SETU Code Lab
Final Report

Diarmuid O'Neill

South East Technological University

24/04/2026

Table of Contents

Table of Figures

Abstract

SETU Code Lab is an in-browser coding platform designed to support both teaching and independent learning for students in computing-related courses. Few platforms merge gamified self-directed code education with learning management and automatic grading. This project provides a platform where lecturers can create coding problems with test cases, automatically evaluate student submissions, and manage course activity, while students can practice their coding skills through a shared problem repository enhanced with gamification features.

This system was implemented using a React frontend, a Node.js and Express backend and a PostgreSQL database. A key technical consideration with this project was the execution of untrusted student code using Docker containers. This required the development of code test harnesses for Java and Python and custom container configurations for code execution.

Other key challenges included handling complex input parsing for Java functions, hosting the system online, and building a usable, yet secure user interface. These were addressed through buffer processing techniques, setting up a Digital Ocean virtual private server and implementing refresh tokens. The final system meets all the core functional and non-functional requirements and supports Java and Python for code execution.

Introduction

SETU Code Lab is an in-browser study tool for students enrolled in computing-related courses in SETU (South East Technological University) and a learning management platform for their lecturers. Lecturers can create problems and courses and add students to these courses. This allows lecturers to easily assign work to groups of students and gather their submissions all in one place. When a lecturer creates a problem, they define several test cases. Student solutions are then run using inputs from these test cases and automatically graded for the lecturer. The lecturer can view every student's solution and see how they performed on each problem and download class results in .CSV format for their convenience.

As well as a learning management platform, SETU Code Lab can also be used by students to study for coding exams and technical interviews. There is a ***Global Problems*** course available to all users, to which any lecturer can add problems to for students to practice in their own time. To keep students engaged the platform also employs several gamification techniques such as daily login streaks, points, badges and a leaderboard. It also supports Java and Python for code execution.

This report reflects on the development process of SETU Code Lab and outlines the challenges encountered, how they were solved and what changes were made to the project along the way. It will also look at the testing process of SETU Code Lab, assess its success and reliability as a software product, and discuss the skills and learnings I have acquired.

Challenges and Learnings

Architecture

One of the first challenges was learning how to structure files in an easily maintainable and extensible way so that as the codebase grows, it is still comprehensible and easy to work on. This was overcome through independent learning and research. The goal when thinking about how the project should be laid out was to keep coupling low and cohesion high.

Backend development, proved to be a big challenge in this regard as I had limited prior experience with it. This required research into:

- How to make HTTP requests from the frontend using TypeScript
- How to define a corresponding route in the backend using TypeScript and Express.js
- How to structure backend files

From this research I learned that the back-end web server should be broken down into several different types of files based on their purpose keeping cohesion high and coupling low. The most notable source for this research was a website called "Treble".

The first back-end file type is ***Routes***; this is where API endpoints are defined and they map to controller methods (see Figure 1). Before a route sends its request to the controller file there may be a middleware file which executes beforehand. This is typically used to check if a user is authenticated to make the request before forwarding it to the controller or for error handling (Gore, 2022). Routes can be called in the frontend by making a fetch request (see Figure 2). Later

this logic was abstracted away to a group of methods called `api.get()`, `api.post()` and `api.delete()` to reduce code duplication (see Figure 3).

```
import { Router } from "express";
import { login, signUp, me, refresh, logout } from "../controllers/auth.controller"
import { verifyToken } from "../middlewares/auth";

const router: Router = Router();

router.post("/login", login);
router.post("/signup", signUp);
router.get("/me", verifyToken, me);
router.post("/refresh", refresh);
router.post("/logout", logout);

export default router;
```

Figure 1. *Example route file*

```
const res = await fetch("/api/auth/signup", {
    method: "POST",
    credentials: "include",
    headers: {
        "Content-Type": "application/json",
    },
    body: JSON.stringify({
        name: sanitizedName, role, email: sanitizedEmail, password, confPassword
    })
});
```

Figure 2. *Making a request to the backend*

```
export const api = {
  get: (url: string) => fetchWithRefresh(url, { method: "GET" }),
  post: (url: string, body: object) =>
    fetchWithRefresh(url, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(body),
    }),
  delete: (url: string) => fetchWithRefresh(url, { method: "DELETE" }),
};
```

Figure 3. *Abstracted request method*

Next the request is forwarded to its corresponding ***Controller*** method in a controller file. Like the route files, each controller file contains only methods related to one use case to keep cohesion high. Controller methods prepare the incoming data in the request before it is sent to a service method(s). Controllers are not supposed to perform any business logic, instead leaving this responsibility to the next type of file in the process, the service file (Gore, 2022). Figure 4 below shows the signup controller method: extracting the needed data from the request, performing some basic input validation via the `isValidPassword()` method, delegating the business logic to the `signUpUser()` service method and attaching cookies before returning the response.

```ts
export const signUp = async (req: Request, res: Response) => {
    try {
        const { name, role, email, password, confPassword } = req.body;

        if (!email || !password || !name || !role || !confPassword) {
            return res.status(400).json({ message: "Email, password, name and role required" });
        }

        isValidPassword(password, confPassword);

        const { user, accessToken, refreshToken } =
            await signUpUser(name, role, email, password);

        res.cookie("token", accessToken, {
            httpOnly: true,
            secure: process.env.NODE_ENV === 'production',
            sameSite: "lax",
            path: "/",
            maxAge: 3 * 60 * 60 * 1000
        });

        res.cookie("refreshToken", refreshToken, {
            httpOnly: true,
            secure: process.env.NODE_ENV === 'production',
            sameSite: "lax",
            path: "/",
            maxAge: 7 * 24 * 60 * 60 * 1000
        });

        res.status(201).json({
            message: "Account created successfully",
            user
        });
    } catch (error: any) {
        res.status(400).json({ message: error.message });
    }
};
```

Figure 4. *Controller method*

The ***Service*** files are intended to handle business logic. This is any operation that need to be performed on data before it is either added to the database or displayed to the frontend (Gore, 2022). Figure 5 shows an example service file from the codebase. This service performs operations such as hashing the user's password and communicating with the `authModel` to add a user to the database. ***Model*** files are responsible for any operation on the database and simply execute database queries (see Figure 6).

```ts
export async function signUpUser(name: string, role: string, email: string, password: string): Promise<LoginResult> {
    const existing = await authModel.getUserByEmail(email);
    if (existing) {
        throw new Error("Email already in use");
    }
    const client = await pool.connect();
    try {
        await client.query("BEGIN");
        const hashed = await bcrypt.hash(password, 10);
        const user = await authModel.createUser(client, name, role, email.toLowerCase(), hashed);
        const accessToken = jwt.sign(
            { user_id: user.user_id, email: user.email, role: user.role },
            process.env.JWT_SECRET!,
            { expiresIn: "3h" }
        );
        const refreshToken = jwt.sign(
            { user_id: user.user_id },
            process.env.JWT_REFRESH_SECRET!,
            { expiresIn: "7d" }
        );
        await courseModel.addUserToCourse(client, user.user_id);
        const { password: _, ...userWithoutPassword } = user;
        await client.query("COMMIT");
        return { user: userWithoutPassword, accessToken: accessToken, refreshToken: refreshToken };

    } catch (error: any) {
        await client.query("ROLLBACK");
        console.error("Error inside signUp:", error);
        throw error;
    } finally {
        await client.release();
    }
}
```

Figure 5. *signUpUser service method*

```ts
export async function createUser(client: any, name: string, role: string, email: string, password: string): Promise<User> {
    const result = await client.query(
        `INSERT INTO users (user_name, role, email, password)
         VALUES ($1, $2, $3, $4)
         RETURNING *`,
        [name, role, email, password]
    );
    return result.rows[0]
}
```

Figure 6. *createUser model method*

Every type of file in the backend is separated by use case. For example, routes relating to submissions are all in the `submission.route.ts` file. Separating directories by their purpose and files by their use case helped massively with my understanding of how server requests work and flow from front end to backend, to database and vice versa. Note that Figure 7 below also shows some extra directories such as `db` which contains a database dump for recreating the database. The `infrastructure` directory handles external connections to the PostgreSQL database and Docker.

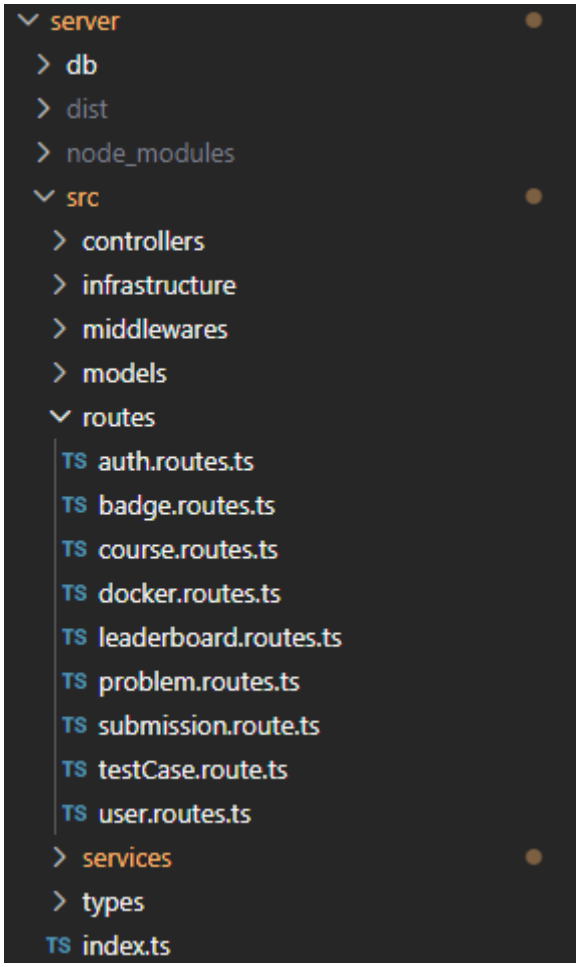

Figure 7. *Server directory structure*

Hosting

Another big challenge early in the project was figuring out how to host the platform. This was a complete unknown to me and required considerable research. My initial instinct was to go with a cloud-based platform like Amazon Web Services as it is an industry standard. However, from my research I learned about different types of hosting such as shared hosting, virtual private server (VPS) hosting, and dedicated hosting. In the end I chose to host the platform using Digital Ocean's Droplet (VPS) offering. This provided me with a balance of a developer-friendly user experience, affordable cost and enough resources to run the platform. I decided against using AWS as it would have required more time to set up and this would have taken precious time away from development.

One issue encountered during deployment was that the application worked correctly in the local development environment but failed to work when hosted. This was primarily due to port configurations. I soon learned I needed to use Nginx to serve my frontend files and proxy incoming requests to the web server port. To solve this, I needed to configure port mappings in an Nginx config file. I also learned I needed to set different environment variables for connecting to the hosted PostgreSQL database. I did this not through a .env file but by using PM2. PM2 is a process manager for Node.js applications installed globally on the host machine (Isaiah, 2026). After endlessly tweaking configuration files and setting the environment variables via PM2, the

platform could eventually be accessed via an IP address. Later in development when containerising the application environment variables were moved out to a .env file. This will be discussed further in a section on containerisation.

Overcoming this challenge taught me about reverse proxies, specifically Nginx and how it is configured and used to serve websites online and forward requests to a web server. This helped me later in development when I wanted to switch from connecting to my platform via an IP address to configuring a domain name. To achieve this, I needed to configure a Domain Name System (DNS) record to point the domain name to the IP address of my platform. I used Cloudflare as my DNS provider as it was the most affordable and easy to use. After configuring the DNS record, I updated the Nginx configuration by setting the `server_name` directive so that it would listen for incoming requests made to the domain and route them correctly to the application.

To summarise, through extensive research into hosting, I developed an understanding of the different types of hosting and their respective providers. I also learned how to configure a reverse proxy using Nginx, manage environment variables securely using PM2, and set up a domain name by configuring DNS records with Cloudflare and integrating this with the Nginx configuration. This significantly improved my understanding of how web applications are hosted and configured in production environments.

Docker Code Execution

The most technically challenging aspect of the project was using Docker to safely and securely execute student submitted code. This was important to get right because if submitted code could execute on the web server it could potentially give an attacker full control of the platform. Initially I had no experience with containerisation tools like Docker, so this was particularly challenging.

My first goal was to execute some Java code inside a Docker container and return a result. The first hurdle with this was learning how to make my application communicate with the Docker engine which was running locally. From my research I discovered an NPM package called `dockerode` which provided the necessary methods for interacting with the Docker engine instance (Dias, 2026). However, before I could start any containers, I needed a way to reference the Docker socket. I did this by using `dockerode` to define the Docker socket in code in a file called `docker.ts`. This allowed me to interact with the Docker instance running on my machine.

This taught me essential skills such as how to interact with an external service using a socket in TypeScript and about the invaluable `dockerode` package and its powerful methods. When the application was later containerised using Docker Compose, the host Docker socket was mounted into the server container, allowing it to spawn execution containers directly on the host machine. This is known as the Docker-outside-of-Docker (DooD) pattern (NestyBox, 2019).

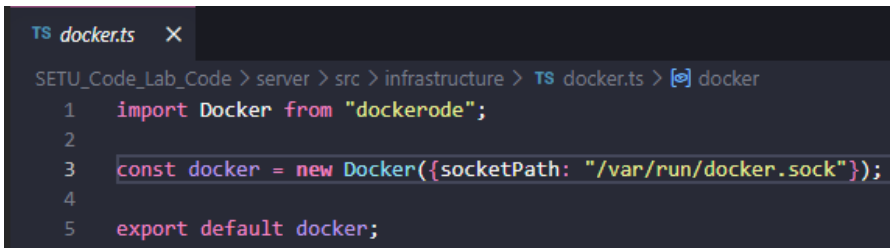

Figure 8. *Docker socket definition*

Custom Java Docker images needed to be created containing a Java JDK and essential Java packages for mapping sample inputs from test cases to parameters in submitted code functions. Initially I tried using a JDK by itself which worked for a simple proof of concept. However, when different inputs needed to be used with the same function I had to use the Java ***ObjectMapper*** object from the Jackson package. This allowed me to map sample JSON inputs from system.in to structured Java objects, which could then be passed as parameters into the user's function (FasterXML, 2026). This taught me how to create custom Docker images, how to tell Docker how to build them using a Dockerfile and how to include packages in a Docker container.

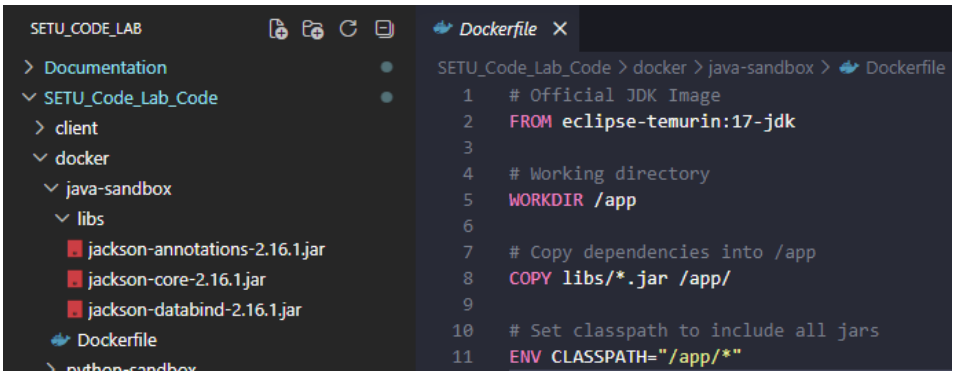

Figure 9. *Java code execution Dockerfile*

Java input functions needed to be pre-processed and placed into a suitable test harness allowing them to properly compile and execute using sample inputs. This test harness started as a static string into which only the submitted code was injected, however this was very inflexible and completely unsuitable. Through careful examination and prototyping on paper, I identified the areas of the test harness that needed to be dynamic (see Figure 10).

```
import com.fasterxml.jackson.databind.ObjectMapper;
import java.util.*;

public class Main {
    static final ObjectMapper mapper = new ObjectMapper();
    ${code}
    static class Input {
        ${inputFields}
    }
    public static void main(String[] args) {
        try {
            Input input = mapper.readValue(System.in, Input.class);
            ${functionCallLine}
            System.out.println(mapper.writeValueAsString(result));
        } catch (Exception e) {
            System.out.println("ERROR:" + e.getMessage());
        }
    }
}
`;
```

Figure 10. *Dynamic Java test harness*

After defining the test harness, I needed to go back and extract the needed data from the lecturer defined placeholder code. Most of this could be handled using regular expressions. One issue encountered later was the extraction of complex nested data types from placeholder code parameters. Parameters such as ***Map<String, Int>*** would break typical parameter splitting

methods that split on commas. To solve this, I measured the level of nesting using a variable called depth. This variable would be incremented when it detected an open bracket and decremented on any closing brackets. This allowed me to only split parameters based on top level commas (see Figure 11). Solving these issues helped me improve my critical thinking, code reasoning and problem-solving skills.

```
export function splitParams(paramsList: string): string[] {
  const params: string[] = [];
  let current = "";
  let depth = 0;

  for (let i = 0; i < paramsList.length; i++) {
    const c = paramsList[i];
    if (c === "<" || c === "[" || c === "{") depth++;
    if (c === ">" || c === "]" || c === "}") depth--;
    if (c === "," && depth === 0) {
      params.push(current.trim());
      current = "";
    } else {
      current += c;
    }
  }
  if (current.trim() !== "") {
    params.push(current.trim());
  }
  return params;
}
```

Figure 11. *splitParams function*

Once the code was prepared and inserted into the test harness, I needed a way to pass it to the Docker container to be executed. After examining the `dockerode` documentation for the `createContainer()` method I discovered it was possible to pass commands to the container (Docker, 2025). Figuring out the correct combination of shell commands took a lot of research. From this I learned some basic shell scripting. For example, how to use the cat command along with a ***heredoc*** to write data to a file. This allowed me to then compile and run Java code inside of the Docker container (see Figure 12).

```
cat << 'EOF' > Main.java
${preprocessedCode}
EOF
cat << 'ENDINPUT' > input.json
${processedInput}
ENDINPUT
javac Main.java
java -cp ".:/app/*" Main < input.json
`;
```

Figure 12. *Commands to compile and run Java code inside Docker container*

The security and resource parameters needed to be set correctly on the `createContainer()` method so that resource and network access were limited on the new container. This task proved simpler than others related to Docker as it was just a matter of setting parameters correctly. Most importantly, `NetworkMode` had to be set to "`none`" to prevent connection to external networks. Resource limits were also set to prevent the system from hanging in the event of an infinite loop or DDOS attacks (see Figure 13). I discovered how to do this via the Docker Engine API documentation (Docker, 2025).

```
const container = await docker.createContainer({
  Image: image,
  WorkingDir: "/app",
  Cmd: ["sh", "-c", cmd],
  Tty: false,
  AttachStdout: true,
  AttachStderr: true,
  HostConfig: {
    NetworkMode: "none",
    Memory: 256 * 1024 * 1024,
    CpuPeriod: 100000,
    CpuQuota: 50000,
    PidsLimit: 50,
  },
});
```

Figure 13. *Setting Docker container parameters*

After the previous steps were in place, starting and stopping the Docker container was easy. The next challenge would be handling the output from the Docker container after student code had been executed. Initially, I assumed that the output from the container would be returned as a simple string. However, this was not the case. Docker returns logs in a buffered format which includes additional metadata headers for each chunk of output. This meant that the raw output could not be used directly, as it contained extra bytes that would interfere with parsing and comparison.

To solve this, I implemented a function called `stripDockerHeader()` (see Figure 14) which iterates through the buffer removing the header information from each chunk. This leaves the original function output. I learned how to do this through looking at the Docker Documentation and the Node.js documentation where I discovered the `readUInt32BE()` method. This method reads an unsigned, 32-bit, big endian integer, after a specified offset from the buffer (node.js, 2026). This allowed me to deduce the start and end points of the payload in the buffer and thus extract it successfully. From my research I learned about big-endian and little-endian integers and how to work with buffers in TypeScript.

```
function stripDockerHeader(buffer: Buffer): string {
  let result = "";
  let i = 0;
  while (i < buffer.length) {
    const headerSize = 8;
    const payloadLength = buffer.readUInt32BE(i + 4);
    const payloadStart = i + headerSize;
    const payloadEnd = payloadStart + payloadLength;
    result += buffer.slice(payloadStart, payloadEnd).toString("utf8");
    i = payloadEnd;
  }
  return result.trim();
}
```

Figure 14. *stripDockerHeader method*

Implementing secure code execution through Docker was one of the most technically challenging aspects of my final year project as it was completely new to me. From persevering through these numerous challenges, I learned about containerisation, how Docker works, the Docker in Docker (DinD) and Docker Outside of Docker (DooD) patterns, shell scripting, buffer manipulation, big and little endian, and regular expression matching. I also exercised my code reasoning, and problem-solving skills. I am glad to have gained experience with Docker and shell

scripting as these skills are highly transferrable and sought after in the software development industry.

Docker Containerisation

Later in development I furthered my Docker skills by containerising the platform itself. This made it easier to run the platform locally on new machines and simplified the user manual. To do this the frontend and backend needed their own individual Dockerfiles. The frontend and backend Dockerfiles both use a multi-stage build process ensuring that only the minimum required artifacts and dependencies are included in the final production images. This enables containers to be lightweight and spin up quickly (Docker, n.d.).

The challenging part of this was setting up the database container and getting all the containers to communicate together as a cohesive application. Initially, I encountered issues with the backend failing to connect to the database due to incorrect host configuration and timing problems during startup. This required me to better understand Docker networking and implement service dependencies and health checks within the Docker Compose configuration.

To do this I had to create a ***docker-compose.yaml*** file (Docker, n.d.). Before containerisation my first solution stored database and JWT credentials internally using PM2, which is suitable for running standalone Node.js applications. The containerised solution required these secrets to be stored in a shared .env file ensuring consistency across all containers. These variables were then used in the Docker compose file to allow the web server container to connect to the database container.

Once the containers were communicating correctly, I soon ran into another issue. I could not create a new account. After reviewing the application code and the containerised database, I discovered this was because the database inside the Docker container was empty. SETU Code Lab requires the database to be seeded with certain data such as the ***global problems***, as all new accounts are automatically added to this course.

To resolve this, I implemented a database initialisation mechanism using Docker's ***/docker-entrypoint-initdb.d*** directory. This allows SQL scripts to be executed automatically when the container is first created. This ensured that the required schema and seed data were consistently applied whenever a new environment was set up (PostgreSQL, n.d.).

Generative Artificial Intelligence Considerations

One challenge was mitigating student use of generative AI without degrading the user experience through overly aggressive detection mechanisms.

It is difficult to block the use of generative AI outright, as students can simply use it on a separate device. SETU Code Lab aims to mitigate its use by increasing the work factor and achieves this by blocking copy and paste, drag and drop and detecting if a user switches tabs. When this is detected their code submission is automatically submitted (See the design document for more information on how this is implemented). This section looks at some of the challenges encountered and reflections associated with mitigating the use of AI.

Initially I wanted to detect when user switches focus from one tab to another using the ***blur*** browser event. This would detect if a user clicked off the current tab. However, certain notifications also trigger this event. This is not suitable as it would to unintentional automatic submissions, so I had to find a new way to balance user experience with the mitigation of AI.

Instead, I chose to use the ***hidden*** browser event which triggers when a tab is no longer visible (CodeStudy.net, 2025). While this is not as strict as the ***blur*** event, it allows me to deter some students from switching between multiple tabs.

Admittedly, this solution is weak as tabs can be opened on a different screen or beside the current screen before a student clicks into a problem. Upon reflection, I should have logged student tab switching behaviour using the ***blur*** event. This information could then be made available to lecturers so that they could decide if a student is switching tabs suspiciously and their marks could be adjusted accordingly. This idea was not implemented because it was discovered too late in development when time had already been committed to other features.

Authentication

Another challenge I encountered mid-way through development was a JWT (JSON Web Token) expiry issue. When a user logs in to SETU Code Lab a session token is issued with an expiry time of three hours. During testing I noticed that after this three-hour mark requests would fail without notifying the user or logging them out. This was a major problem for the platform as a user may often take longer than three hours to complete a problem. As a result, when they try to submit their solution, their request will fail silently, and their progress will be lost. At first, I thought maybe I could set the token expiry to twenty-four hours, but this is not a secure solution. Logging out users was also not an option as it would degrade the user experience. After researching what similar websites do, a suitable solution was discovered.

I implemented a second refresh token with an expiry of seven days. When a user logs in, they receive a JWT token for authentication and a second JWT token for refreshing. The authentication token is validated on every request to ensure the user is allowed to make requests to protected endpoints on the web server. When this token expires after three hours, the refresh token is also validated. If the refresh token is still valid the user receives a new authentication token (see Figure 15). This way there is no interruption to the user and the platform remains secure (GeeksforGeeks, 2025). Note that these tokens are stored securely in HTTP-Only cookies to prevent the possibility of XSS (Cross-Site Scripting) attacks (CodeStudy, 2025). This taught me how to implement a secure, industry-standard authentication process, how to use JWT tokens correctly and how to balance security with the user's experience.

```
export function refreshToken(req: Request, res: Response) {
    const token = req.cookies.refreshToken;
    if (!token) return res.status(401).json({ message: "No refresh token" });

    try {
        const decoded = jwt.verify(token, process.env.JWT_REFRESH_SECRET!) as { user_id: string };

        const newAccessToken = jwt.sign(
            { user_id: decoded.user_id },
            process.env.JWT_SECRET!,
            { expiresIn: "3h" }
        );

        res.cookie("token", newAccessToken, {
            httpOnly: true,
            maxAge: 3 * 60 * 60 * 1000
        });

        return res.status(200).json({ message: "Token refreshed" });

    } catch (error) {
        return res.status(401).json({ message: "Invalid refresh token" });
    }
}
```

Figure 15. *refreshToken method assigning new refresh token in auth.service.ts*

Reflection

What I Achieved

SETU Code Lab successfully implements all core and non-core requirements outlined in the functional specification. Additionally, Python code execution, which was originally considered a stretch goal, was implemented in just two days after solving the challenge of Java execution. The UI design process also went smoothly due to prior React experience and the use of PenPotUI (open-source web design tool) for rapid prototyping. Containerising the application into Docker containers proved to be successful also and allowed the user manual and local platform setup to be simplified.

I am particularly proud of the user interface as I like its design, and some users have commented on its ease of use. I am also very proud of problem creation, as this was one of the hardest sections to get right. This was in part because I had to convert lecturer test case inputs to JSON before saving them to the database and convert them back to a human readable format when displaying a problem. This allowed me to easily manipulate the data when I needed to run it with a student problem, while also keeping the user interface clean (as they would not have to enter/see test inputs and outputs in JSON format).

The problem creation screen also had some very uninformative error messages when lecturers defined problems incorrectly. This was solved by compiling and running lecturer problems using their own test cases before saving them to the database. This allowed me to take advantage of the error messages provided by Java which lecturers may be more familiar with, further improving the user experience.

What I did not Achieve

Several features were cut or simplified due to time constraints: execution-time leaderboard rankings became points/streak-based, a monthly ranking system became date filters, and tag-

based filtering became difficulty filters. This taught me to estimate feature complexity more accurately, better prioritisation, and adapt requirements into simpler alternatives when complex database changes were not accounted for.

What I Learned

The most valuable technical skill I gained was learning how to structure the backend architecture. Early in the project the backend started as mixed concerns until I adopted the route-controller-service-model pattern, which reduced cognitive load. When I returned to authentication code weeks later to add refresh tokens, I could navigate it instantly because each layer had a single, clear purpose.

Learning how to containerise the application also proved highly valuable, particularly in simplifying deployment. Prior to this, setting up the application locally, required a long and error-prone manual process. By introducing Dockerfiles for both the frontend and backend, along with a Docker Compose configuration to manage service communication, I was able to standardise the environment setup. This reduced the setup process to a small number of commands and eliminated inconsistencies between development environments.

The defects that arose close to the end of development taught me the true value of early feedback and testing. Java code execution proved to be very slow and should have been detected and fixed much sooner in the development process. Some defects such as menus stretching off screen, black text appearing on a dark background and strange errors with validating lecturer inputted test cases were discovered very late in development and almost made it to the final release. They were discovered because of feedback from a user. This close call taught me the value of gaining feedback early on in development.

What I would do Differently

One shortcoming of the project is the length of time it takes to execute Java problems. This was pointed out by several users who tested the platform. Test case TC-NF05a states "*The system shall return code execution results in ≤ 30 seconds for Java problems ≤100 lines of code and ≤10 test cases*". In production, during testing the system took 27.43 seconds. While this passes the test, this test case was far too lenient, and it should be much faster. This test should have been reduced to a maximum of 10 seconds, conducted much sooner and fixed accordingly. I underestimated the performance impact until user testing, by which point I'd committed time to other features. The poor performance is due to slow JVM startup and fresh containers being created for every test case. The solution would be to optimize this by pre-loading containers containing the JVM to eliminate the overhead from spinning up a new container for every single test case.

This problem would have been mitigated by testing earlier and more often in the development lifecycle. If I had to undertake this project again from the beginning, I would firstly track testing in a table in Excel from the beginning and test each feature thoroughly before moving on to the next. By testing each feature sooner, I could have uncovered the Java execution performance issue and fixed it earlier before the code became too complex and harder to work with.

I would also try to get feedback much earlier in the development life cycle from users and in a more structured manner. Feedback should have been gathered from multiple Lecturers and Students after developing each feature, by giving them a short task and survey. This way I could track feedback, develop more useful and usable features for my users and include it in my final report. Some bugs with the UI were only discovered after late feedback sessions. Some users

required clarification on how to use the problem creation screen to create test cases. Through proper early feedback, I could have identified this issue and made the UI easier to use.

If I could do this project again, I would log student behaviour around switching tabs early on, to give lecturers more information for grading students. This would also work to mitigate students using AI as I felt this was a weak point in the project. I would also take more personal notes over the course of the project to track challenges encountered and learnings. This would help me improve my documentation.

Testing

To test SETU Code Lab's success and reliability as a software product, ***User Acceptance Testing (UAT)*** was performed for each use case. These are black box tests which verify the functional requirements outlined in the Functional Specification from the user's perspective. As it was not possible to find a user with adequate time to perform User Acceptance Testing, this testing was conducted by me from the perspective of the user. Separate tests were also carried out for non-functional requirements.

The first table, seen on the following pages shows the functional tests that were performed. These tests map directly to the core and non-core functional requirements outlined in the Functional Specification. In this table, ***TID*** refers to the test case ID and ***RID*** refers to the requirement ID, with ***FR*** corresponding to Functional Requirements and ***NFR*** corresponding to Non-Core Functional Requirements. ***TC11e*** resulted in the discovery of a defect where user streaks were not updating every time a user made a submission. A user's current streak would only update when they made a successful submission. This defect has since been fixed, the test re-executed, and the test case updated in the table.

The next table starting on page 27 shows the non-functional tests that were performed. These tests map directly to the non-functional (FURPS+) requirements outlined in the Functional Specification. In this table user tests such as ***TC-NF02*** and ***TC-NF03*** were carried out by actual users and not the developer. Test case ***TC-NF08d*** revealed a user interface scrolling issue on the ***manageProblems***, and ***manageClasses*** page. This test involved testing the platform's user interface on a smaller screen size (1280×720). This issue was fixed and retested successfully. Similarly, test case ***TC-NF08e*** also revealed an issue where the test case entry form on the ***createProblem*** page was too small on larger screen sizes (2560×1440). This issue was also fixed and retested successfully.

Functional Testing

TID	RID	DESCRIPTION	STEPS	EXPECTED RESULT	ACTUAL RESULT	P/F
TC01	FR1	The system shall allow students to browse a list of coding problems	1. Log in as student 2. Navigate to homepage	List of problems visible with corresponding title, author and difficulty	Problems visible as expected	**PASS**
TC02	FR2	The system shall provide a built-in code editor for writing solutions	1. Log in as student 2. Select a problem	Code editor is visible, displays placeholder code and is editable	Code editor visible, placeholder code visible and editor is editable	**PASS**
TC03a	FR3	The system shall execute correct inputted code against predefined test cases	1. Log in as student 2. Select a problem 3. Input a correct solution 4. Click submit	System executes code against all predefined test cases, results are displayed on the screen, and a submission summary is displayed on the screen	System executes code against all predefined test cases, results are displayed on the screen, and a passing submission summary is displayed on the screen	**PASS**
TC03b	FR3	The system shall execute incorrect inputted code against predefined test cases, return errors and result in a failed submission	1. Log in as student 2. Select a problem 3. Input an incorrect solution 4. Click submit	System executes code against all predefined test cases, errors are displayed on the screen, and a submission summary is displayed on the screen	System executes code against all predefined test cases, errors are displayed on the screen, and a failed submission summary is displayed on the screen	**PASS**
TC04a	FR4	The system shall indicate if a student has passed a test case	1. Log in as student 2. Select a problem 3. Input a correct solution 4. Click run	System shows the expected output and the actual output for each test case in the test case window and displays PASS	System shows the expected output and the actual output for each test case in the test case window and displays PASS	**PASS**

TC04b	FR4	The system shall indicate if a student has failed a test case	1. Log in as student 2. Select a problem 3. Input an incorrect solution 4. Click run	System shows the expected output and the actual output for each test case in the test case window and displays FAIL	System shows the expected output and the actual output for each test case in the test case window and displays FAIL	**PASS**
TC05	FR5	The system shall save student submissions to their profile	1. Log in as student 2. Select a problem 3. Input a solution 4. Click submit 5. Click profile 6. Click the submission under Submission History	System displays the problem title, pass/fail, and the date and time at which the problem was submitted on the student's profile. System also additionally displays test case results, points breakdown and submitted solution on the submission record itself	System displays the problem title, pass/fail, and the date and time at which the problem was submitted on the student's profile. System also additionally displays test case results, points breakdown and submitted solution on the submission record itself	**PASS**
TC06	FR6	The system shall allow lecturers to upload new problems and test cases	1. Log in as lecturer 2. Click lecturer menu > manage problems > create new problem 3. Enter problem details in required format 4. Add valid test cases 5. Click save 6. Click lecturer menu > manage courses 7. Click edit icon on global problems 8. Add new problem to global problems	System should display newly created problem on the homepage while the global problems course is selected. Upon clicking this problem, all problem details should be visible, and the problem should be able to be completed.	System displays newly created problem on the homepage while the global problems course is selected. Upon clicking this problem, all problem details are visible, and the problem can be completed.	**PASS**

TC07	FR7	The system shall allow lecturers to create courses and assign problems to groups of students	1. Log in as a lecturer 2. Click lecturer menu > manage courses 3. Click create new course 4. Fill in course details adding students and problems 5. Click save 6. Log out 7. Log in as each student added to the course 8. Click courses and select the course that was just created	System should display the display the problems that were added to the new course by the lecturer account, for the student accounts that were added to the course	System displays the problems that were added to the new course by the lecturer account, for the student accounts that were added to the course	**PASS**
TC08a	FR8	The system shall allow lecturers to see results and solutions from students which they have assigned problems to	(prerequisite: problems have been submitted by student accounts enrolled in courses under test) 1. Log in as a lecturer 2. Click lecturer menu > manage courses 3. Click on a global problems 4. Observe results 5. Click on individual result 6. Observe student results and solution for problem	System should display a percentage result in a table format for every student enrolled in global problems and every problem. The system should also navigate to the individual student submission upon clicking its corresponding cell in the table. A grey dash should be visible for students that have not made any submissions yet for a particular problem after this course was created.	System displays percentage results in a table format for every student enrolled in global problems and every problem. The system also navigates to the individual student submission upon clicking its corresponding cell in the table. A grey dash is visible for students that have not made any submissions yet for a particular problem after the course was created.	**PASS**

TC08b	FR8	The system shall allow lecturers to download results for global problems and any course they have created	(prerequisite: problems have been submitted by student accounts enrolled in courses under test) 1. Log in as a lecturer 2. Click lecturer menu > manage courses 3. Click on global problems or a course created by this lecturer account 4. Click Download Results.csv 5. Click on downloaded .csv file	System should download a .CSV file entitled <course-name>_results.csv. This file should contain a table of all student results, totals across multiple problems and averages across multiple students.	System downloads a .CSV file entitled <course-name>_results.csv. This file contains a table of all student results, totals across multiple problems and averages across multiple students.	**PASS**
TC09a	NFR1	The system should allow all users to search and filter for problems with a specific name	(prerequisite: several problems of different difficulty and name have been added to the global problems course) 1. Log in as any user type 2. Observe the displayed problems 3. Enter a search query corresponding to one of the problem titles 4. Observe result 5. Repeat test for other user type	The system should only show problems whose titles contain the search query. All other problems should be hidden.	The system only shows problems whose titles contain the search query. All other problems are hidden.	**PASS**

TC09b	NFR1	The system should allow all users to filter problems by difficulty	(prerequisite: several problems of different difficulty and name have been added to the global problems course) 1. Log in as any user type 2. Observe the displayed problems 3. Select the Hard difficulty filter 4. Observe result 5. Repeat test for other user type	The system should only display problems with a difficult rating of 4 or 5 stars.	The system only displays problems with a difficult rating of 4 or 5 stars.	**PASS**
TC10a	NFR2	The system should provide a leaderboard ranking students on points earned	(prerequisite: several student accounts have completed problems on the global problems course and have different scores) 1. Log in as a student 2. Navigate to Leaderboard 3. Ensure the global problems course is selected, all time is selected and score is selected 4. Observe ranking of students	The system should display a table ranking students enrolled in global problems by total score descending. The name, current streak and total score of each student should be displayed.	The system displays a table ranking students enrolled in global problems by total score descending. The name, current streak and total score of each student is visible.	**PASS**

TC10b	NFR2	The system should provide a leaderboard ranking students based on their current streak	(prerequisite: several student accounts have completed problems on the global problems course and have different scores) 1. Log in as a student 2. Navigate to Leaderboard 3. Ensure the global problems course is selected, all time is selected and current streak is selected 4. Observe ranking of students	The system should display a table ranking students enrolled in global problems by current streak descending. The name, current streak and total score of each student should be displayed.	The system displays a table ranking students enrolled in global problems by current streak descending. The name, current streak and total score of each student is visible.	**PASS**
TC11a	NFR3	The system should incorporate gamification elements such as daily log in streaks	1. Create a new student account 2. Select and complete a problem correctly 3. Observe current streak of 1 on the profile page 4. set last_solved_date in the users table to be -1 day for this user 5. Complete another problem correctly 6. Observe current streak increase to 2 on the profile page	The system should increase the student's streak to 2. This should be visible on the student account's profile.	The system increases the student's streak to 2. This is visible on the student account's profile.	**PASS**

TC11b	NFR3	The system should award a badge when a student completes their first problem	1. Create a new student account 2. Select and complete one problem correctly 3. Observe the "First Steps" badge in the submission alert 4. Navigate to profile 5. Observe the "First Steps" badge on the profile page	The system should display the "First Steps" badge on the submission alert and on the student's profile.	The system displays the "First Steps" badge on the submission alert and on the student's profile.	**PASS**
TC11c	NFR3	The system should award a badge when a student achieves a 3-day streak	1. Create a new student account 2. Achieve a 3-day streak 3. Upon completing the third problem observe the "On Fire" badge in the submission alert 4. Navigate to profile 5. Observe the "On Fire" badge on the profile page	The system should display the "On Fire" badge on the submission alert and on the student's profile.	The system displays the "On Fire" badge on the submission alert and on the student's profile.	**PASS**
TC11d	NFR3	The system should award a badge when a student completes 5 problems	1. Create a new student account 2. Complete 5 problems correctly 3. Upon completing the fifth problem observe the "Problem Solver" badge in the submission alert 4. Navigate to profile 5. Observe the "Problem Solver" badge on the profile page	The system should display the "Problem Solver" badge on the submission alert and on the student's profile.	The system displays the "Problem Solver" badge on the submission alert and on the student's profile.	**PASS**

TC11e	NFR3	The system should reset current streak after any submission if the user did not complete a different problem on the previous day	1. Log in as a student 2. Successfully submit a new correct problem to increase current streak to one 3. Set last_solved_date to 2 days in the past 4. Successfully submit a new incorrect problem 5. Observe the student account's current streak on the profile page	The system should reset the student's current streak to 0, this should be visible on the student's profile.	Student current streak is 0 on the profile page.	**PASS**

Non-Functional Testing

TID	CATEGORY	DESCRIPTION	STEPS	EXPECTED RESULT	ACTUAL RESULT	P/F
TC-NF01	Functionality	The system shall support at least one programming language (Java) for code execution.	1. Log in as student 2. Select a Java problem 3. Ensure Java is selected 4. Write valid Java solution 5. Click run	Code executes and test case results are returned with no language error	Code executes and test case results are returned with no language error	**PASS**
TC-NF02	Usability	Students with existing accounts shall be able to log in, select a problem, write some code, and submit it with no prior training in ≤ 90 seconds.	1. Provide student with login credentials 2. Start timer 3. Student logs in, selects any problem, writes minimal valid code and clicks Submit 4. Stop timer on submission confirmation	Full flow completed in ≤ 90 seconds	Full flow completed in 62 seconds	**PASS**
TC-NF03	Usability	Lecturers with existing accounts should be able to log in, create a new basic code problem, and upload it with no prior training in ≤ 5 minutes.	1. Provide lecturer with login credentials 2. Start timer 3. Lecturer logs in, creates a basic code problem (title, description, ≥1 test case) and publishes it 4. Stop timer on successful upload	Full flow completed in ≤ 5 minutes (300 seconds)	Full flow completed in	
TC-NF04	Reliability	The system shall have 99% uptime	1. Set up an uptime check on Digital Ocean 2. Observe uptime	Uptime ≥ 99%	Uptime is 99.69%	**PASS**

TC-NF05a	Performance	The system shall return code execution results in ≤ 30 seconds for Java problems ≤100 lines of code and ≤10 test cases.	1. Log in as student 2. Select a Java problem with ≤100 lines of solution code and ≤10 test cases 3. Submit a valid solution 4. Start timer on click of Submit 5. Stop timer when results are fully displayed	Results displayed within 30 seconds of submission	Results displayed after 27.43 seconds	**PASS**
TC-NF05b	Performance	The system shall return code execution results in ≤ 30 seconds for Python problems ≤100 lines of code and ≤10 test cases.	1. Log in as student 2. Select a Python problem with ≤100 lines of solution code and ≤10 test cases 3. Submit a valid solution 4. Start timer on click of Submit 5. Stop timer when results are fully displayed	Results displayed within 30 seconds of submission	Results displayed after 3.71 seconds	**PASS**
TC-NF06	Security	The system shall ensure submissions cannot access the network.	1. Log in as student 2. Submit code that attempts an outbound network request (e.g. HTTP GET to external URL) 4. Observe results	No network response data is returned to the user	No network response data is returned to the user	**PASS**

TC-NF07	Security	The system shall ensure that only authorized users (lecturers and students) can access their respective dashboards.	1. Attempt to access the student pages via URL without logging in 2. Attempt to access the lecturer pages via URL without logging in 3. Log in as a student and attempt to access lecturer pages via URL	Steps 1–2: Redirected to login page. Step 3: Access denied / redirected (student cannot reach lecturer dashboard)	Steps 1–2: Redirected to login page. Step 3: Access denied / redirected (student cannot reach lecturer dashboard)	**PASS**
TC-NF08a	Supportability	The system shall remain usable and readable at 1920×1080 (most common European desktop resolution).	1. Set browser viewport to 1920×1080 2. Inspect all pages for text overflow, overlapping elements, or broken layouts	No layout defects observed at this resolution	No layout defects observed at this resolution	**PASS**
TC-NF08b	Supportability	The system shall remain usable and readable at 1536×864.	1. Set browser viewport to 1536×864 2. Inspect all pages for layout defects	No layout defects observed at this resolution	No layout defects observed at this resolution	**PASS**
TC-NF08c	Supportability	The system shall remain usable and readable at 1366×768.	1. Set browser viewport to 1366×768 2. Inspect all pages for layout defects	No layout defects observed at this resolution	No layout defects observed at this resolution	**PASS**
TC-NF08d	Supportability	The system shall remain usable and readable at 1280×720.	1. Set browser viewport to 1280×720 2. Inspect all pages for layout defects	No layout defects observed at this resolution	No layout defects observed at this resolution	**PASS**

TC-NF08e	Supportability	The system shall remain usable and readable at 2560×1440.	1. Set browser viewport to 2560×1440 2. Inspect all pages for layout defects	No layout defects observed at this resolution	No layout defects observed at this resolution	**PASS**
TC-NF09	Compatibility	The system shall be supported on the latest version of Firefox.	1. Open the application in the latest release of Firefox 2. Complete the full student flow (login > select problem > write code > submit) 3. Complete the full lecturer flow (login > create problem > publish) 4. Check for console errors or visual defects	All flows complete without errors. No browser-specific defects	All flows complete without errors. No browser-specific defects	**PASS**
TC-NF10	Compatibility	The system shall be supported on the latest version of Google Chrome.	1. Open the application in the latest release of Chrome 2. Complete student and lecturer flows 3. Check for console errors or visual defects	All flows complete without errors. No browser-specific defects	All flows complete without errors. No browser-specific defects	**PASS**
TC-NF11	Compatibility	The system shall be supported on the latest version of Microsoft Edge.	1. Open the application in the latest release of Edge 2. Complete student and lecturer flows 3. Check for console errors or visual defects	All flows complete without errors. No browser-specific defects	All flows complete without errors. No browser-specific defects	**PASS**

Conclusion

SETU Code Lab successfully delivers a coding practice tool and learning management platform that meets all the core requirements outlined in the Functional Specification.

The biggest technical challenge of this project was the implementation of secure Java and Python code execution using isolated Docker containers. The hardest part of this was not the Docker connection or spinning up the containers themselves, but rather the preprocessing of submitted Java code and submitted placeholder code before they are sent to a container for execution. It took me a while to figure out how to structure the dynamic Java test harness, extract the needed data, and read JSON test inputs into the Java code.

My biggest technical oversight was not pre-loading and reusing Docker containers for Java test case execution. This taught me the importance of conducting non-functional testing and feedback sessions earlier in the development lifecycle so that key performance issues could be detected and fixed. If I had detected this performance issue early, I could have implemented a pool of Docker containers with JVMs pre-loaded. This would have eliminated the slow spin up time of the Docker containers on every test case run.

The most valuable skill I have learned from this project is how to structure a backend into route, controller, service and model files. This helped me understand how a web server handles requests and taught me a highly valuable backend pattern that ensures code is maintainable and extensible. Combined with Docker, hosting, and deployment experience, I'm now confident I can build and ship higher quality production grade software.

Bibliography

CodeStudy.net, 2025. *How to Detect Browser Tab Activity: Check If User Switched Tabs (Not Just Minimized) with JavaScript*. [Online]
Available at: https://www.codestudy.net/blog/detect-if-browser-tab-is-active-or-user-has-switched-away/#the-page-visibility-api-a-modern-solution
[Accessed 18 April 2026].

CodeStudy, 2025. *Where to Store Refresh Tokens in a SPA? Secure Client-Side Storage Best Practices*. [Online]
Available at: https://www.codestudy.net/blog/where-to-store-the-refresh-token-on-the-client/
[Accessed 6 April 2026].

Dias, P., 2026. *NPM - dockerode*. [Online]
Available at: https://www.npmjs.com/package/dockerode
[Accessed 05 April 2026].

Docker, 2025. *Docker Engine API*. [Online]
Available at: https://docs.docker.com/reference/api/engine/version/v1.52/
[Accessed 5 April 2026].

Docker, n.d. *Docker Compose | Docker Docs*. [Online]
Available at: https://docs.docker.com/compose/
[Accessed 20 April 2026].

Docker, n.d. *Multi Stage | Docker Docs*. [Online]
Available at: https://docs.docker.com/build/building/multi-stage/
[Accessed 20 April 2026].

FasterXML, 2026. *jackson-databind 2.21.2 javadoc*. [Online]
Available at: https://javadoc.io/doc/com.fasterxml.jackson.core/jackson-databind/latest/index.html
[Accessed 18 April 2026].

GeeksforGeeks, 2025. *JWT Authentication With Refresh Tokens*. [Online]
Available at: https://www.geeksforgeeks.org/node-js/jwt-authentication-with-refresh-tokens/
[Accessed 6 April 2026].

Gore, A., 2022. *treblle*. [Online]
Available at: https://treblle.com/blog/egergr
[Accessed 5 April 2026].

Isaiah, A., 2026. *How to Configure Nginx as a Reverse Proxy for Node.js Applications*. [Online]
Available at: https://betterstack.com/community/guides/scaling-nodejs/nodejs-reverse-proxy-nginx/
[Accessed 20 April 2026].

NestyBox, 2019. *Secure Docker-in-Docker with System Containers*. [Online]
Available at: https://blog.nestybox.com/2019/09/14/dind.html
[Accessed 06 April 2026].

node.js, 2026. *Node.js API*. [Online]
Available at: https://nodejs.org/api/buffer.html#bufreadint32beoffset
[Accessed 06 April 2026].

PostgreSQL, n.d. *Postgres Official Image | Docker Hub*. [Online]
Available at: https://hub.docker.com/_/postgres
[Accessed 20 April 2026].