SETU Code Lab
Design Document

Diarmuid O'Neill

South East Technological University

19/04/2026

Table of Contents

Table of Figures 4
Introduction 6
Hosting 6
Architecture Diagram 7
Sequence Diagrams 9
- Solve a Problem 9
- Login 10
- Sign Up 10
- View Problems 11
- Create Problem / Test Cases 11
- Update Problem / Test Cases 12
- Create / Update Course 12
- View Results 13
- View Profile 13
- View Leaderboard 14

Packages 15
Important Algorithms 16
- Test Harnesses for Running Submitted Code 16
- Preprocessing Java Input 17
- Preprocessing Python Input 19
- Execution and Judging 19
- Approach to the use of Artificial Intelligence 23

Entity Relationship Diagram 24
- Database Design Notes 25

SQL Queries 26
- Login / Sign Up 26
- View Problems 26
- Solve Problem 26
- CRUD Problem 27
- CRUD Test Case 28

CRUD Course 28
View Results 30
View Profile 31
Leaderboard 31
Multi-Language Support 33
Badges 33
User Interface 34
Logo 34
High-Level UI Flow 35
Login 36
Sign Up 36
View Problems 37
View Problem > Solve Problem 37
View Problem > Solve Problem >Submission Alert 38
View Problem > Manage Problems 38
View Problem > Manage Problems > Create/Update Problem 39
View Problem > Manage Courses 39
View Problem > Manage Courses > Create/Update Course 40
View Problem > Manage Courses > View Course 40
View Problem > Manage Course > View Course > View Result 41
View Problem > Profile 41
View Problem > Leaderboard 42
Bibliography 43

Table of Figures

Figure 1. Architecture Diagram 7
Figure 2. Solve a Problem sequence diagram 9
Figure 3. Login sequence diagram 10
Figure 4. Sign Up sequence diagram 10
Figure 5. View Problems sequence diagram 11
Figure 6. Create Problem / Test Cases sequence diagram 11
Figure 7. Update Problem / Test Cases sequence diagram 12
Figure 8. Create / Update Course sequence diagram 12
Figure 9. View Results sequence diagram 13
Figure 10. View Profile sequence diagram 13
Figure 11. View leaderboard sequence diagram 14
Figure 12. Java test harness 16
Figure 13. Python test harness 16
Figure 14. Signature regex which extracts details from placeholder code 17
Figure 15. splitParams function 17
Figure 16. parameters being parsed 18
Figure 17. functionCallLine built dynamically 18
Figure 18. preprocessPythonInput function 19
Figure 19. startContainer function part one 20
Figure 20. startContainer function part two, commands that run in Docker container . 20
Figure 21. startContainer function part three, container execution and timeout 21
Figure 22. startContainer function part four, processing output and returning data 22
Figure 23. deepEquals function 22
Figure 24. AntiCheat.ts file to deter users from using generative AI 23
Figure 26. Entity Relationship diagram 24
Figure 27. Dynamic SQL statement, defines how leaderboard will be displayed 32
Figure 28. Logo 34
Figure 29. User Interface flow diagram 35
Figure 30. Login screen design 36
Figure 31. Sign Up screen design 36
Figure 32. View Problems screen design 37
Figure 33. Solve a Problem screen design 37
Figure 34. Submission alert design 38
Figure 35. Manage Problems screen design 38
Figure 36. Create / Update Problem screen design 39
Figure 37. Manage Courses screen design 39
Figure 38. Create / Update Course screen design 40
Figure 39. View Course screen design 40
Figure 40. View Result Screen design 41

Figure 41. View Profile screen design 41
Figure 42. Leaderboard screen design 42

Introduction

The purpose of this document is to outline the proposed design for SETU Code Lab. It will explore how each part of the project is intended to be implemented. It includes sections on hosting, sequence diagrams, important algorithms, important packages, database design and the user interface.

SETU Code Lab is an in-browser study tool intended to be used by students enrolled in computing related courses at SETU and their lecturers. Students can select and complete problems and view their results and statistics on their profile. Lecturers can create problems, and courses which allow them to assign problems to groups of students and easily track their progress. This will help students prepare for upcoming exams and coding interviews and will also help lecturers easily conduct and grade lab work. The platform will also use some gamification mechanics to keep students engaged.

Java has been selected as the first supported language for SETU Code Lab as it is the first language that new students are exposed to at SETU. Support for more languages such as Python or Go may be added in the future if time allows.

Hosting

The system will be hosted using DigitalOcean's Droplet service. This is a virtual private server (VPS). The $12 per month regular option provides 2GBs of RAM, 1 CPU, 50GB of SSD storage and 1TB of bandwidth which should be enough for SETU Code Lab with some optimization. The chosen operating system for this server is Ubuntu 24.04 as it is very stable, provides excellent Docker support and is familiar to the developer.

For security purposes an SSH key will be generated and used to access the server console. This is faster and more secure than the password option that is offered by DigitalOcean. The Github repository containing the project will be cloned onto the server and the needed dependencies will be installed such as Node.js, Node Package Manager (NPM) and Nginx for serving the frontend.

Architecture Diagram

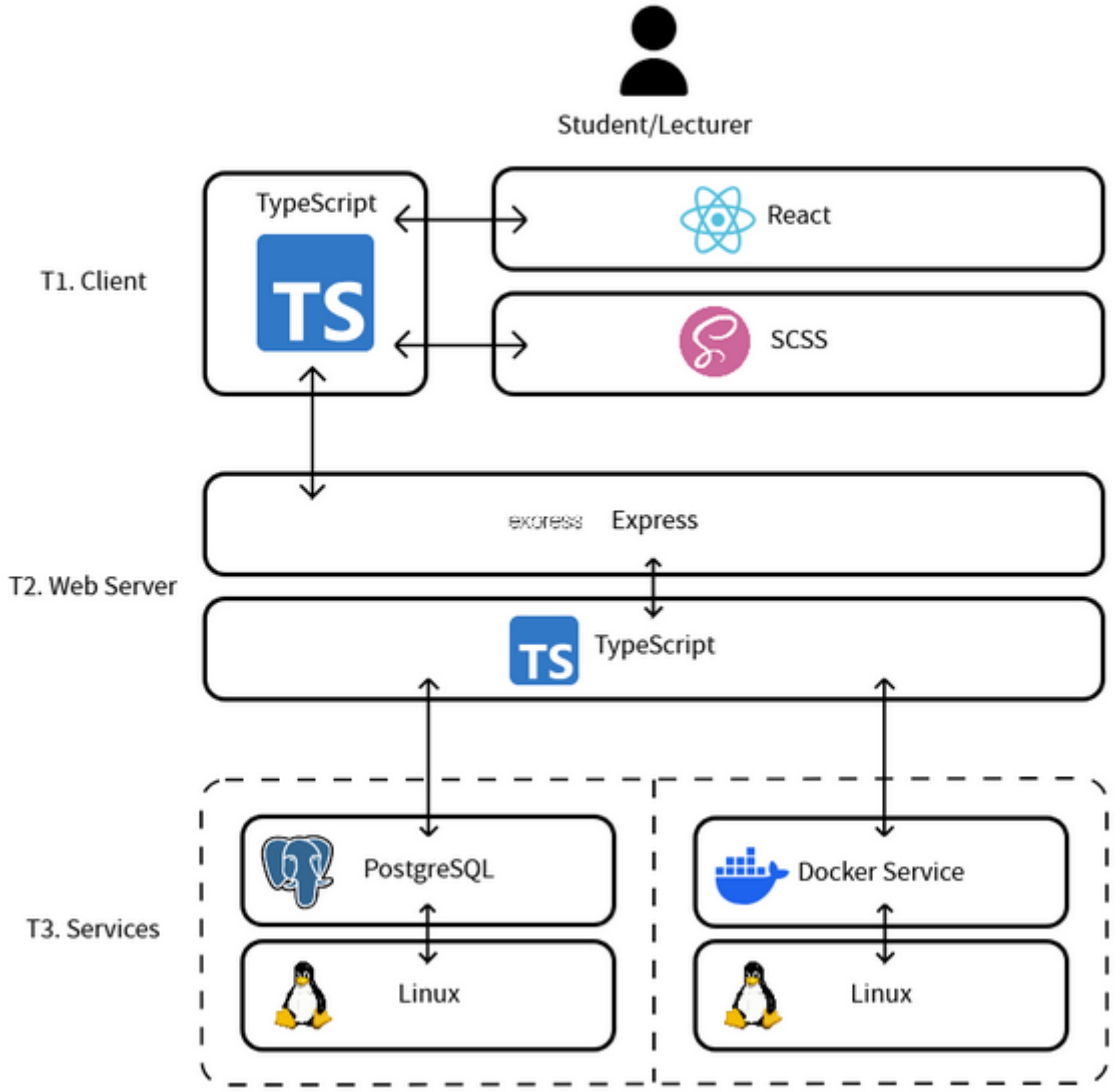

Figure 1. *Architecture Diagram*

SETU Code Lab uses a three-tier architecture. Tier one (T1) is the client and is built using React, Typescript and SCSS. This is where the user interface resides with everything required for each screen separated out into its own directory.

Tier two (T2) is the web server and is implemented using Node.js (this is a JavaScript runtime which allows TypeScript to run on a web server) and Express.js which is a Node.js framework used to simplify the routing process. The web server is structured into six layers: routes, middleware, controllers, services, models, and types. Routes define the API endpoints and direct incoming requests to the appropriate controller. Middleware handles cross-cutting concerns such as authentication and request parsing before requests reach the controller. Controllers process the incoming request and coordinate

the response. Services contain the core logic and interact with the models. Models handle direct communication with the PostgreSQL database. Types define the TypeScript interfaces and types used across the server.

Tier three (T3) is the services layer and consists of a PostgreSQL database which stores all the application data and the Docker service which is used to create temporary containers in which user submitted code runs in isolation.

Sequence Diagrams

Solve a Problem

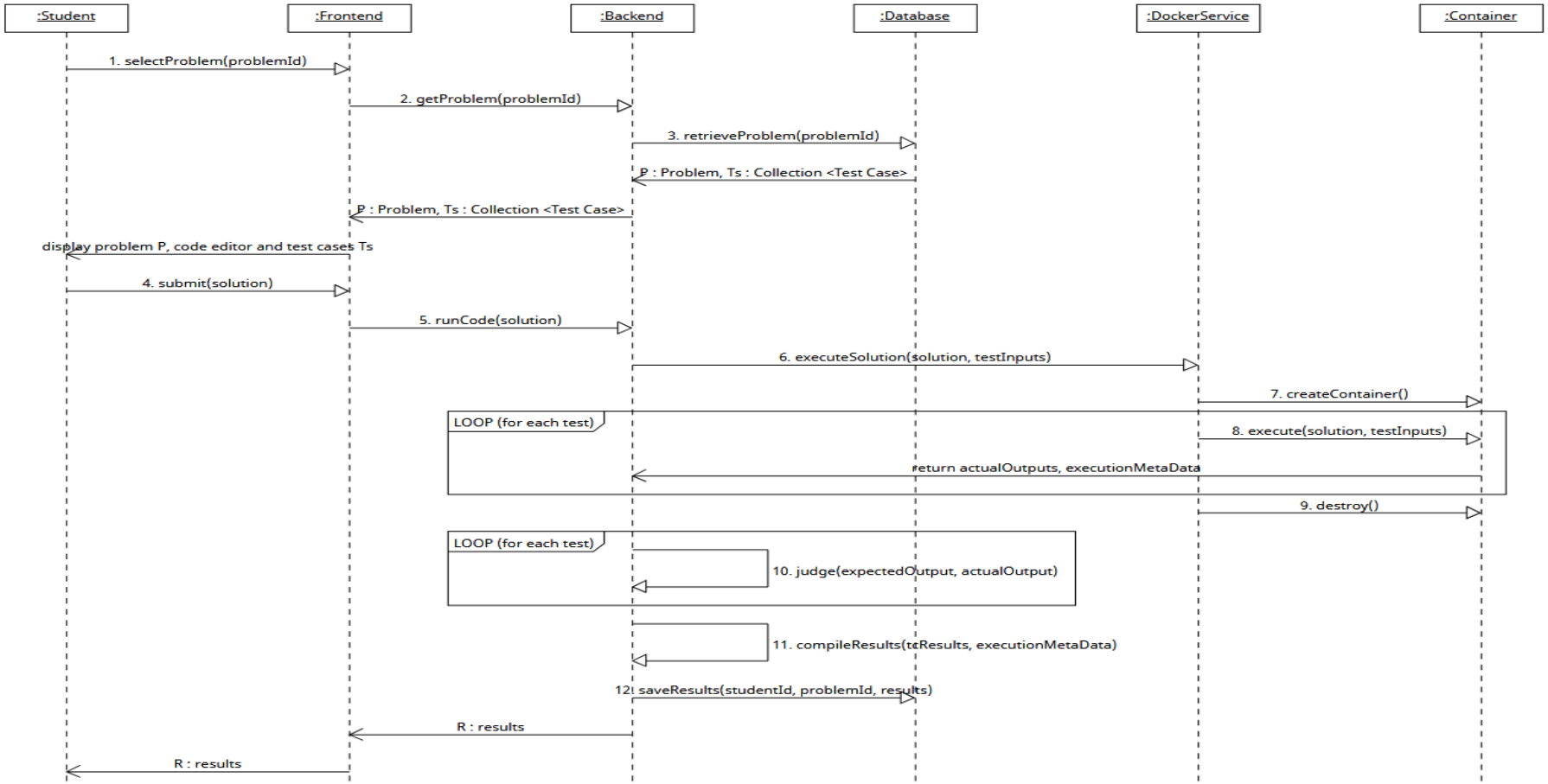

Figure 2. *Solve a Problem sequence diagram*

Login

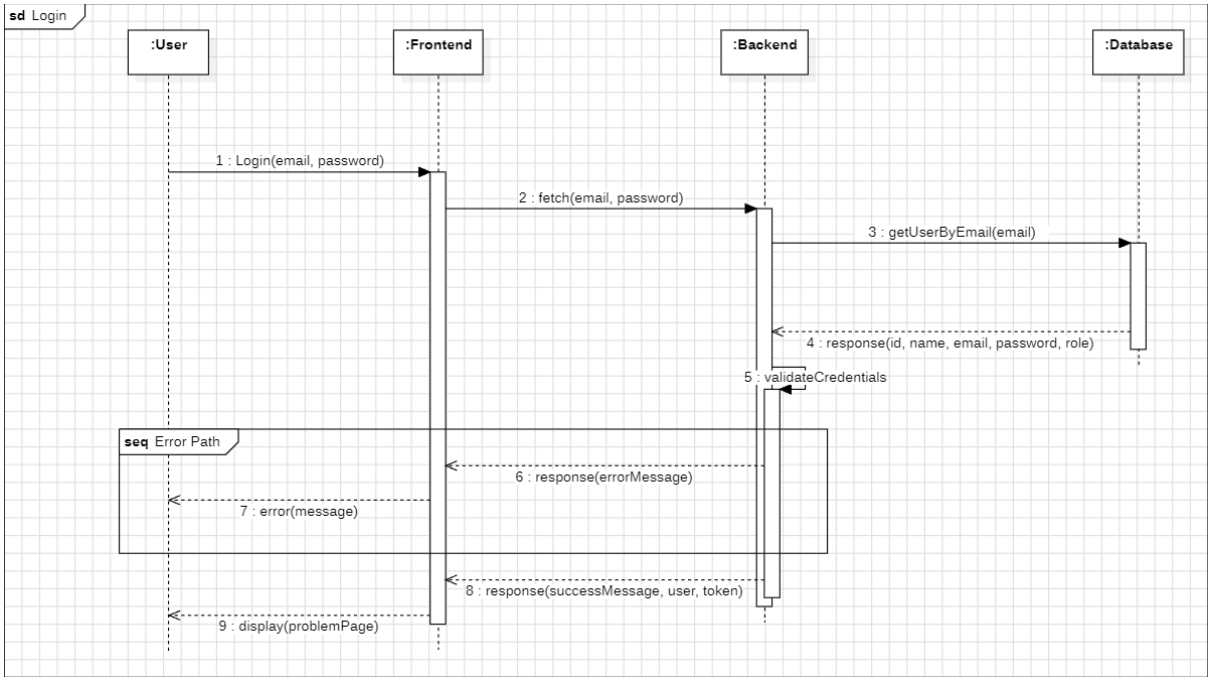

Figure 3. *Login sequence diagram*

Sign Up

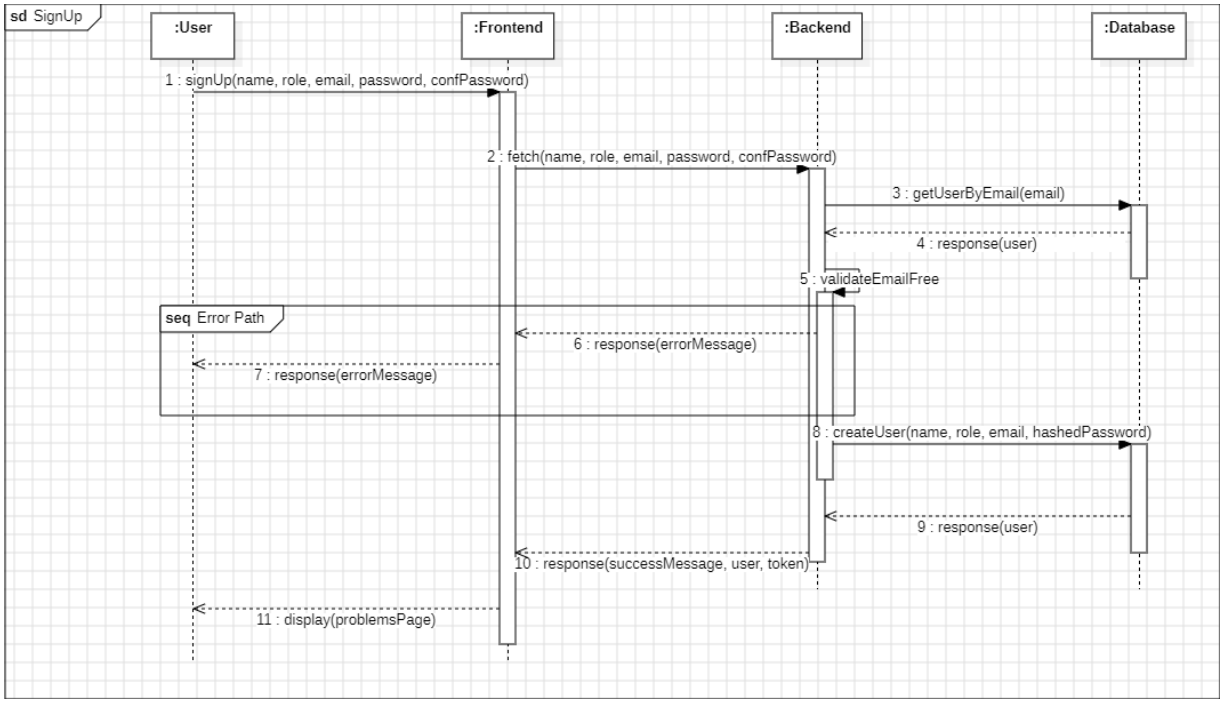

Figure 4. *Sign Up sequence diagram*

View Problems

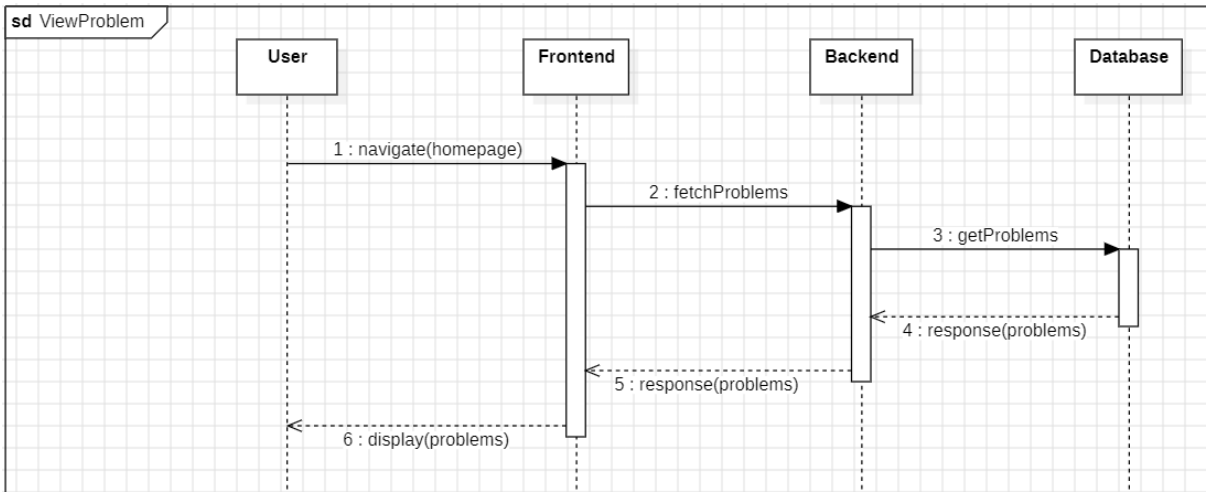

Figure 5. *View Problems sequence diagram*

Create Problem / Test Cases

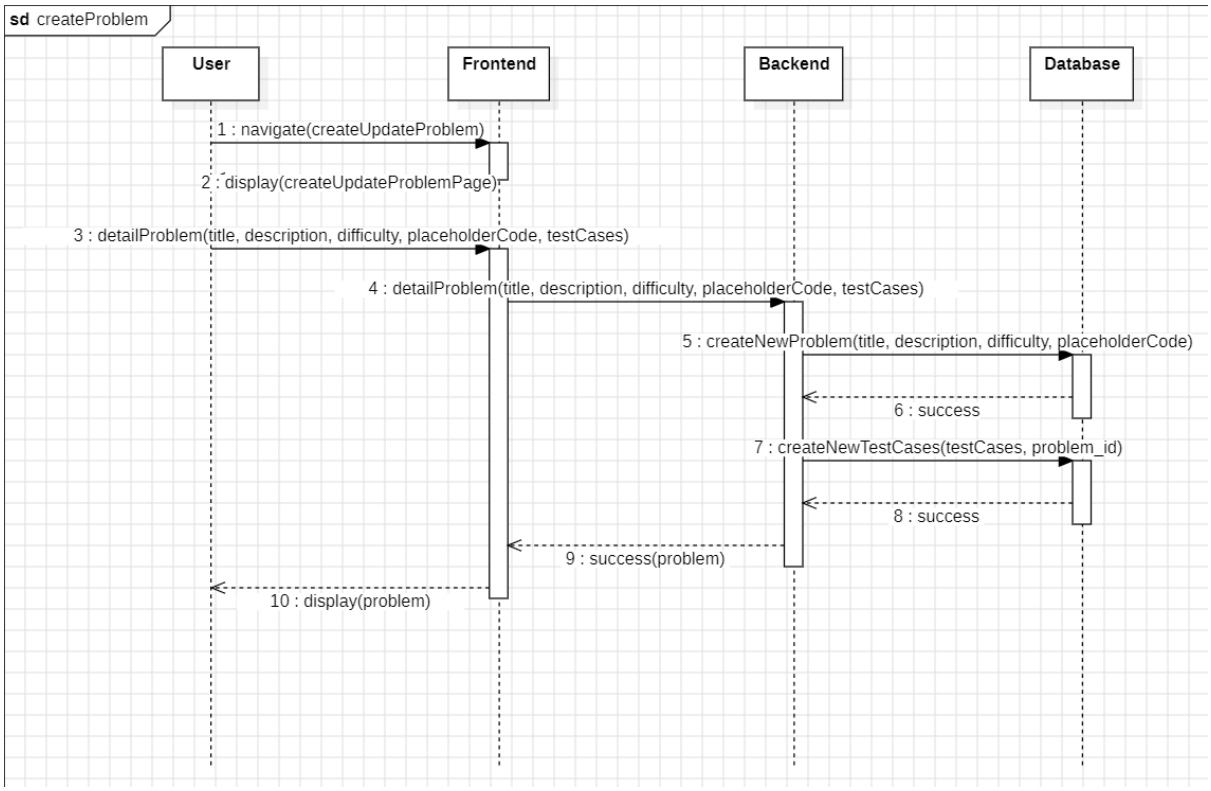

Figure 6. *Create Problem / Test Cases sequence diagram*

Update Problem / Test Cases

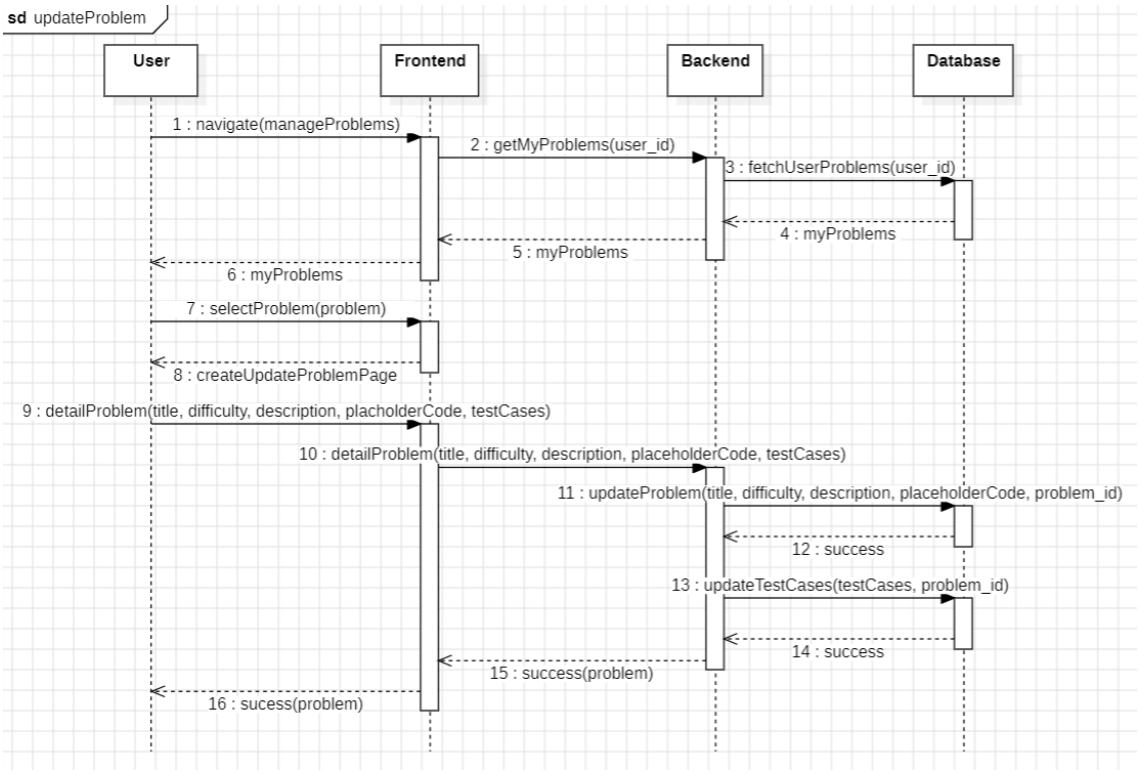

Figure 7. *Update Problem / Test Cases sequence diagram*

Create / Update Course

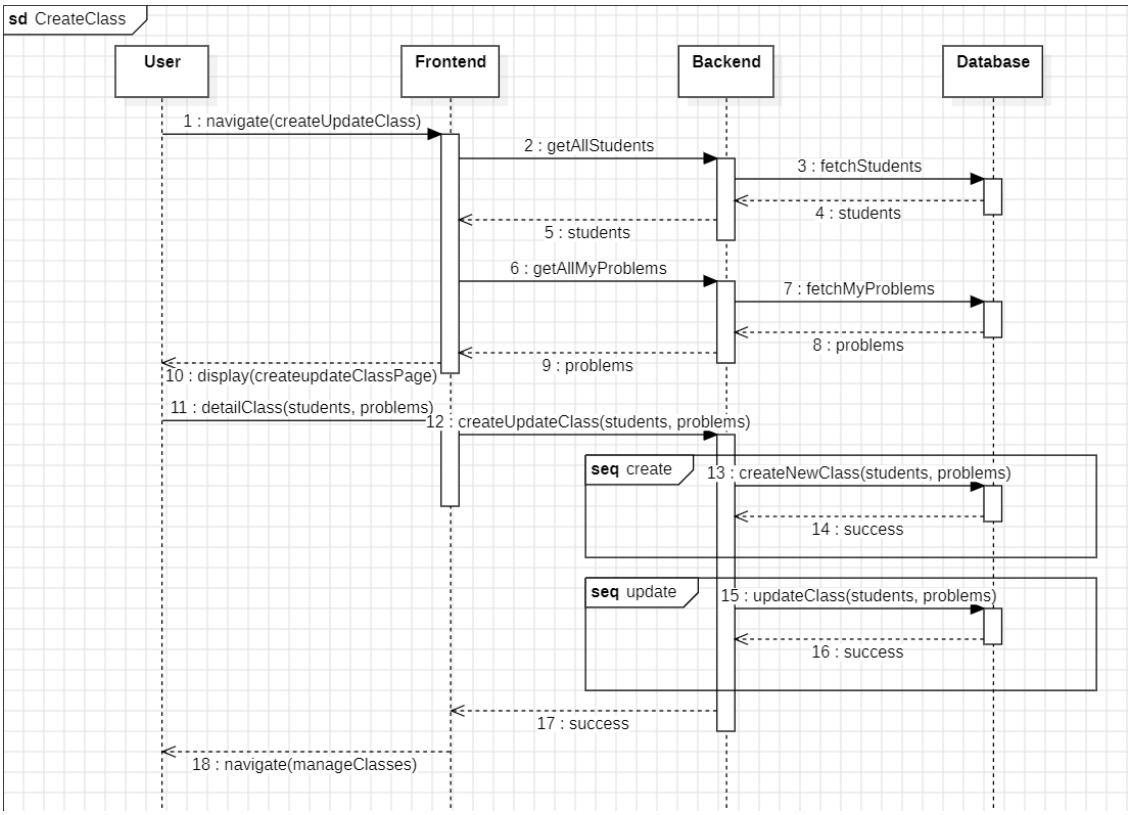

Figure 8. *Create / Update Course sequence diagram*

View Results

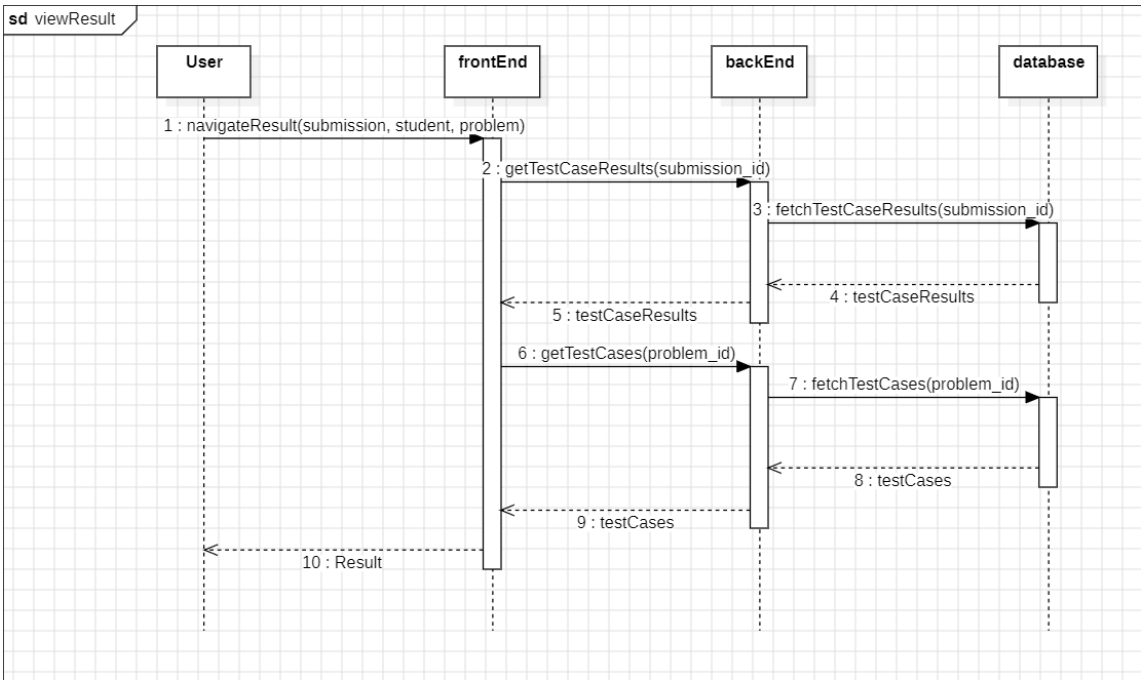

Figure 9. *View Results sequence diagram*

View Profile

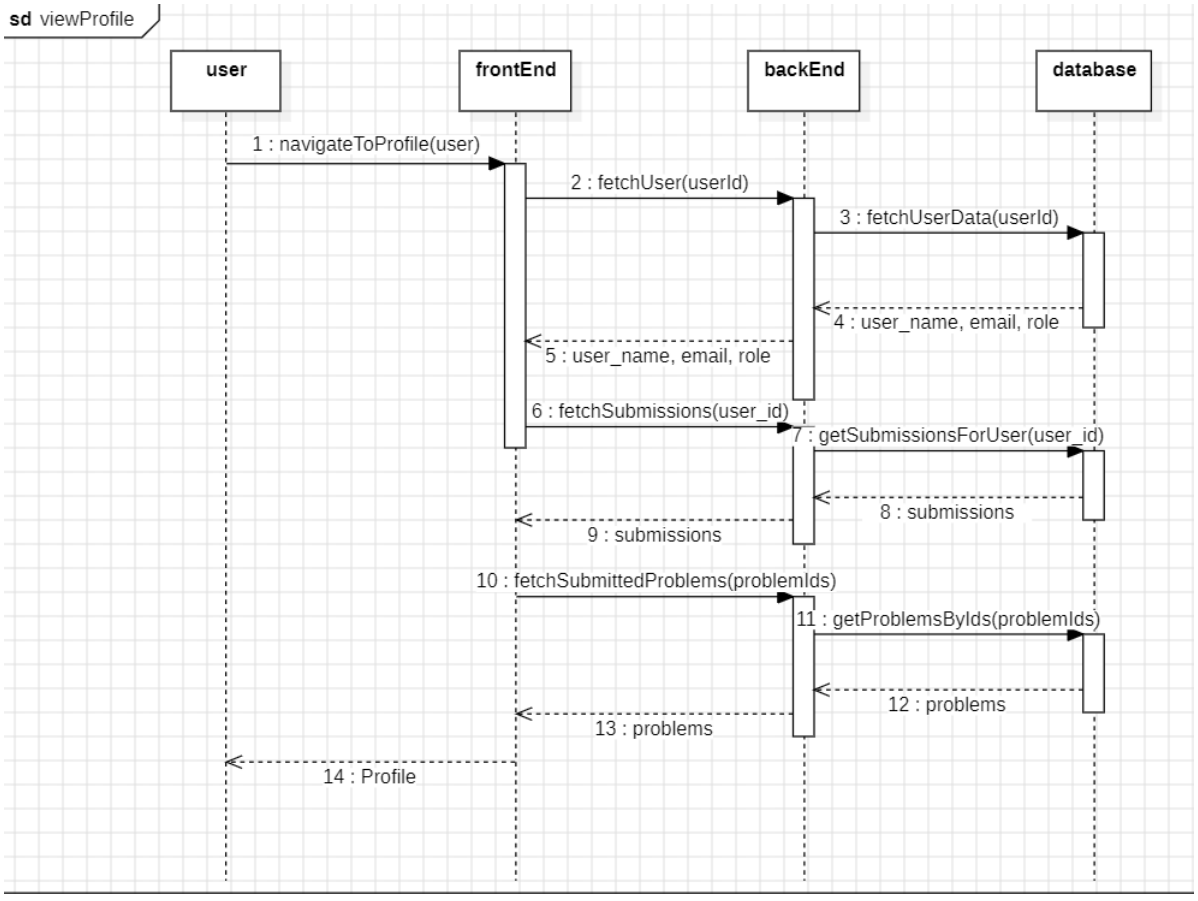

Figure 10. *View Profile sequence diagram*

View Leaderboard

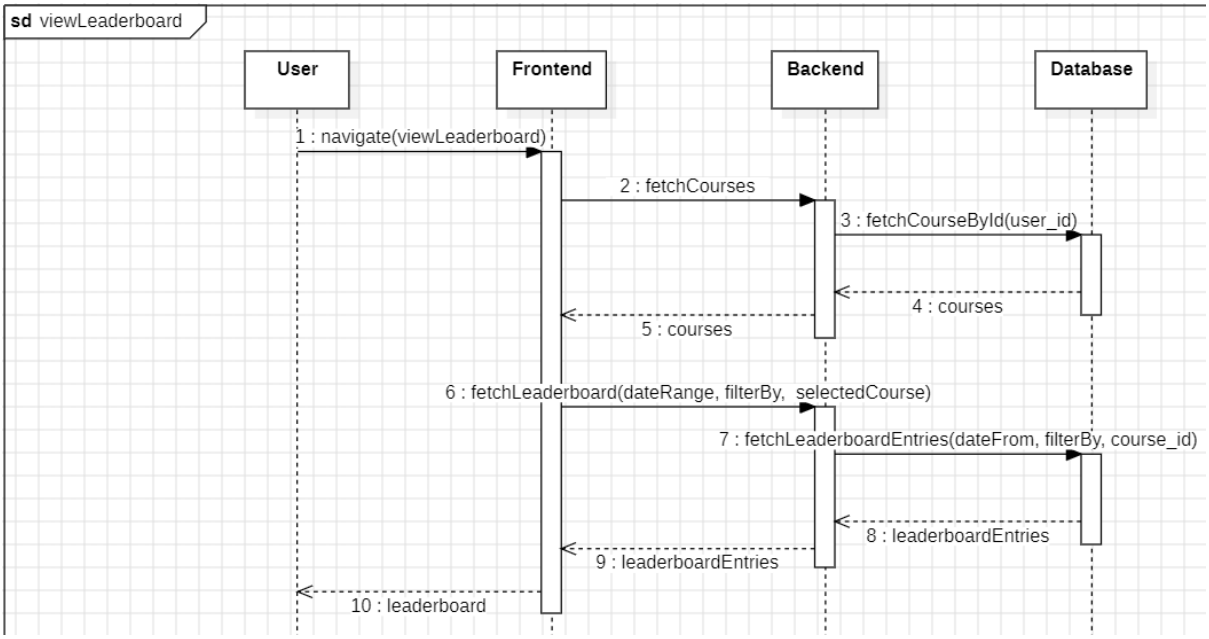

Figure 11. *View leaderboard sequence diagram*

Packages

Several packages will be used during the development of SETU Code Lab. Some of these are necessary to build important features and some vastly improve user experience. These packages include:

- **react-timer-hook** – A custom react hook designed to simplify the implementation of timers and stopwatches. This is needed for the stopwatch that measures how long a student takes to complete a particular problem (Labib, 2025).
- **react-markdown** – Enables markdown support so that lecturers can neatly format problem descriptions (John Otander, 2025).
- **react-spinners** – Provides free, stylish, pre-built loading indicators for React applications (Hu, 2026).
- **dockerode** – Enables interaction between the web server and docker's remote API (Dias, 2025).
- **@types/dockerode** – Provides required dockerode types which are needed for use with TypeScript (TypeScript, 2026).
- **cookie-parser** – Middleware for Node.js and Express that enables easy parsing of cookies from incoming requests (S., 2025).
- **@codemirror/state @codemirror/view @codemirror/basic-setup –** This group of packages together provide the in-browser code editor that is needed to allow students to solve problems on SETU Code Lab (CodeMirror, 2026).
- **@codemirror/theme-one-dark** – This is a dark theme for the CodeMirror code editor and nicely fits the theming of the rest of SETU Code Lab.
- **@codemirror/lang-java** – Provides code editor support for Java such as language specific syntax highlighting (Haverbeke, 2025).
- **@codemirror/lang-python** – Provides code editor support for Python such as language specific syntax highlighting (Haverbeke, 2026).
- **@codemirror/language** – Implements required infrastructure to support different programming languages (Marijn Haverbeke, 2026).

Important Algorithms

Test Harnesses for Running Submitted Code

To run student submitted functions they must be injected into a static string harness which contains a main class and the needed dependency imports to map sample inputs to function parameters. This allows any inputted function to run in isolation in a docker container with sample inputs from associated test cases. This harness is mostly the same for all submissions; however, some parts need to be dynamic to allow functions with parameters of different primitive and complex types and functions with different numbers of parameters to run.

```
return `
  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 12. *Java test harness*

```
  return `
import json
import sys

${code}

input_data = json.load(sys.stdin)
result = ${functionName}(**input_data)
print(json.dumps(result))
`;
```

Figure 13. *Python test harness*

Preprocessing Java Input

A function called `preprocessJavaInput()` takes in the original placeholder code defined by the lecturer and the code solution written by the student and returns the complete test harness ready for execution seen above. Firstly, a signature regex is run against the placeholder code to extract: the function return type, function name, and the parameter list (see figure 14.).

```
function preprocessJavaInput(placeholder_code: string, code: string): string {
  const signatureRegex =
    /public\s+static\s+([\w<>\[\], ?]+)\s+(\w+)\s*\(([^)]*)\)/;
  const match = placeholder_code.match(signatureRegex);
  if (!match) {
    throw new Error("Could not find method signature in placeholder code.");
  }
  const returnType = match[1];
  const functionName = match[2];
  const paramsList = match[3];
```

Figure 14. *Signature regex which extracts details from placeholder code*

The next step is to split the parameter list into individual parameters. This requires another helper function called *`splitParams(paramsList)`* to handle complex and nested parameter types. This function takes in the parameter list which is a long string of all the parameter types and their names concatenated and returns an array of separated parameter types and names e.g. the string “int a, int b” becomes [“int a”, “int b”].

This function works by splitting parameters on the “,“ character however, it also measures depth allowing it to only split on top level commas. This is necessary for nested data types such as Map<String, Int> whose internal commas would otherwise break if depth was not considered.

```
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 15. *splitParams function*

Next our array of parameter types and their names [“int a”, “int b”] are split again into the form: { type: "int", name: "a" }, { type: "int", name: "b" }

```
const params = splitParams(paramsList);
const parsedParams = params.map((p) => {
  const match = p.match(/(.+)\s+(\w+)$/);
  if (!match) {
    throw new Error("Invalid parameter format: " + p);
  }
  return {
    type: match[1].trim(),
    name: match[2].trim(),
  };
});
```

Figure 16. *parameters being parsed*

Finally, the variable `functionCallLine`, which is needed to call the function, is built dynamically using `returnType`, `functionName` and `paramNames`.

```
const paramNames = parsedParams.map((p) => p.name);

const functionCallLine = `${returnType} result = ${functionName}(${paramNames.map((n) => "input." + n).join(", ")});`;

return `
  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 17. *functionCallLine built dynamically*

Preprocessing Python Input

Pre-processing Python input is much simpler in comparison to Java. This time all that is needed is the function name which can be extracted using a Python specific signature regex.

```
function preprocessPythonInput(placeholder_code: string, code: string): string {
  const signatureRegex = /def\s+(\w+)\s*\(([^)]*)\)/;
  const match = placeholder_code.match(signatureRegex);
  if (!match) {
    throw new Error(
      "Could not find method signature in Python placeholder code.",
    );
  }
  const functionName = match[1];

  return `
import json
import sys

${code}

input_data = json.load(sys.stdin)
result = ${functionName}(**input_data)
print(json.dumps(result))
`;
}
```

Figure 18. *preprocessPythonInput function*

Execution and Judging

Student code needs to be run against test cases with sample inputs and expected outputs to judge whether their solution is viable. To do this, student code is pre-processed and executed in an isolated docker container using sample inputs from test cases. This ensures that submissions are tested under the exact same conditions for every test case, every time and that student inputted code executes securely, in isolation with no access to the external network or file system.

The first part of the `startContainer()` function seen in figure 19. This handles the preparation of a test case and code for execution. The image parameter is used to tell the Docker service which Docker image to use to create the correct environment for each language to execute. The `processedInput` variable is the sample inputs from the current test case in the correct format and the `preprocessedCode` variable uses the respective preprocessing helper function mentioned earlier to prepare the correct test harness for the language being executed. The `startTime` variable is used to measure how long a test case will take to run.

```
export async function startContainer(
  image: string,
  placeholder_code: string,
  code: string,
  testCase: TestCase,
  language: string,
): Promise<TestCaseResult> {
  const processedInput = JSON.stringify(testCase.input_value);
  const preprocessedCode =
    language === "python"
      ? preprocessPythonInput(placeholder_code, code)
      : preprocessJavaInput(placeholder_code, code);
  const startTime = Date.now();
```

Figure 19. *startContainer function part one*

The next part of the function seen in figure 20. shows the commands that will run inside of the Docker container once the test harness is built. For both languages, the pre-processed code is written to a source file using a heredoc `(cat << 'EOF')`, and the processed test case input is written to input.json. The source file is then executed with the input piped in via stdin.

For Python, the harness is written to `main.py` and run directly with `python3 main.py < input.json`. For Java, the harness is written to `Main.java`, compiled with `javac Main.java`, and then executed with `java -cp ".:/app/*" Main < input.json`. The `-cp ".:/app/*"` flag sets the classpath to include both the current directory and any dependencies in `/app`, which are needed for JSON deserialization. The key difference is the additional compilation step required for Java before execution, which is not needed for Python as it is interpreted at runtime.

```
  const cmd =
    language === "python"
      ? `
cat << 'EOF' > main.py
${preprocessedCode}
EOF
cat << 'ENDINPUT' > input.json
${processedInput}
ENDINPUT
python3 main.py < input.json
`
      : `
cat << 'EOF' > Main.java
${preprocessedCode}
EOF
cat << 'ENDINPUT' > input.json
${processedInput}
ENDINPUT
javac Main.java
java -cp ".:/app/*" Main < input.json
`;
```

Figure 20. *startContainer function part two, commands that run in Docker container*

Figure 21. shows how the Docker container is created using the shell commands from the previous step. Several resource constraints are applied here to ensure fairness and security. `NetworkMode: "none"` restricts any external network access, preventing external requests. `Memory`, `CpuPeriod`, `CpuQuota` and `PidsLimit` are set to cap memory, CPU, and process count, preventing a single submission from consuming excessive resources. A timeout of 100 seconds is also set which forcibly kills the container if execution exceeds the limit, preventing infinite loops from blocking execution indefinitely. The `endTime` variable is used to calculate how long the test case took to execute.

Note that `docker.createContainer()` is a function from the dockerode package which creates the docker container but does not execute it. `docker.start()` starts the docker container, `docker.kill()` forcibly kills the container and `docker.wait()` waits until the container exits naturally.

```
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,
  },
});

await container.start();

const timeout = setTimeout(async () => {
  try {
    await container.kill();
  } catch {}
}, 100000);

try {
  await container.wait();
} finally {
  clearTimeout(timeout);
}

const endTime = Date.now();
```

Figure 21. *startContainer function part three, container execution and timeout*

The final part of the `startContainer()` function (seen in figure 22.) takes the output from the test case, strips the docker headers, and compares the processed output with the expected output from the test case. This equality check is performed by a helper function called `deepEqual()` before `startContainer()` returns the relevant details to be displayed to the user and saved to the database.

```
const logs = await container.logs({ stdout: true, stderr: true });
await container.remove({ force: true }).catch(() => {});

const combinedOutput = stripDockerHeader(logs as Buffer);

let actualOutput: any;
try {
  actualOutput = JSON.parse(combinedOutput);
} catch {
  actualOutput = combinedOutput;
}

const passed = deepEqual(actualOutput, testCase.expected_value);

return {
  test_case_id: testCase.test_case_id as number,
  passed,
  actual_output: combinedOutput,
  runtime_ms: endTime - startTime,
};
```

Figure 22. *startContainer function part four, processing output and returning data*

`deepEqual()` is an important function which checks if the expected output matches the actual output for a given test case. This function is needed because a simple TypeScript "`===`" operator only checks reference equality for objects and arrays, not their contents e.g. arrays of arrays, [[2,1], [1,2]]. To solve this, `deepEqual()` examines these cases recursively ensuring each element inside of nested objects are equal (see figure 23.).

```
function deepEqual(a: any, b: any): boolean {
  if (typeof a !== typeof b) return false;
  if (typeof a !== "object" || a === null) return a === b;
  if (Array.isArray(a) !== Array.isArray(b)) return false;
  if (Array.isArray(a)) {
    if (a.length !== b.length) return false;
    return a.every((item, i) => deepEqual(item, b[i]));
  }
  const keysA = Object.keys(a).sort();
  const keysB = Object.keys(b).sort();
  if (keysA.join() !== keysB.join()) return false;
  return keysA.every((key) => deepEqual(a[key], b[key]));
```

Figure 23. *deepEquals function*

Approach to the use of Artificial Intelligence

It is likely that some students using SETU Code Lab will attempt to use generative AI to assist them with completing problems. This is not how the platform is intended to be used as it hinders students' ability to learn coding concepts effectively. While it is impossible to prevent the use of generative AI entirely, the platform aims to increase the work factor to discourage most users from using it. A file called antiCheat.ts has been created which disables copy/paste functionality and can detect when a user switches tabs. This is only active on the "solve a problem" screen and can be bypassed in certain situations. This only acts as a deterrent to the use of AI assistance.

```
export const useAntiCheat = (submitted: boolean) => {
  const [shouldAutoSubmit, setShouldAutoSubmit] = useState(false);

  useEffect(() => {
    const prevent = (e: Event) => e.preventDefault();
    const preventContext = (e: MouseEvent) => e.preventDefault();

    const handleKeyDown = (e: KeyboardEvent) => {
      if (
        (e.ctrlKey || e.metaKey) &&
        ["c", "v", "x", "a"].includes(e.key.toLowerCase())
      ) {
        e.preventDefault();
        alert("Warning: Copy/Paste is disabled");
      }
    };

    const handleVisibility = () => {
      if (document.hidden && !submitted) {
        setShouldAutoSubmit(true);
        alert("You left the tab. Your work has been automatically submitted.");
      }
    };

    const handleDragOver = (e: DragEvent) => {
      e.preventDefault();
      e.stopPropagation();
    };

    const handleDrop = (e: DragEvent) => {
      e.preventDefault();
      e.stopPropagation();
      alert("Warning: Drag and drop is disabled");
    };

    const handleDragStart = (e: DragEvent) => {
      e.preventDefault();
      e.stopPropagation();
    };

    document.addEventListener("visibilitychange", handleVisibility);
    document.addEventListener("copy", prevent);
    document.addEventListener("paste", prevent);
    document.addEventListener("cut", prevent);
    document.addEventListener("contextmenu", preventContext);
    document.addEventListener("keydown", handleKeyDown);
    document.addEventListener("dragover", handleDragOver, true);
    document.addEventListener("drop", handleDrop, true);
    document.addEventListener("dragstart", handleDragStart, true);
    document.addEventListener("dragenter", prevent, true);

    return () => {
      document.removeEventListener("visibilitychange", handleVisibility);
      document.removeEventListener("copy", prevent);
      document.removeEventListener("paste", prevent);
      document.removeEventListener("cut", prevent);
      document.removeEventListener("contextmenu", preventContext);
      document.removeEventListener("keydown", handleKeyDown);
      document.removeEventListener("dragover", handleDragOver, true);
      document.removeEventListener("drop", handleDrop, true);
      document.removeEventListener("dragstart", handleDragStart, true);
      document.removeEventListener("dragenter", prevent, true);
    };
  }, [submitted]);
  return { shouldAutoSubmit, setShouldAutoSubmit };
};
```

Figure 24. *AntiCheat.ts file to deter users from using generative AI*

Entity Relationship Diagram

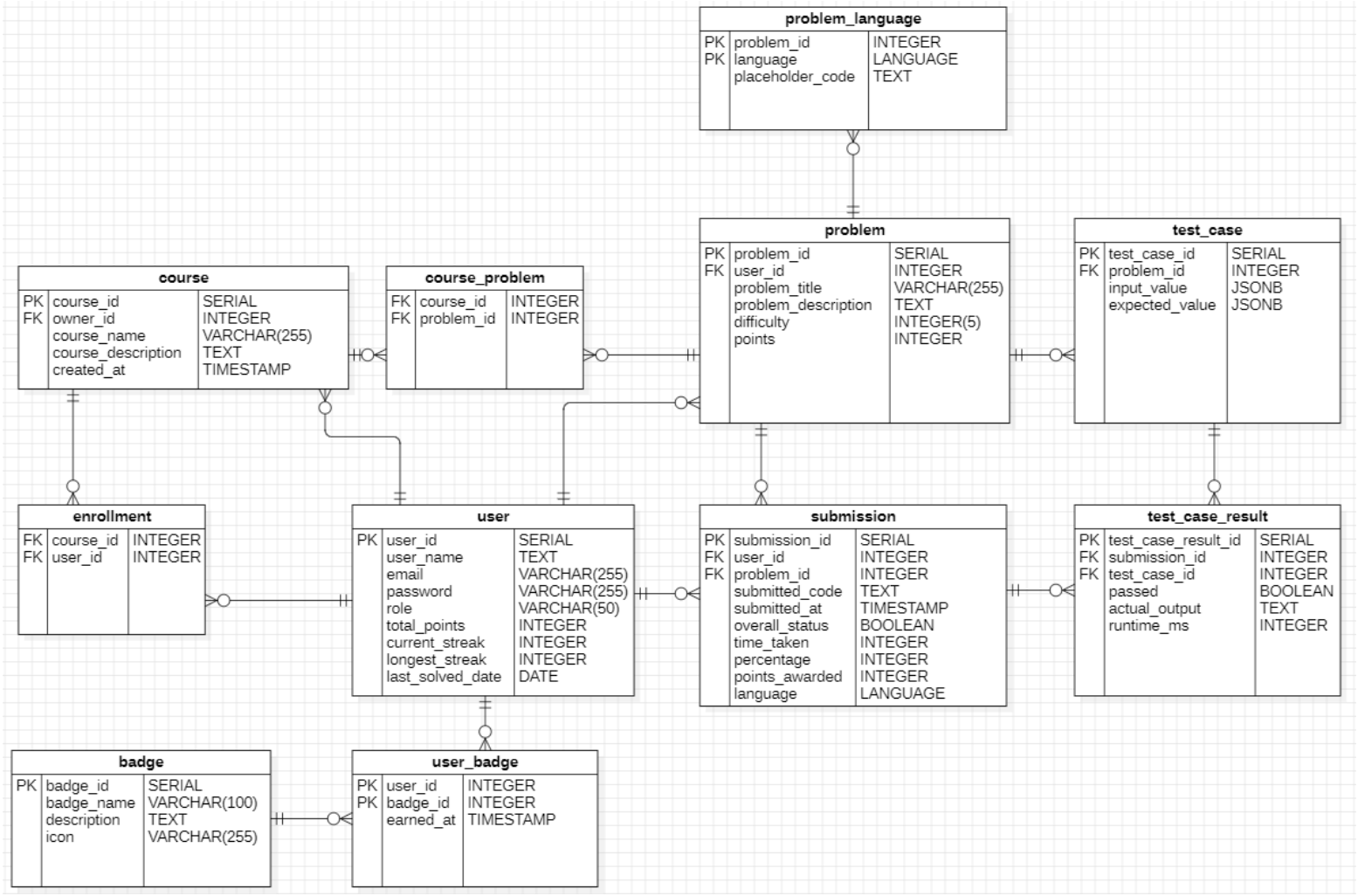

Figure 25. *Entity Relationship diagram*

Database Design Notes

- All tables use SERIAL primary keys for auto-incrementing IDs.
- Cascading deletes are applied on child tables (test_case, submission, course_problem, enrollment) so that removing a parent record automatically cleans up related data.
- The enrollment and course_problem tables use composite primary keys.
- problem.difficulty is constrained to values 1–5.
- users.role is constrained to 'student' or 'lecturer'.
- test_case.input_value and expected_value are stored as JSONB to support flexible, structured test data.
- Streak tracking (current_streak, longest_streak, last_solved_date) is maintained on the users table and updated at submission time.
- A database dump file called init.sql can be found in SETU_Code_Lab_Code > server > db > init. This file shows the SQL statements required to recreate the database. This also automatically initializes the database upon running `docker compose up -d` when running the platform locally for the first time (See user manual for more information).

SQL Queries

Login / Sign Up

createUser

```
INSERT INTO users (user_name, role, email, password)
    VALUES ($1, $2, $3, $4)
    RETURNING *
```

getUserByEmail

```
SELECT user_id AS id, user_name AS name, email,
       password, role
    FROM users
    WHERE email = $1
```

View Problems

fetchProblems

```
SELECT problem.*, users.user_name
    FROM problem AS problem
    JOIN users AS users ON problem.user_id = users.user_id
```

Solve Problem

fetchTestCases

```
SELECT * FROM test_case WHERE problem_id = $1
```

createSubmission

```
INSERT INTO submission
    (user_id, problem_id, submitted_code, overall_status,
     time_taken, percentage, points_awarded)
    VALUES ($1, $2, $3, $4, $5, $6, $7)
    RETURNING *
```

createTestCaseResult

```
INSERT INTO test_case_result
    (submission_id, test_case_id, passed, actual_output, runtime_ms)
    VALUES ($1, $2, $3, $4, $5)
```

CRUD Problem

insertProblem

```
INSERT INTO problem
    (user_id, problem_title, problem_description,
     difficulty, points)
    VALUES ($1, $2, $3, $4, $5)
    RETURNING *
```

fetchProblemsByUserId

```
SELECT problem.*, users.user_name
    FROM problem AS problem
    JOIN users AS users ON problem.user_id = users.user_id
    WHERE problem.user_id = $1
```

getAllAvailableProblems

```
SELECT * FROM problem WHERE user_id = $1 OR user_id = 1
```

updateProblem

```
UPDATE problem
    SET problem_title=$1, problem_description=$2,
        difficulty=$3, points=$4
    WHERE problem_id = $5
    RETURNING *
```

deleteProblem

```
DELETE FROM problem WHERE problem_id = $1 RETURNING *
```

CRUD Test Case

createTestCase

```
INSERT INTO test_case (problem_id, input_value, expected_value)
    VALUES ($1, $2::json, $3::json)
    RETURNING *
```

updateTestCase

```
UPDATE test_case
    SET input_value=$1, expected_value=$2
    WHERE test_case_id = $3
    RETURNING *
```

deleteTestCase

```
DELETE FROM test_case WHERE test_case_id = $1 RETURNING *
```

CRUD Course

fetchAllStudents

```
SELECT user_id AS student_id, user_name AS student_name
    FROM users WHERE role = 'student'
```

insertCourse

```
INSERT INTO course (owner_id, course_title, course_description)
    VALUES ($1, $2, $3) RETURNING *
```

insertCourseProblem

```
INSERT INTO course_problem (course_id, problem_id)
    VALUES ($1, $2) RETURNING *
```

insertEnrollment

```
INSERT INTO enrollment (course_id, user_id)
    VALUES ($1, $2) RETURNING *
```

fetchCourseByUserId

```
SELECT c.* FROM course c
    JOIN enrollment e ON c.course_id = e.course_id
    WHERE e.user_id = $1 ORDER BY c.course_id ASC
```

fetchCreatedCourseByUserId

```
SELECT * FROM course WHERE owner_id = $1 ORDER BY course_id ASC
```

updateCourseDetails

```
UPDATE course SET course_title=$1, course_description=$2
    WHERE course_id = $3 RETURNING *
```

deleteCourseProblems

```
DELETE FROM course_problem WHERE course_id = $1
```

deleteEnrollmentsByCourseId

```
DELETE FROM enrollment WHERE course_id = $1
```

fetchProblemIdsFromCourse

```
SELECT problem_id FROM course_problem WHERE course_id = $1
```

fetchStudentIdsFromCourse

```
SELECT user_id FROM enrollment WHERE course_id = $1
```

View Results

fetchStudentsOnCourse

```
SELECT u.user_id, u.user_name FROM users u
    JOIN enrollment e ON u.user_id = e.user_id
    WHERE e.course_id = $1
      AND u.role = 'student'
```

getSubmissionsForCourse

```
SELECT DISTINCT ON (s.user_id, s.problem_id)
    s.submission_id, s.user_id, s.problem_id,
    s.submitted_code, s.submitted_at, s.overall_status,
    s.time_taken, s.percentage
    FROM submission s
    WHERE s.user_id = ANY($1)
      AND s.problem_id = ANY($2)
      AND s.submitted_at > $3::timestamp
    ORDER BY s.user_id, s.problem_id, s.submitted_at DESC
```

fetchTestCaseResults

```
SELECT * FROM test_case_result WHERE submission_id = $1
```

View Profile

fetchUserData

```
SELECT user_id, user_name, email, role, total_points,
       current_streak, longest_streak, last_solved_date
    FROM users WHERE user_id = $1
```

deleteAccount

```
DELETE FROM users WHERE user_id = $1
```

getSubmissionsForUser

```
SELECT * FROM submission
    WHERE user_id = $1 ORDER BY submitted_at DESC
```

getProblemsByIds

```
SELECT * FROM problem WHERE problem_id = ANY($1::int[])
```

Leaderboard

updateUserPoints

```
UPDATE users SET total_points = total_points + $1
    WHERE user_id = $2
```

getStreakData

```
SELECT last_solved_date, current_streak, longest_streak
    FROM users WHERE user_id = $1
```

updateStreak

```
UPDATE users
    SET current_streak = $1,
        longest_streak = GREATEST(longest_streak, $1),
        last_solved_date = CURRENT_DATE
    WHERE user_id = $2
```

checkFirstSolve

```
SELECT submission_id FROM submission
    WHERE user_id = $1 AND problem_id = $2 AND overall_status = true
```

fetchLeaderboardEntries

```
SELECT DISTINCT u.user_id, u.user_name, u.total_points,
    u.current_streak, u.longest_streak
    FROM users u
    INNER JOIN submission s ON u.user_id = s.user_id
    INNER JOIN course_problem cp ON s.problem_id = cp.problem_id
    WHERE cp.course_id = $1
```

`$1` defines how the data what data will be displayed in the leaderboard and in what way dynamically (see figure 16).

```
const values: any[] = [courseId];
let paramIndex = 2;

if (dateFrom) {
    query += ` AND u.role = 'student' AND s.submitted_at >= $${paramIndex}`;
    values.push(dateFrom);
    paramIndex++;
} else {
    query += ` AND u.role = 'student'`;
}

if (filterBy === "score") {
    query += ` ORDER BY u.total_points DESC`;
} else if (filterBy === "currentStreak") {
    query += ` ORDER BY u.current_streak DESC`;
} else {
    query += ` ORDER BY u.total_points DESC`;
}
```

Figure 26. *Dynamic SQL statement, defines how leaderboard will be displayed*

Multi-Language Support

getProblemLanguageData

```
SELECT * FROM problem_language
    WHERE problem_id = $1
```

insertProblemLanguage

```
INSERT INTO problem_language (problem_id, language, placeholder_code)
    VALUES problem_id = $1
```

deleteProblemLanguages

```
DELETE FROM problem_language
    WHERE problem_id = $1
```

Badges

fetchUserBadges

```
SELECT b.* FROM badge b
    JOIN user_badge ub
    ON b.badge_id = ub.badge_id
    WHERE ub.user_id = $1
```

awardBadgeIfNotExists

```
INSERT INTO user_badge (user_id, badge_id)
     VALUES ($1, $2)
     ON CONFLICT DO NOTHING
     RETURNING badge_id
```

User Interface

This section outlines the initial designs for every screen on the SETU Code Lab platform and how they flow from one screen to the next. It also includes other designs that may not necessarily be a screen for example the logo and the submission alert. Note that these designs are only for illustrative purposes and may not reflect exactly what the final platform looks like.

Logo

This is the logo and favicon for SETU Code Lab. It has been designed in the shape of the letter C and is intended to be simple and recognizable. The light grey and bright red colours have been chosen as they contrast well with the dark background chosen for the rest of the platform.

Figure 27. *Logo*

High-Level UI Flow

The diagram below shows how to navigate to each screen on the platform. Additional UI elements such as pop-ups and drop-down menus are not shown here, just the main screens. Items in the lecturer only box are only accessible to users with the lecturer role.

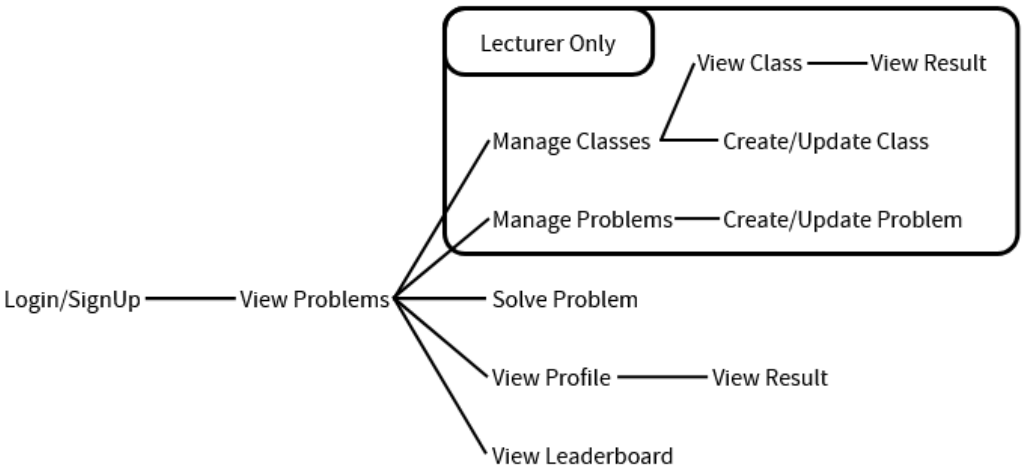

Figure 28. *User Interface flow diagram*

Login

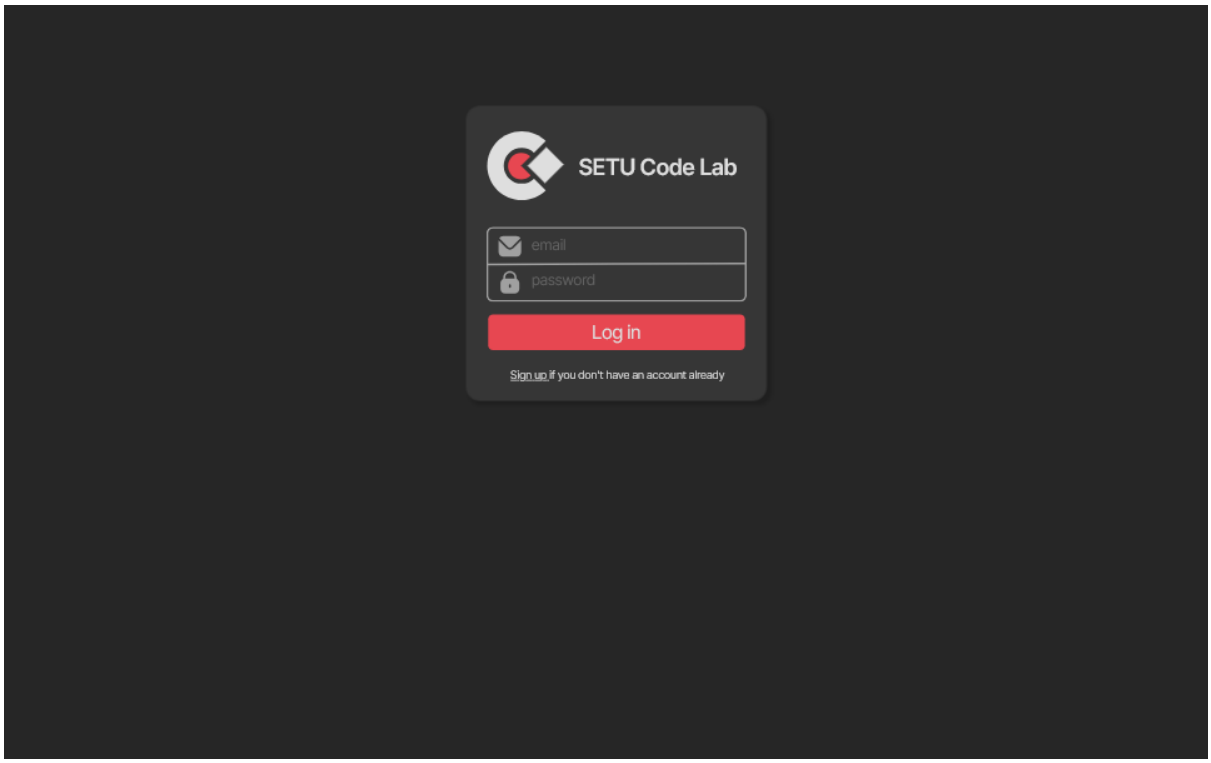

Figure 29. *Login screen design*

Sign Up

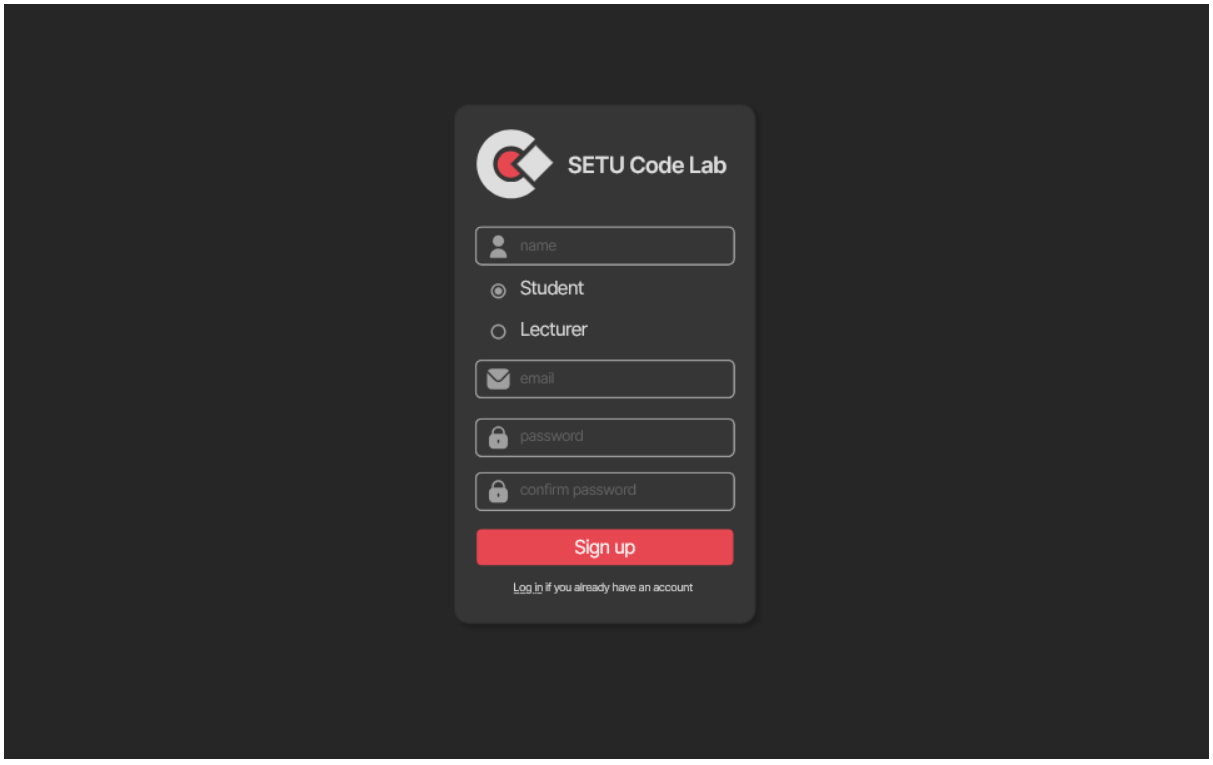

Figure 30. *Sign Up screen design*

View Problems

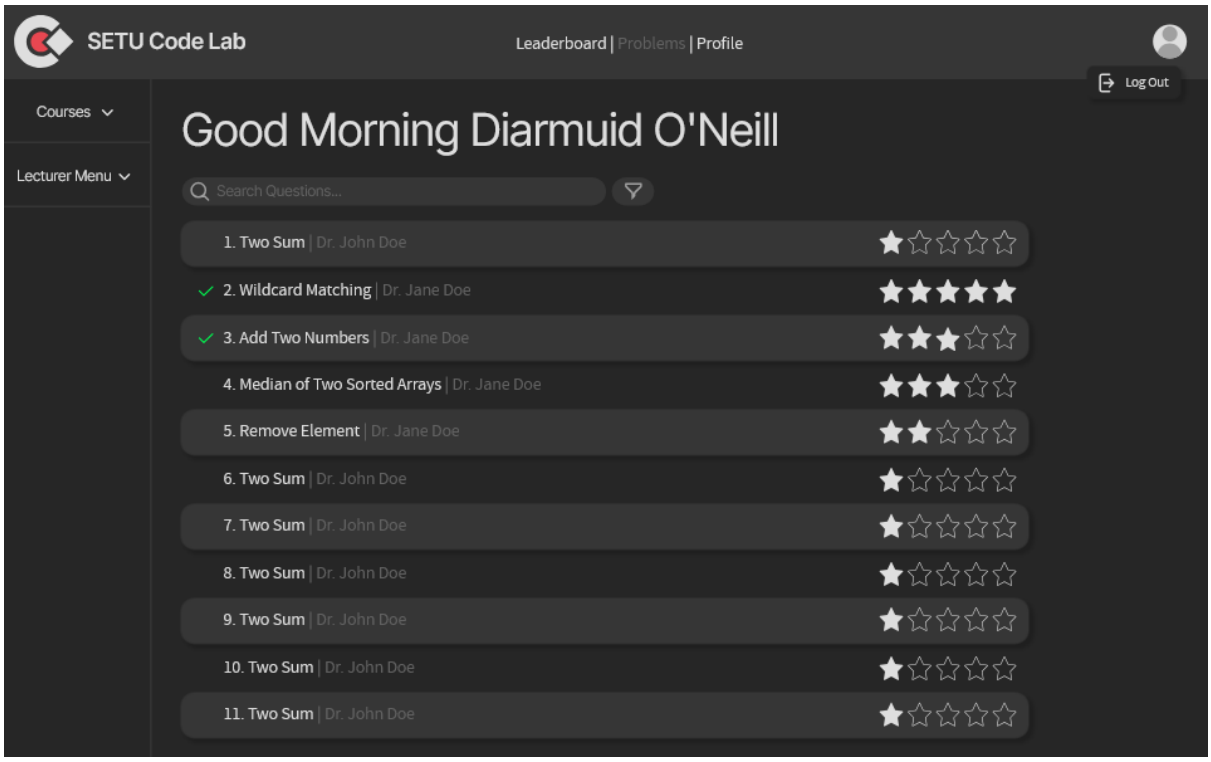

Figure 31. *View Problems screen design*

View Problem > Solve Problem

Figure 32. *Solve a Problem screen design*

View Problem > Solve Problem >Submission Alert

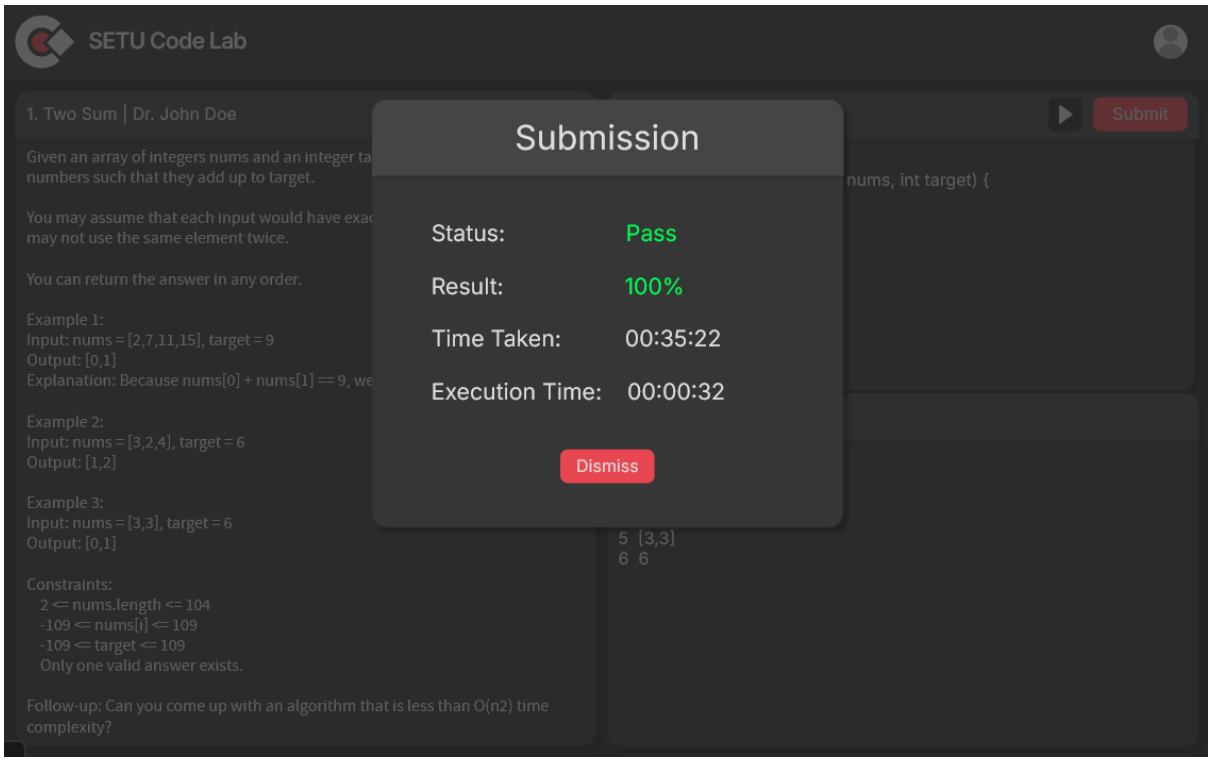

Figure 33. *Submission alert design*

View Problem > Manage Problems

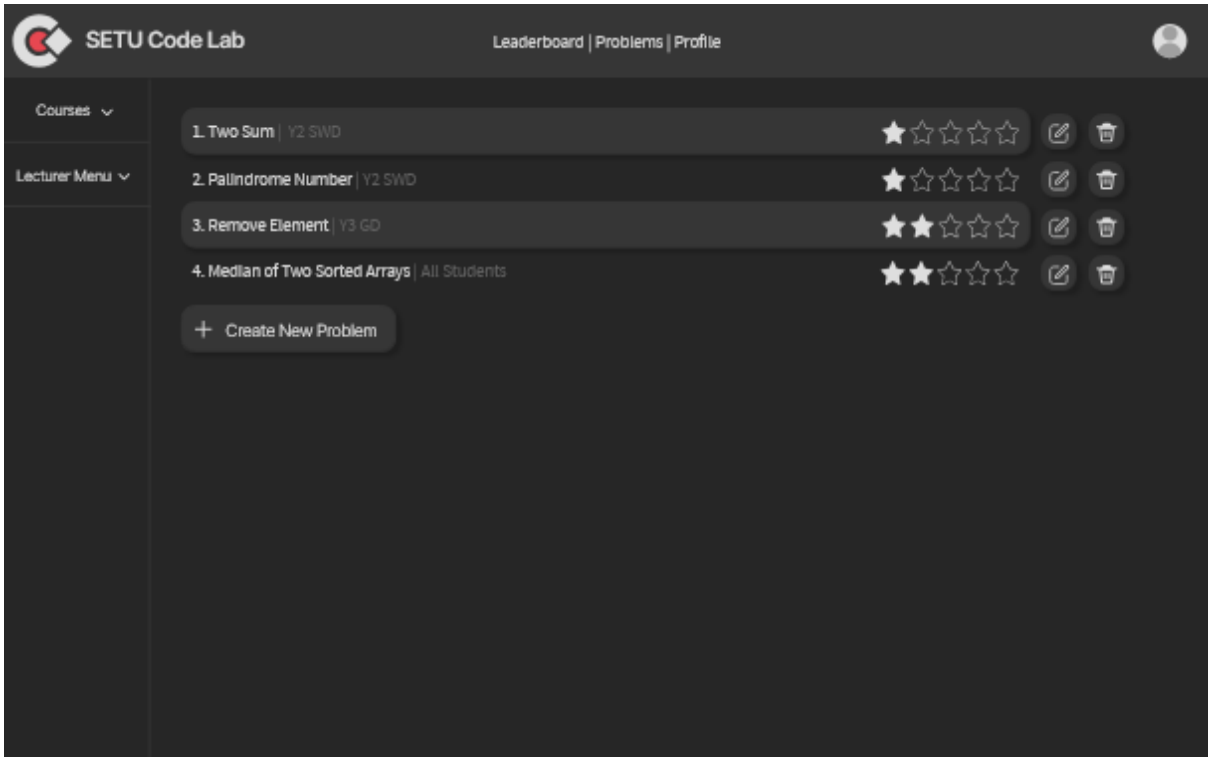

Figure 34. *Manage Problems screen design*

View Problem > Manage Problems > Create/Update Problem

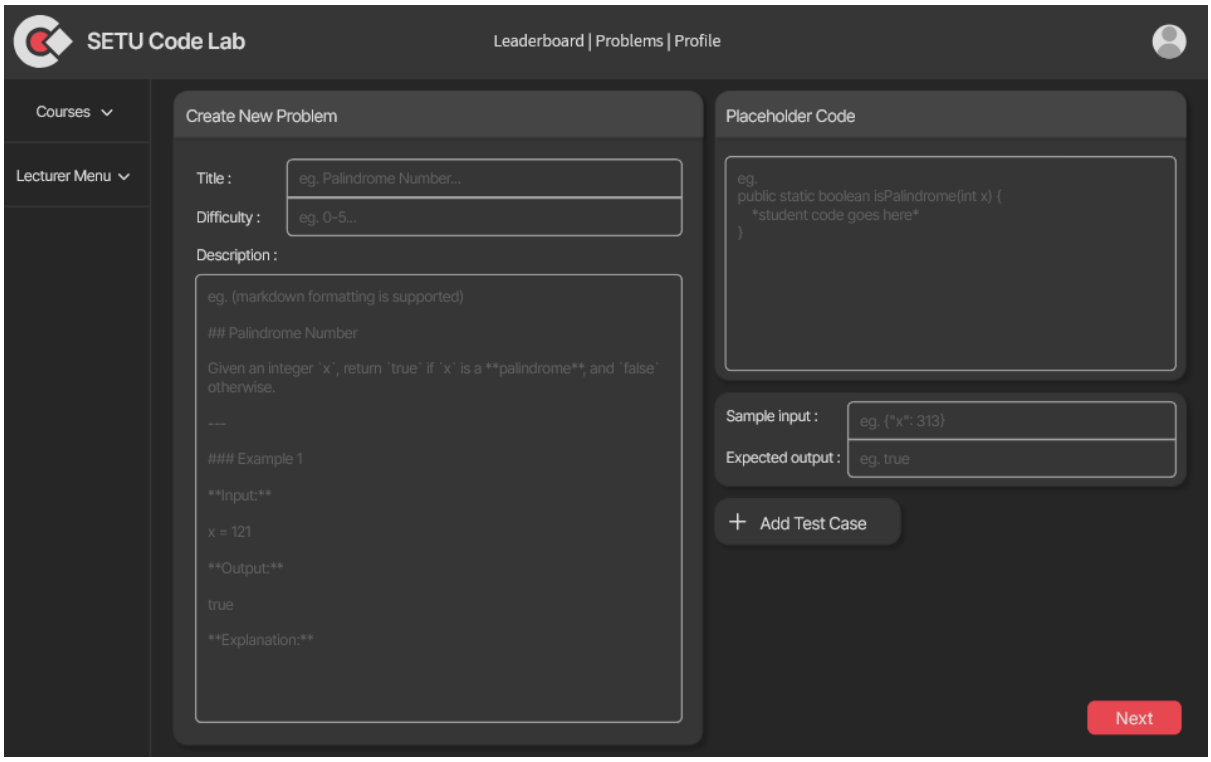

Figure 35. *Create / Update Problem screen design*

View Problem > Manage Courses

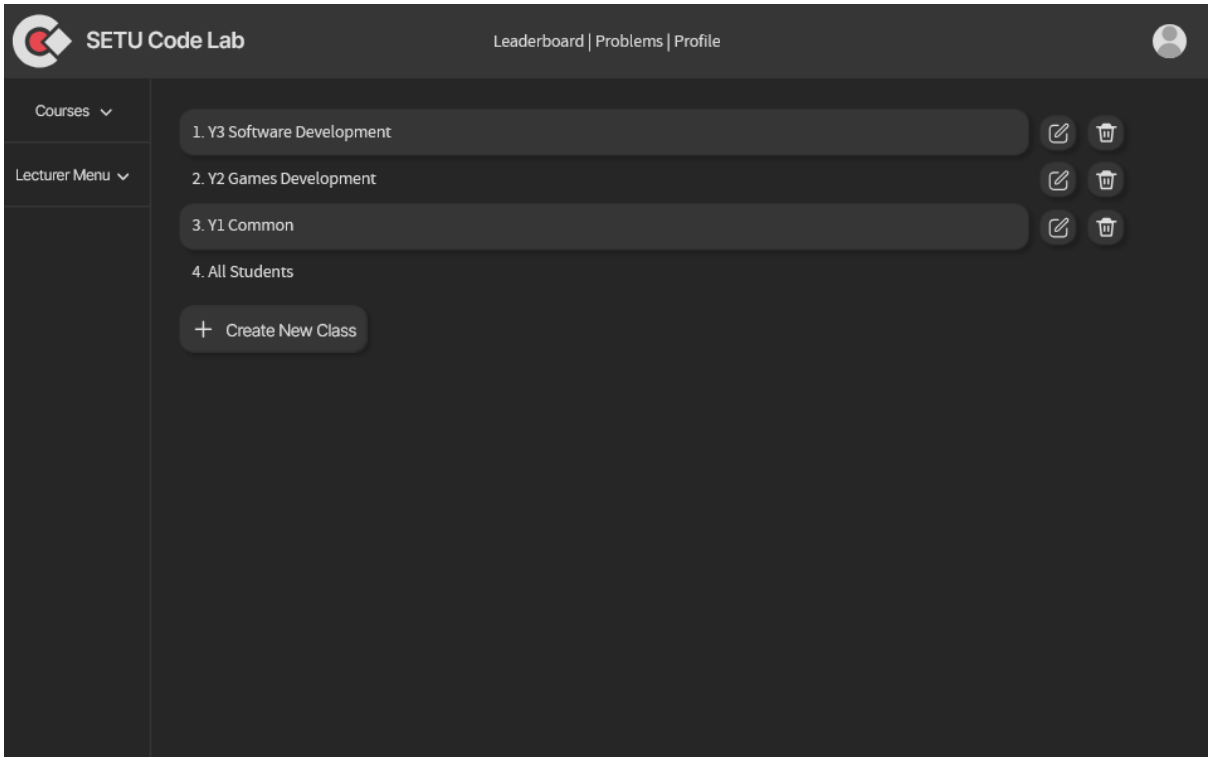

Figure 36. *Manage Courses screen design*

View Problem > Manage Courses > Create/Update Course

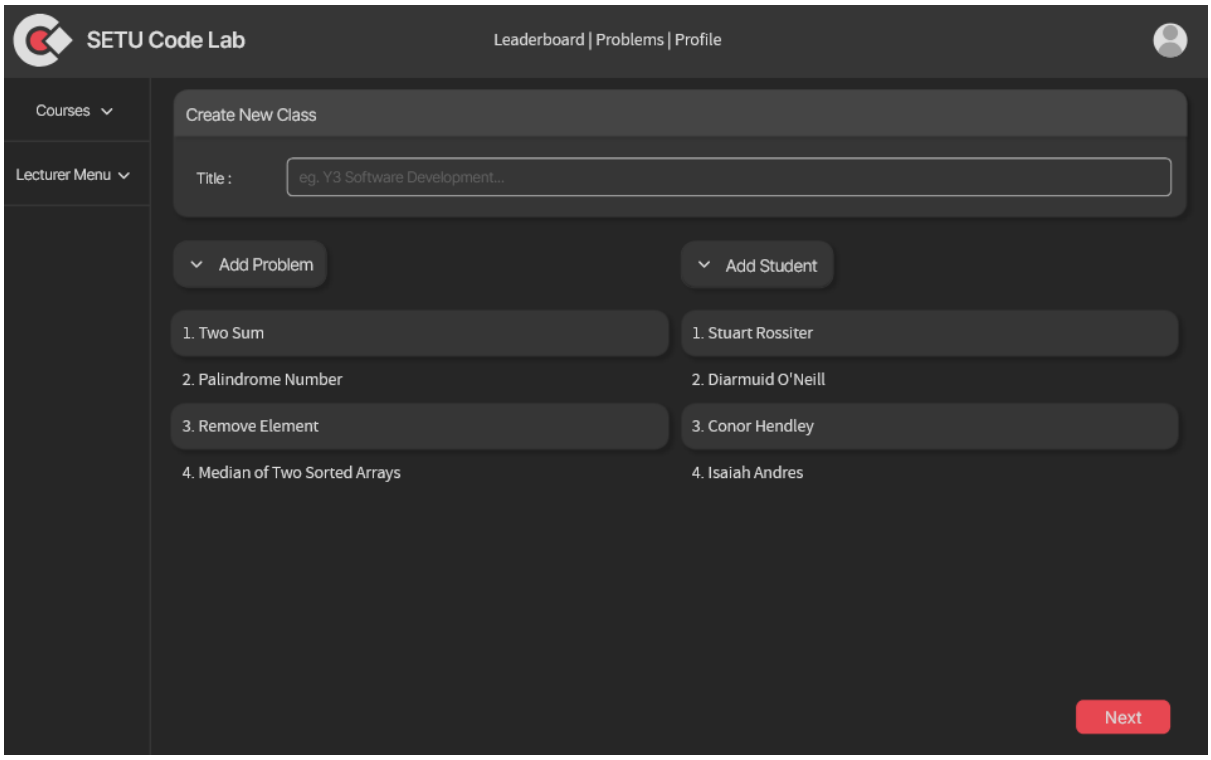

Figure 37. *Create / Update Course screen design*

View Problem > Manage Courses > View Course

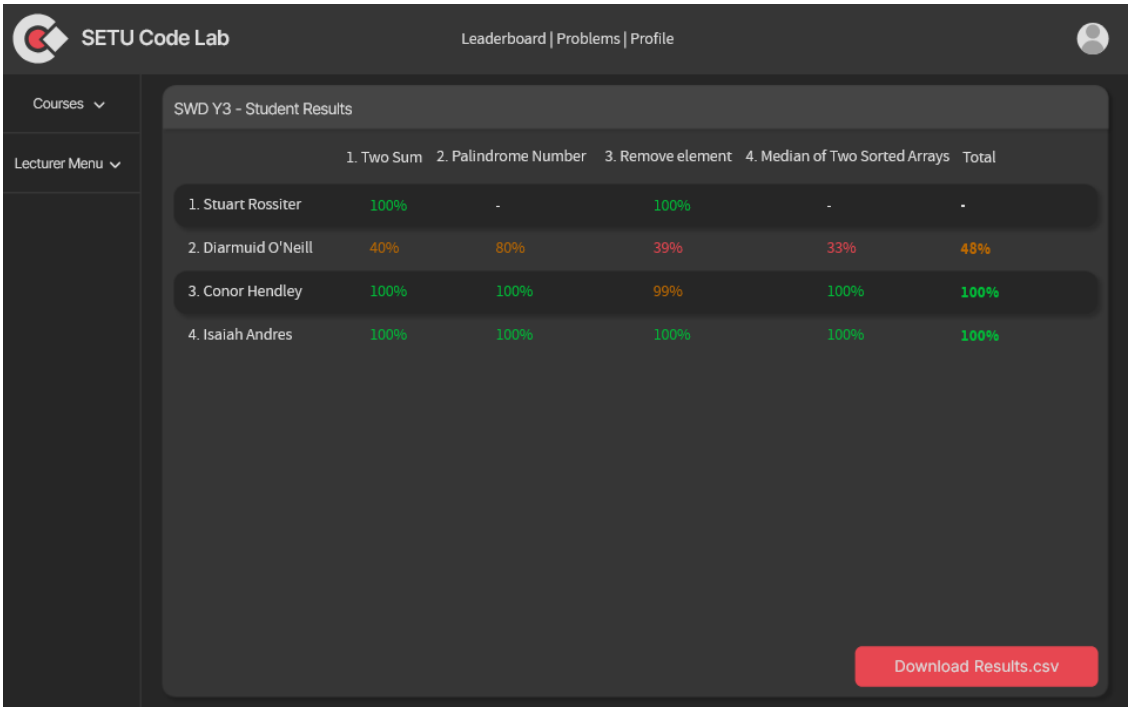

Figure 38. *View Course screen design*

View Problem > Manage Course > View Course > View Result

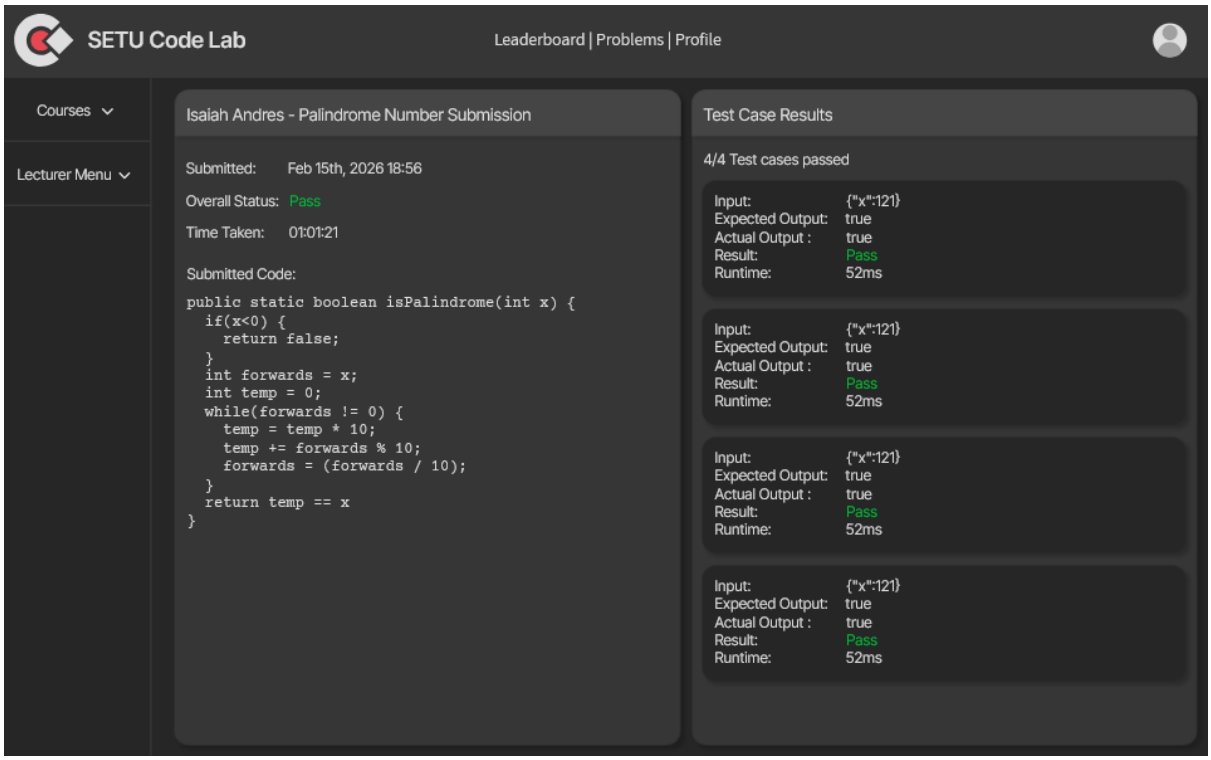

Figure 39. *View Result Screen design*

View Problem > Profile

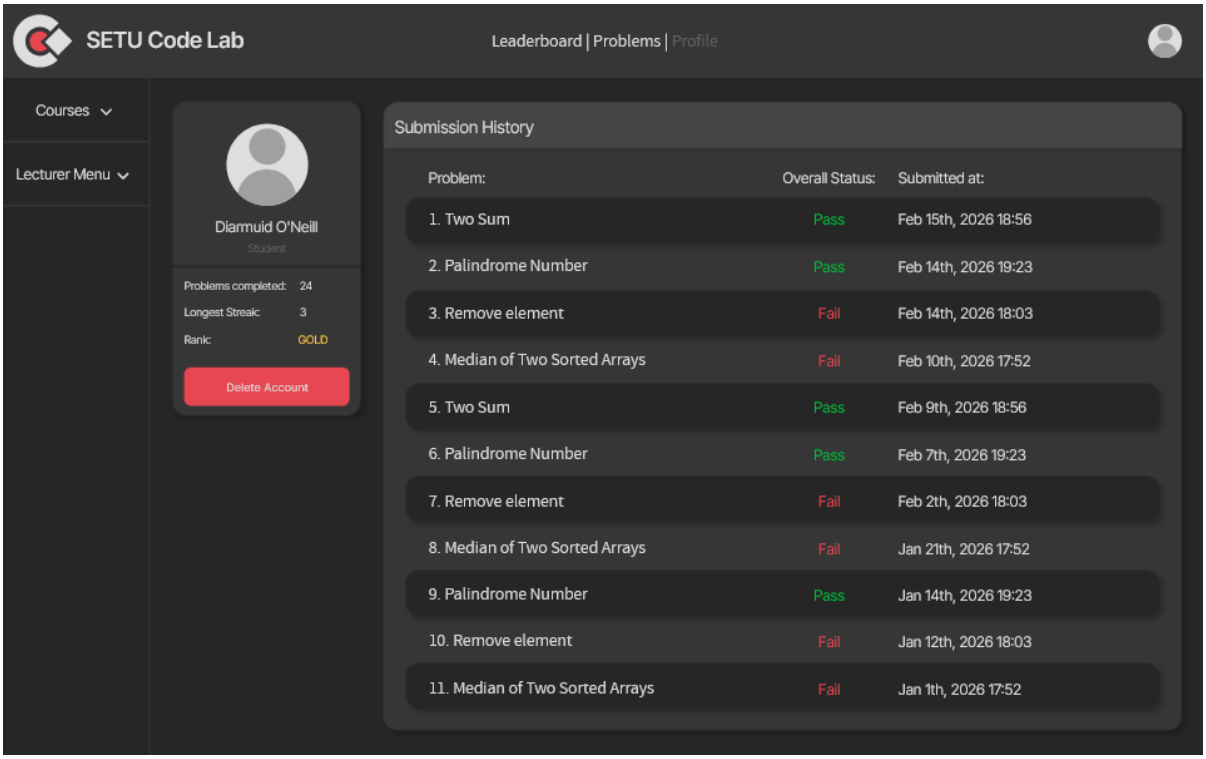

Figure 40. *View Profile screen design*

View Problem > Leaderboard

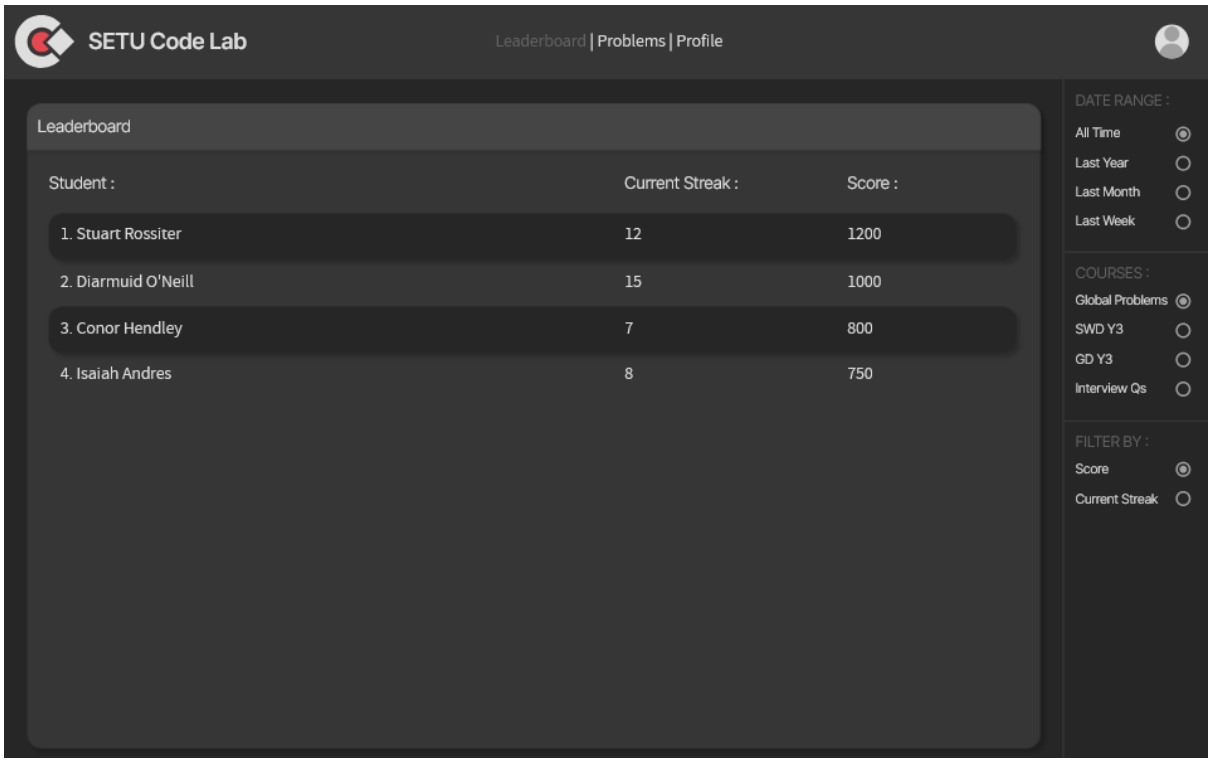

Figure 41. *Leaderboard screen design*

Bibliography

CodeMirror, 2026. *CodeMirror - System Guide*. [Online]
Available at: https://codemirror.net/docs/guide/
[Accessed 5 March 2026].

Dias, P., 2025. *NPM - dockerode*. [Online]
Available at: https://www.npmjs.com/package/dockerode
[Accessed 5 March 2026].

Haverbeke, M., 2025. *NPM - @codemirror/lang-java*. [Online]
Available at: https://www.npmjs.com/package/@codemirror/lang-java
[Accessed 5 March 2026].

Haverbeke, M., 2026. *Github - codemirror lang/python*. [Online]
Available at: https://github.com/codemirror/lang-python
[Accessed 17 March 2026].

Hu, D., 2026. *React Spinners*. [Online]
Available at: https://www.davidhu.io/react-spinners/
[Accessed 5 March 2026].

John Otander, T. W. R. H., 2025. *NPM - react-markdown*. [Online]
Available at: https://www.npmjs.com/package/react-markdown
[Accessed 5 March 2026].

Labib, A., 2025. *NPM - react-timer-hook*. [Online]
Available at: https://www.npmjs.com/package/react-timer-hook
[Accessed 5 March 2026].

Marijn Haverbeke, e. a., 2026. *NPM - @codemirror/language*. [Online]
Available at: https://www.npmjs.com/package/@codemirror/language
[Accessed 5 March 2026].

S., R., 2025. *NPM - cookie-parser*. [Online]
Available at: https://www.npmjs.com/package/cookie-parser
[Accessed 5 March 2026].

TypeScript, 2026. *NPM - @types/dockerode*. [Online]
Available at: https://www.npmjs.com/package/@types/dockerode
[Accessed 5 March 2026].