the weak password based off of the information in the profile. The idea behind it was to raise awareness about social engineering attacks, by simulating how the combination of having a weak password and sensitive information visible on social media profiles can lead to hacks.

However, over the course of the development phase, the design of the app was changed slightly from having two different modes (easy & difficult) to merging the challenges into one progressive mode where the complexity increased as the player progressed. With this, as well as other factors explained below in the testing section, it was decided to remove this challenge from the application to improve the user experience.

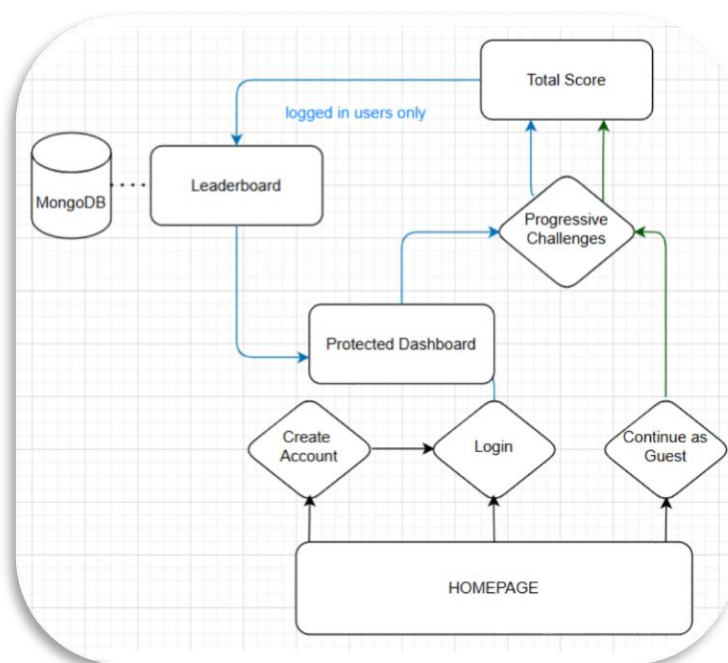


Figure 8: Updated User Flow diagram.

5.4.2 Spot the Phish

The next challenge developed was *‘Spot the Phish’*, a simulated phishing email where players are tasked to identify three ‘red flags’ that indicate the email is suspicious. Developing began by styling the page, which required more effort than the previous challenge. A real email from a bank was used as inspiration for the font and layout. MUI’s `<Box>` and `<Typography>` components were used to divide and style the layout using the `‘sx’` prop, which allows for custom CSS styling inside MUI’s React components.

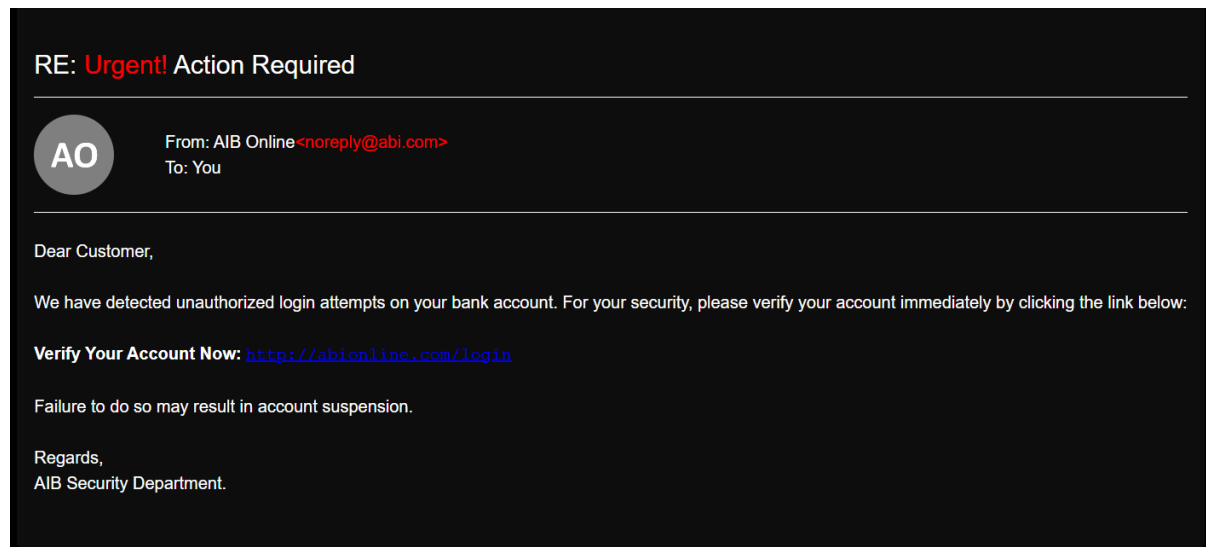


Figure 9: ‘Spot the Phish’ challenge UI.

The logic behind the challenge begins with initializing the state variable:

```
const [foundFlags, setFoundFlags] = useState<number[]>([]);
```

an array of numbers to track the red flags that are found. As players scan the email and click on the red flags, the foundFlags array is updated with the ID of the clicked red flag. This was done by wrapping the relevant text in a `<Typography>` component and using `onClick={ () => handleFlagClick(1) }`, a prop that calls the `handleFlagClick()` function and passes the ID of the red flag found to the foundFlags array. The text then changes to red as seen above in figure 9, indicating a flag has been found.

The `handleFlagClick()` function (**Appendix G**) first checks if the clicked flag’s ID is already stored in the array, if not, then the ID is added. This check stops players from clicking the same flag multiple times and is the reason why an array is more suitable than a simple number variable to track progress in this challenge. Once the 3 red flags have been found, the Boolean state variables ‘success’ and ‘showDialog’ are set to true, the timer is stopped and the score calculation is done. This causes a dialog box to open with an educational message about the challenge and allows the player to progress to the next one.

5.4.3 Quiz Round

The 3rd challenge to be developed was the *‘Quiz Round’*, a quick-fire series of questions based on cybersecurity fundamentals. The task is to simply answer the multiple choice questions within the time limit to progress and earn bonus points for each correct answer and time remaining.

This is the first challenge in the project where an interface was used in the logic. Although having previously touched on TypeScript interfaces in React during the full-stack-2 module,

the concept was still relatively new. For the *‘Quiz Round’* challenge, the interface ‘QuizQuestion’ was used to define the make-up of the questions that each have a question (string), answers (an array of strings) and a correct answer (number) which is the ID of the correct answer in the array. Using an interface here ensures that the format of all the questions stay the same:

```
export interface QuizQuestion {
  question: string;
  answers: string[]; // array of strings
  correctAnswer: number; // index of the correct answer
}
```

Figure 10: QuizQuestion type interface.

Then, an array of QuizQuestion objects of type QuizQuestion[] (from interfaces.tsx) were created for the challenge in client/src/components/QuizQuestions.tsx (see **Appendix H**). While the styling for this challenge was straightforward, the logic proved to be the most complex, and time consuming of the challenges.

The two key factors in controlling the flow of this task were to track which question is currently being displayed and how many correct answers were selected by the user. To control the display of questions, the following state variable was used:

`const [currentQuestion, setCurrentQuestion] = useState<number>(0);` Initialized with the value of zero, this `currentQuestion` variable was used to control the index of the question array:

```
<Typography variant="h5">
  /* display the question found in the array of quizQuestions at index of
  currentQuestion */
  {quizQuestions[currentQuestion].question}
</Typography>
```

Figure 11: currentQuestion variable to index array of quizQuestions.

The same was done to list the answers related to the question of the same index (value of `currentQuestion`). The answer buttons, when clicked, passed the index (of the answer selected) to the `handleAnswerClick(index)` function:

```
<Box sx={{ display: "flex", flexDirection: "column", mt: 2 }}>
  /* display the list of answers in the array of quizQuestions at index of
  currentQuestion (https://javascript.info/array-methods#map) */
  {quizQuestions[currentQuestion].answers.map((answer, index) => (
    <Button
      key={index} // https://legacy.reactjs.org/docs/lists-and-keys.html
      variant="contained"
      sx={{ mt: 1 }}
      onClick={() => handleAnswerClick(index)}
    >
      {answer}
    </Button>
  ))}
</Box>
```

Figure 12: Listing the choice answers to choose from for each question.

From this point, an `if` statement in the `handleAnswerClick(index: number)` function checks if the index of the selected answer matches the index of the correct answer stored in the `quizQuestions` array at the index of the `currentAnswer` variable. If true, the `correctAnswers` variable is incremented.

With the tracking of correct answers taken care of, the next `if` statement controlled the flow of questions by checking if the `currentQuestion` variable is less than the length of the `quizQuestions` array - 1. If true, the `currentQuestion` variable is incremented, and the next question is displayed in the UI. If false, the challenge is ended. This logic, along with timer control & score calculation can be viewed in **Appendix I**.

5.4.4 Code Exposed

This challenge tests users on their knowledge of security in software development. They're presented with a code editor in the UI and must find the security flaw or vulnerability in the code. The logic behind this task is the very same as the *'Spot the Phish'* challenge explained earlier, however the styling was the most time consuming of all the challenges. The code editor 'VSCode' was used as inspiration for the styling and the many different colours for comments, strings, and imports which increased the time and effort required to complete the styling. It was planned to have multiple files in the code editor and create a full directory, but under time constraints it was not feasible.



```
Project Files
src
  JS create-jwt.js

create-jwt.js

import jwt from 'jsonwebtoken';
const passKey = 'passKey123';

// Define the payload
const payload = {
  sub: '1234567890',
  name: 'Rory Gallagher',
  roles: ['admin', 'user'],
  iat: Math.floor(Date.now() / 1000), // Issued at (current timestamp)
  exp: Math.floor(Date.now() / 1000) + (60 * 60), // Expiration time (1 hour from now)
};

// Generate the token
const token = jwt.sign(payload, passKey, { algorithm: 'HS256' });

console.log(token);
```

Figure 13: 'Code Exposed' challenge UI.

5.4.5 Cert Inspector

The cert inspector challenge was much less complex to create the others. A simple form to accept the correct answer and minimal styling was all that was required. Although this challenge was easy to implement, it adds to the user experience and creates an interesting task where players must go looking for the digital certificate of the data-heist website and retrieve information. Initially, the correct answer to pass this challenge was the SHA 256 fingerprint from the digital cert, but after some testing and feedback (explained in the testing section), this was changed to the certificate's expiry date. A link was provided to the online tool, SSL labs' server test, for mobile users to improve their experience as it was difficult to retrieve certificate information on a mobile browser.



Figure 14: Cert Inspector UI²²

5.4.6 The Port Watcher

The final and most complex task to get through as a player, The Port Watcher, also proved difficult to develop. Guided through four steps of a simulated vulnerability scan using 'nmap', the users must find the right commands and enter them into the terminal to perform the requested scan at each step. Entering the 'nmap -h' command outputs a help response where they can find clues and use information discovered in previous steps to find the right command.

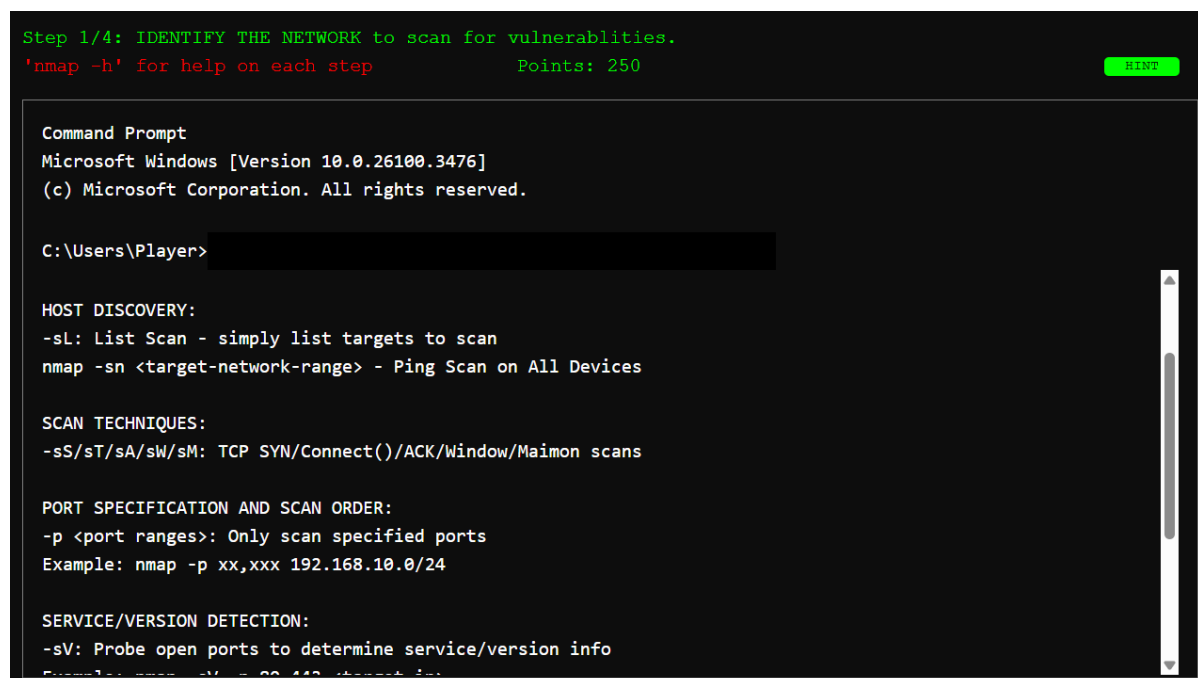


Figure 15: The Port Watcher UI with 'nmap -h' response.

The styling took quite a lot of time and effort to replicate the windows command prompt, in particular, the input field caused the most issues. MUI has many different styles of `<TextField>` components, none of which were found suited to the totally blank input field of the windows terminal. Eventually, MUI's `<Input>` component¹⁷ was found and customized to suit. Most interestingly, the `autofocus` prop provided a neat detail of the cursor automatically being active and visible (flashing) in the input field when the challenge opens. A small but essential detail for the replication of the windows command prompt, where the terminal opens ready to accept commands, without the requirement to click on the field.

The first step of programming the logic was to create terminal outputs for valid commands. An object of type `Record<string, string>`¹⁸ allowed for storing commands and their corresponding responses. This was done in a separate `NmapCommands.tsx` file in the components folder which allowed for many commands and their responses to be added, while keeping `ThePortWatcherPage.tsx` file clean. At this point, if the inputted command is valid, the terminal responded with its corresponding output:

```
let terminalOutput = "Invalid command. Try again.";
// always allow all commands
if (inputCommand in storedCommands) {
  terminalOutput = storedCommands[inputCommand];
}
```

Figure 16: Finding the right response in stored commands.

Otherwise, the output is "Invalid command. Try again!" as this is the string the `terminalOutput` variable is initialised at.

Then, step logic and guidance were introduced. A guidance message for each step was displayed in the UI to instruct users what they were doing and what type of scan was required. This was done using basic 'if' statements that checked which step the user was on and if the correct command was entered. If both conditions are true, the terminal output was set to the correct response, and the guide message was updated for the next step. See appendix J for the full `handleKeyPress()` function that handles The Port Watcher challenge step-based guidance and logic.

5.5 Timer System

To implement the timer system, React's `useEffect(() => {})` hook was used so that the timer function runs independent from or "on the side"¹⁹ of the main application logic. The interface `TimerProps{}` was used to define the type of props passed to the timer function:

```
export interface TimerProps {
  initialTime: number;
  start: boolean; // timer stop/start control
  onTimeUp: () => void; // function calls when time runs out
  onTimeUpdate: (time: number) => void; // for tracking time used for points
}
```

Figure 17: TimerProps

These props allowed for the timer to be controlled and adjusted for each of the individual challenges. Then the timer logic to decrease its value (time) by 1 every second was programmed in the `useEffect()` hook using the `setTimeout()` method, *an asynchronous function meaning that the timer function will not pause execution of other functions in the functions stack.*²⁰. The timer was then displayed in 4 out of 5 challenges each with a different amount of initial time to begin with. The time remaining on challenges was used in calculations for point allocation. See **Appendix K** for the timer function logic.

5.6 Points Allocation, Leaderboard & Hints

The way in which the points were allocated varied across the different challenges. The scoring system has been designed to reward users more heavily for the more complex tasks and penalise if hint buttons were used:

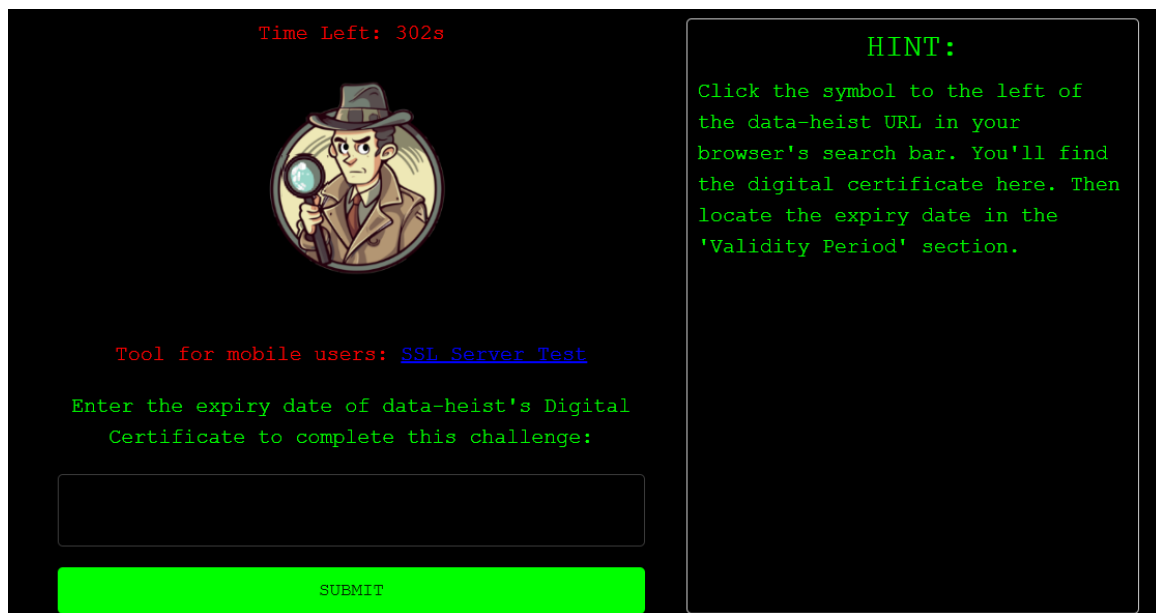


Figure 18: Cert Inspector UI – hint button pressed.

Using the Cert Inspector challenge as an example, the score was calculated using the time remaining and subtracting 100 points if the hint button was used. When the correct answer is submitted, the `handleSubmit()` function is triggered which stops the timer and calculates the score for this challenge. The challenge score is then added to the overall total score and saved locally in the browser to be brought forward to the next challenge:

```
setSuccess(true); // end of challenge
setTimerStart(false); // stop the timer
const challengeScore = time - hintUsed;
setChallengeScore(challengeScore); // update state for dialog box
const updatedTotal = totalScore + (challengeScore); // calculate total score
localStorage.setItem("totalScore", (updatedTotal).toString()); // store
updated total score
```

Figure 19: Score calculation.

The hint button, when clicked, displays a hint to help the user as seen above in figure 18. This works by using the `onClick()` prop to call a `handleHintClick()` function that causes the button to disappear, set the `hintUsed` value to 100 and set the hint message to display:

```
const handleHintClick = () => {
  setShowHintButton(false);
  setHintUsed(100)
  setHint(`Click the symbol to the left of the data-heist URL in your
  browser's search bar. You'll find the digital certificate here.
  Then locate the expiry date in the 'Validity Period' section.`)
}
```

Figure 20: *handleHintClick* function.

This is how the score is calculated and stored locally in the browser until the end of the game. At the feedback page, the final score is presented to the user regardless of if they chose to play as a guest or logged in. If the user logged in they have additional access for their score to be persistently saved and feature on the leaderboard (if within the top ten high scores).

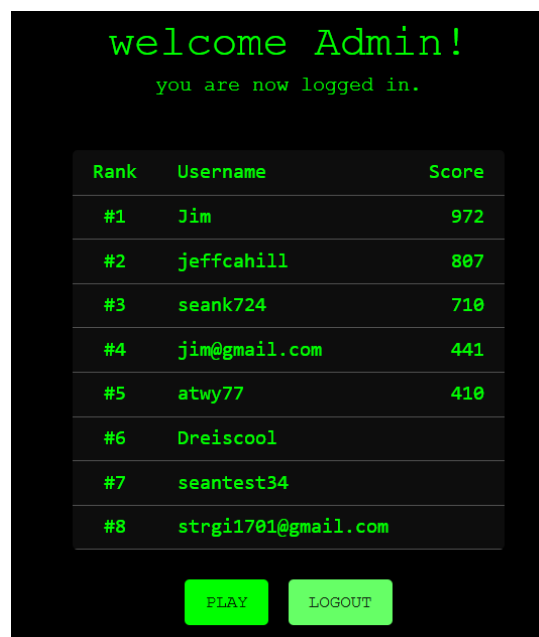
The final score is passed to an `updateHighScore()` function in `client/src/api/scoreAPI.tsx` where an axios POST request is sent to the backend with the score and username as the body, and the user's JWT token for authorization. This triggers the `updateHighScore()` function in the backend which first finds the user in the database, then checks if their score is higher than their previous score (or zero if no previous is found) before saving it to the database.

The `getLeaderboard()` function in the backend then finds the users with the top ten scores by sorting them descending order and limited to 10:

```
const topUsers = await User.find().sort({ score: -1 }).limit(10);
```

Figure 21: *getLeaderboard* function (backend)

The function is then called in the frontend with an axios GET request and displayed in the UI. To view the frontend and backend functions for storing scores and fetching the leaderboard see **Appendix's L** and **M** respectively.



welcome Admin!
you are now logged in.

Rank	Username	Score
#1	Jim	972
#2	jeffcahill	807
#3	seank724	710
#4	jim@gmail.com	441
#5	atwy77	410
#6	Dreiscool	
#7	seantest34	
#8	strgi1701@gmail.com	

PLAY LOGOUT

Figure 22: *Leaderboard*

5.7 Deployment

The application was deployed successfully, first using Vercel's free tier for the frontend and second, Render's free tier for the backend. *CORS, or Cross-Origin Resource Sharing, is a mechanism that allows web pages to access resources from servers on different domains*²¹ which enabled communication between the deployed frontend & backend.

6. Testing

The application was first put out for testing on the 23/02/2025, once the first beta version of the frontend was deployed on Vercel. At this point, users could play 3 challenges and see their total score at the end. Signup/login, leaderboard, and hint button features were not implemented at this time, but it was beneficial to get some initial feedback.

The 'Password Cracker' challenge, although thought to be an easy challenge, proved to be causing significant difficulties for some users. The poor password 'Max', the name of the person's dog in the social media profile, proved for some users to be such a poor password that it was difficult to guess. This left some players stuck on the first challenge without being able to progress to the next. This highlighted the need for hint buttons so that players can get through the game. Over time, the 'Password Cracker' challenge generally seemed to hinder the user experience rather than improve it. As well as this, the underlying logic was a simple form, and this same logic was used in another challenge, 'The Cert Inspector'. For these reasons the challenge was removed as it was deemed to be of small value for both the overall application and showing skills as a developer.

A major issue with the 'Cert Inspector' challenge was highlighted during the testing phase. Initially, the SHA 256 fingerprint for the data-heist website was used as the correct answer to pass the challenge. However, during testing, it was found that different browsers or networks displayed different encrypted fingerprints from the digital certificates. It was assumed the reason behind this was not that the fingerprint had changed, but the way in which it is encrypted may vary across different networks or browsers. Without investigating the issue too much, the correct answer was changed to the certificate's expiry date which quickly resolved the problem.

When the backend was deployed on Render, the backend features were tested. All worked as expected, as they had done locally, with the only major issue being the 'cold-start delay' that came with using the free tier on Render. If it was the first time the app was used in some time, logins and signups would take a significant amount of time, due to this cold-start or the backed 'waking up'. For this reason, alert messages were developed to notify users that there may be a delay of up to a minute (or two) on signup and login:



Figure 23: Login alert message.

The final version of the app was released for testing on 02/04/2025. With all features implemented at this point, the general feedback was that the user experience was good overall with no major issues. One user mentioned that another button for an instructional reminder would be beneficial as during some challenges, the actual task was forgotten and that a reminder would help. Unfortunately, at this point, time was limited with the submission date closing in. The reminder button would have to be left out.

7. Conclusion

Overall, the project is deemed to be a success, resulting in a fully functioning full-stack web application that meets the objectives that were set out at the beginning. Highly valuable experience was gained throughout the process, particularly in the area of the chosen stack: Node.js & Express, React + Vite, MUI and TypeScript, Mongoose & MongoDB.

The application was largely built from scratch, using elements from different projects and modules completed during the course, as well as help from new sources of information which were plentiful on the web, a huge benefit of developing a project with the chosen technology stack.

An unexpected amount of time was spent on the styling of some of the challenges. In particular, the code editor in 'Code Exposed' and the windows command prompt terminal in 'The Port Watcher'. Although the end product of the UI turned out well, personally, styling would not be considered a strong point as a developer, with much more enjoyment found in programming logic and the problem solving side of development.

Programming the logic behind the different tasks was indeed challenging but highly rewarding. Looking at the design, it was at first daunting to imagine the difficulties that may arise throughout the development. However, by starting with simple forms and taking the more complex challenges by one step at a time, the application was successfully completed.

The importance of planning and creating a detailed timeline to stick to at the beginning was hugely appreciated. Each week the features were developed accordingly to the plan and timeline so that progress could be tracked. This made it possible to forecast what features could be implemented within the given time frame.

For the most part, everything that was set out in the plan was achieved. If more time was available perhaps some overall polishing would be done across the UI, implement more features such as an action when time runs out, buttons to skip challenge and maybe to remind users of the task at hand. Progress saving and backend unit testing were two items included in the design that were not achieved due to time constraints.

The project landing page website can be viewed [here](#) where links to the deployed app, the parent GitHub repo, and more information can be accessed.

8. AI Usage

Due to the abundance of information on the web, with official documentation such as [MUI.com](#), [React.dev](#) & [MongoDB.com](#) any additional resources needed to build the project were widely available due to the large community of developers using these technologies. For particular code snippets that have been taken from resources, past projects, or online have been referenced in the bibliography and by using comments throughout the code.

Artificial Intelligence (ChatGPT) was utilized to speed up certain processes in the development phase that otherwise would've taken a considerable amount of time to complete manually. Specifically, it helped with styling the *Code Exposed* & *The Port Watcher* challenges, as well as creating realistic, fake data for the simulated nmap responses.

As mentioned previously, for confirmation on the right deployment strategy, ChatGPT was used to advise and step through the process of deploying the frontend on Vercel, and the Backend on Render.

8. Bibliography

1. <https://github.com/CamilleSA/GameOfCybersecurity>
2. [https://en.wikipedia.org/wiki/React_\(software\)](https://en.wikipedia.org/wiki/React_(software))
3. <https://dr-arsanjani.medium.com/the-react-framework-does-scale-4c7db0d0e3aa>
4. <https://www.geeksforgeeks.org/advantages-and-disadvantages-of-css/>
5. <https://bambooagile.eu/insights/key-advantages-of-vue-js-in-web-app-development>
6. <https://ddi-dev.com/blog/programming/the-good-and-the-bad-of-vue-js-framework-programming/>
7. [https://en.wikipedia.org/wiki/Django_\(web_framework\)](https://en.wikipedia.org/wiki/Django_(web_framework))
8. <https://graffersid.com/node-js-advantages-and-disadvantages/>
9. <https://www.information-age.com/kubernetes-vs-docker-pros-and-cons-123501812/>
10. <https://www.netguru.com/blog/django-pros-and-cons>
11. <https://www.mongodb.com/resources/compare/advantages-of-mongodb>
12. <https://www.thinkautomation.com/our-two-cents/understanding-the-key-mongodb-pros-and-cons>
13. Deployment strategy – ChatGPT
14. <https://vite.dev/>
15. Generate token - <https://stackoverflow.com/questions/73441984/typescript-logged-user-data>
16. MUI theme provider - <https://github.com/mui/material-ui/blob/v6.4.2/apps/pigment-css-vite-app/src/main.tsx>
17. MUI input component: <https://mui.com/base-ui/react-input/>
18. Record type object: <https://refine.dev/blog/typescript-record-type/#best-practices>
19. React useEffect hook: <https://legacy.reactjs.org/docs/hooks-effect.html>
20. setTimeout() method: <https://developer.mozilla.org/enUS/docs/Web/API/Window/setTimeout>
21. CORS - https://www.google.com/search?q=CORS&oq=CORS&gs_lcrp=EgZjaHJvbWUyBggAEEUYOTIHCAEQABiPAjIHCAIQABiPAjIHCAMQABiPAjIGCAQQRRg8MgYIBRBFgdwyBggGEEUYPDIGCAcQLhA0gEIMTc2N2owajGoAgiwAgE&sourceid=chrome&ie=UTF-8
22. Cert-Inspector Image: <https://pngtree.com/so/inspector-clipart>

9. Appendix

A – backend/src/connect.js

```
// https://github.com/wit-hdip-comp-sci-2023/full-stack-1/blob/main/prj/playtime/playtime-0.5.0/src/models/mongo/connect.js
import * as dotenv from "dotenv";
import Mongoose from "mongoose";

dotenv.config();

export function connectMongo() {

  Mongoose.set("strictQuery", true);
  Mongoose.connect(process.env.db);
  const db = Mongoose.connection;

  db.on("error", (err) => {
    console.log(`database connection error: ${err}`);
  });

  db.on("disconnected", () => {
    console.log("database disconnected");
  });

  db.once("open", function () {
    console.log(`database connected to ${this.name} on ${this.host}`);
  });
}
```

B – backend/src/utils/generateToken.js

```
//credit: https://stackoverflow.com/questions/73441984/typescript-logged-user-data
import jwt from "jsonwebtoken";

export const generateToken = (userId) => {
  const token = jwt.sign({ id: userId }, process.env.JWT_SECRET, {
    expiresIn: "1h",
  });
  console.log("Generated Token:", token); // Log the generated token
  return token;
};
```

C – backend/src/accountsController.js

```
//https://github.com/SeanKervick/hikeplaceV2/blob/master/src/controllers/accounts-controller.js

import bcrypt from "bcrypt";
import { User } from "../models/User.js";
import { generateToken } from "../utils/generateToken.js";

export const accountsController = {
  signup: async (req, res) => {
    const { username, email, password } = req.body;

    try {
      // check if user already exists
      const existingUser = await User.findOne({ username });
      if (existingUser) {
        return res
          .status(400)
          .json({ message: "email address already in-use" });
      }

      // hash password
      const hashedPassword = await bcrypt.hash(password, 10);
      console.log("hashed password:", hashedPassword);

      // create user
      const newUser = new User({
        username,
        email,
        password: hashedPassword,
      });
      console.log("user created:", newUser);

      // save user
      const savedUser = await newUser.save();
      console.log("user saved:", savedUser);

      // generate a token
      const token = generateToken(savedUser._id);
      // pass username in response to frontend for displaying in UI & assign
      // role for access control
      res.status(201).json({ message: "user created successfully", token,
        username: savedUser.username, role: "user" });
    } catch (error) {
      res.status(500).json({ message: "backend signup error" });
    }
  },

  login: async (req, res) => {
```

```
const { email, password } = req.body;
try {
  // check if the credentials match the admin's credentials from the .env
  file
  if (email === process.env.ADMIN_EMAIL && password ===
process.env.ADMIN_PASSWORD) {
    console.log("logging in: admin");
    const token = generateToken({id: "admin"});
    return res.json({ message: "admin logged in successfully", token,
username: "Admin", role: "admin" }); // assign role for access control
  }

  // search database by email address
  const user = await User.findOne({ email });
  if (!user) return res.status(400).json({ message: "user not found" });

  // compare provided password with stored hashed password
  const isPasswordValid = await bcrypt.compare(password, user.password);
  if (!isPasswordValid) return res.status(400).json({ message: "Invalid
email or password" });

  // successful login, generate token
  const token = generateToken(user._id);
  // pass username in response to frontend for displaying in UI & assign
role for access control
  res.status(200).json({ message: "login successful", token, username:
user.username, role: user.role, });
} catch (error) {
  res.status(500).json({ message: "error logging in" });
}
},
```

D – backend/src/middleware/jwt.js

```
// https://github.com/johnrellissetu/assignment-2-api-security-SeanKervick
import jwt from 'jsonwebtoken';

const verifyToken = (req, res, next) => {
  // Get token from Authorization header
  // will split into ["Bearer", "xxxx.yyyy.zzzz"]
  const token = req.headers['authorization']?.split(' ')[1];
  console.log("Token Received:", token);

  if (!token) {
    return res.status(401).json({ message: 'No token provided' });
  }
  try {
    // Verify token
```

```
    const decoded = jwt.verify(token, process.env.JWT_SECRET); // Add the
    decoded payload to the request for access in other routes
    console.log("JWT_SECRET:", process.env.JWT_SECRET);

    req.user = decoded;
    console.log("Decoded Token:", decoded);

    next(); // Proceed to the next middleware or route handler
  } catch (error) {
    console.error(error);
    return res.status(403).json({ message: 'Token invalid' });
  }
};

export { verifyToken };
```

E – client/src/components/PrivateRoute.tsx

```
// credit: https://blog.stackademic.com/how-to-create-a-private-route-in-
react-typescript-d43e2b162d46

import { Navigate, Outlet } from "react-router-dom";

const PrivateRoute = () => {
  const isAuthenticated = !!localStorage.getItem("token"); // check if token
  exists

  return isAuthenticated ? <Outlet /> : <Navigate to="/login" replace />;
};

export default PrivateRoute;
```

F – client/src/pages/SignUpPage.tsx

```
const handleSubmit = async (e: React.FormEvent) => { // type of event object
  specified
  e.preventDefault(); // stops browser from reloading the page

  // backend api call
  try {
    setError(null);
    setDelayMessage("Sign-up may take a minute (or two) while the backend
    server wakes up due to the free tier delay. Thanks for waiting!");
    const response = await
    axios.post(`${import.meta.env.VITE_API_URL}/api/signup`, formData);

    console.log("account created successfully:", response.data);
```

```

        localStorage.setItem("token", response.data.token); // store JWT token
        locally in the browser
        localStorage.setItem("username", response.data.username); // save
        username in local storage for displaying in UI
        navigate("/dashboard"); // redirect to dashboard

    } catch (error) {
        console.error("signup error frontend:", error);
        setDelayMessage(null);
        setError("username already exists");
    }
};

```

G – client/src/pages/challenges/SpotThePhishPage.tsx

```

const handleFlagClick = (id: number) => {
    // if foundFlags does not include the id of the clicked flag
    if (!foundFlags.includes(id)) {
        // add the id to the array (https://react.dev/learn/updating-arrays-in-state)
        setFoundFlags([...foundFlags, id]);

        if (foundFlags.length + 1 === 3) {
            setSuccess(true);
            setTimerStart(false); // Stop the timer

            const challengeTotal = (time + ((foundFlags.length + 1) * 10))
            setChallengeScore(challengeTotal); // set challenge score for end of
            challenge dialog UI
            const updatedTotal = totalScore + challengeTotal; // calculate total
            score
            localStorage.setItem("totalScore", (updatedTotal).toString()); //
            store updated total score

            setShowDialog(true); // Show dialog before next challenge
        }
    }
};

```

H – client/src/components/QuizQuestions.tsx

```

import { QuizQuestion } from "../../types/interfaces"

// export array of question objects
export const quizQuestions: QuizQuestion[] = [
    {
        question: "Q1: Which of the following is a strong password?",
        answers: ["9876passwr49846", "Max1990", "Tr#8!vP29xD"],
        correctAnswer: 2, // index of correct answer
    }
];

```



```
    },  
    {  
      question: "Q2: What is the primary purpose of a Virtual Private Network (VPN)?",  
      answers: [  
        "Boost internet speed",  
        "To log in to websites from different regions",  
        "To secure & encrypt your internet connection",  
      ],  
      correctAnswer: 2, // index of correct answer  
    },  
    {  
      question: "Q3: Which of the following best defines malware",  
      answers: [  
        "Software designed to improve system performance",  
        "Malicious software intended to damage",  
        "A tool used for encryption",  
      ],  
      correctAnswer: 1, // index of correct answer  
    },  
  ],  
];
```

I – client/src/pages/challenges/QuizRoundPage.tsx

```
const handleAnswerClick = (index: number) => {  
  let newCorrectAnswers = correctAnswers; // start with current correctAnswers value  
  
  // check if the selected answer (index) is correct  
  if (index === quizQuestions[currentQuestion].correctAnswer) {  
    newCorrectAnswers += 1; // increment correctAnswers  
    setCorrectAnswers(newCorrectAnswers); // update state  
  }  
  
  // if currentQuestion <number> is less than the array of questions, add 1 (move to next question)  
  if (currentQuestion < quizQuestions.length - 1) {  
    setCurrentQuestion(currentQuestion + 1);  
  } else {  
    // end of quiz  
    setSuccess(true);  
    setTimerStart(false); // stop the timer  
  
    const challengeTotal = time + (newCorrectAnswers * 10); // calculate challenge total  
    setChallengeScore(challengeTotal); // update state for end of challenge dialog box UI  
  }  
}
```

```

        const updatedTotal = totalScore + challengeTotal; // calculate overall
total
        localStorage.setItem("totalScore", updatedTotal.toString()); // store
updated total score

        setShowDialog(true); // show dialog at end of this challenge
    }
};

```

J – client/src/pages/challenges/ThePortWatcherPage.tsx

```

// https://react-typescript-cheatsheet.netlify.app/docs/basic/getting-
started/forms_and_events/
const handleKeyPress = (e: React.KeyboardEvent<HTMLInputElement>) => {
    if (e.key === "Enter") {
        let nextGuide = guideResponse; // guide response control
        let terminalOutput = "Invalid command. Try again.";

        // always allow all commands
        if (inputCommand in storedCommands) {
            terminalOutput = storedCommands[inputCommand];
        }

        // step-based guidance through scan
        if (step === 1 && inputCommand === "ipconfig") {
            terminalOutput = storedCommands[inputCommand];
            nextGuide = `Well done! The IPv4 address and Subnet Mask found
below, determines that the network range is 192.168.10.0/24.
Use this range to PING ALL DEVICES on the network and discover
active hosts.`;
            setHint("");
            setStep(2);
        } else if (step === 2 && inputCommand === "nmap -sn 192.168.10.0/24")
{
            terminalOutput = storedCommands[inputCommand];
            nextGuide = `Great! You've found 7 active hosts (see below). Now
scan for common open ports like HTTP (80) and
HTTPS (443) using -p 80,443 to find potentially vulnerable
services.`;
            setHint("");
            setStep(3);
        } else if (step === 3 && inputCommand === "nmap -p 80,443
192.168.10.0/24") {
            terminalOutput = storedCommands[inputCommand];
            nextGuide = `Nice work! Ports 80 & 443 on gateway.local
(192.168.10.1) are open.
Run a SERVICE DETECTION SCAN on this device to identify the software
– it may reveal exploitable vulnerabilities!`;
            setHint("");
            setStep(4);
        }
    }
};

```

```

    } else if (step === 4 && inputCommand === "nmap -sV -p 80,443
192.168.10.1" ) {
        terminalOutput = storedCommands[inputCommand];
        nextGuide = `Congratulations! You've identified that the router is
running Apache httpd 2.4.49.
This version is outdated and has known vulnerabilities, making it a
potential security risk.`;
        setStep(5);
        setHint("");
        setShowHintButton(false);
        // challenge complete
        setSuccess(true);

        setChallengeScore(points); // update state for end of challenge
dialog box UI
        const updatedTotal = totalScore + points; // calculate total score
        localStorage.setItem("totalScore", (updatedTotal).toString()); //
store updated total score

        setTimeout(() => {
            setShowDialog(true); // show dialog at end of this challenge
        }, 15000);

    }

    setTerminalOutput(terminalOutput);
    setGuideResponse(nextGuide);
    setInputCommand(""); // clear the input field
    setShowHintButton(true);
}
};

```

K – client/src/components/Timer.tsx

```

const Timer: React.FC<TimerProps> = ({ initialTime, start, onTimeUp,
onTimeUpdate }) => { // define Timer component as a function
    const [time, setTime] = useState<number>(initialTime); // set state variable
to initialTime in each challenge

    useEffect(() => { // useEffect hook used to run the function 'on the side'
https://legacy.reactjs.org/docs/hooks-effect.html
        if (!start) { // wait until 'start' is true
            setTime(initialTime); // reset timer when a new challenge starts
            return;
        }

        if (time > 0) {
            setTimeout(() => { // "asynchronous function" -
https://developer.mozilla.org/en-US/docs/Web/API/Window/setTimeout

```

```
    setTime(time - 1);
    onTimeUpdate(time - 1); // for tracking points
  }, 1000);
} else {
  onTimeUp(); // call function when timer reaches 0
}
}, [initialTime, time, start, onTimeUpdate, onTimeUp]); // useEffect
dependencies

// display time remaining in challenge
return <Typography color="red">Time Left: {time}s</Typography>;
};

export default Timer;
```

L – backend/src/scoreController.js

```
import { User } from "../models/User.js";

export const updateHighScore = async (req, res) => {
  const { username, score } = req.body;
  // check body for debugging
  console.log(req.body);

  try {
    // find user by username (stored locally in browser at login and used for
    API call)
    const user = await User.findOne({ username });
    if (!user) return res.status(404).json({ error: "User not found" });

    // only update the score if higher than the user's previously saved score
    if (score > (user.score || 0)) {
      user.score = score;
      await user.save();
    }

    res.status(200).json({ message: "Score updated", highScore: user.score });
  } catch (err) {
    res.status(500).json({ error: "Failed to update score" });
  }
};

export const getLeaderboard = async (req, res) => {
  try {
    // get top 10 users, sort them by score in descending order (-1)
    https://www.mongodb.com/docs/manual/reference/method
    const topUsers = await User.find().sort({ score: -1 }).limit(10);
    res.status(200).json(topUsers);
  } catch (err) {
```

```
    console.error("Error fetching leaderboard:", err);
    res.status(500).json({ error: "Failed to fetch leaderboard" });
  }
};
```

M – client/src/api/scoreController.tsx

```
import axios from "axios";

// pass finalScore from feedback page to store by username
export const updateHighScore = async (score: number) => {

  const token = localStorage.getItem("token");
  const username = localStorage.getItem("username");

  await axios.post(
    `${import.meta.env.VITE_API_URL}/api/update-score`, // post api route to
    update user's score
    { score, username },
    {
      headers: { Authorization: `Bearer ${token}` },
    }
  );
};

export const getLeaderboard = async () => {
  const { data } = await
  axios.get(`${import.meta.env.VITE_API_URL}/api/leaderboard`); // call api
  route for fetching leaderboard
  return data;
};
```