SETU Code Lab
Functional Specification

Diarmuid O'Neill

South East Technological University

1/12/2025

Table of Contents

Application Definition

SETU Code Lab is an in-browser study tool for students enrolled in computing-related courses in SETU (South East Technological University) and their lecturers. The core features allow students to select a code problem from a list and attempt to solve it in a built-in code editor. They will be able to run their solution safely in the browser and see if it passes the required test cases. When they are satisfied with their answer, they can press the submit button. This will show them if their solution is correct and how long it took to complete. This information is then saved to the student's profile so they can keep track of their progress. The intended benefit of this is to allow an easy and gamified way for students to practice their coding skills for any upcoming programming exams or coding interviews they might have.

Other core features include the ability for lecturers to upload their own code problems and test cases to a repository. This is useful for increasing the number of available problems and allows lecturers to use the platform in their labs to teach students specific coding concepts.

The platform will also have a full lecturer dashboard where lecturers can create class groups from lists of students and assign specific code problems to these classes. They will then be able to view all their student submissions in one place for grading purposes, saving time and increasing the application's usefulness in practical lab settings.

Some non-core features include search and filtering of code problems by type or difficulty, and public leaderboards based on the number of problems solved or fastest executing solution, which would further gamify learning.

Java has been selected as the first supported language as it is the first language Software Development students at SETU learn as part of their course. Support for more languages is considered a stretch goal that may be implemented if time allows.

User Groups

The two user groups identified for this application are the student and the lecturer.

A student is any person enrolled in a computing related course in SETU. Students need an easy way to see and study many different coding concepts both for their modules in college and for any code interviews they may have. SETU Code Lab aims to provide a repository of problems demonstrating these important concepts, an easy way to practice them and all while making the process as seamless and enjoyable as possible.

A lecturer is any person involved in teaching or supervising computing related modules in SETU. Lecturers need an easy way to create and distribute practical lab work to students and retrieve results for grading. SETU Code Lab will allow lecturers to assign problems to groups of students and receive their results as soon as they have completed the assigned problem. There is also the benefit of being able to reuse certain code problems year after year which saves even more time.

Requirements

Functional Requirements

Core Features

1. The system shall allow students to browse a list of Java coding problems
2. The system shall provide a built-in code editor for writing solutions
3. The system shall safely execute inputted code against predefined test cases
4. The system shall indicate if a student's submission has passed or failed the test cases
5. The system shall save student submissions to their profile
6. The system shall allow lecturers to upload new problems and test cases
7. The system shall allow lecturers to assign problems to groups of students
8. The system shall allow lecturers to see results and solutions from students which they have assigned problems to

Non-Core Features

1. The system should allow students to search and filter for problems with a specific name or difficulty
2. The system should provide a leaderboard ranking students on number of problems solved and/or ranking solutions on shortest execution time
3. The system should incorporate gamification elements such as daily log in streaks, badges and a ranking system

Non-Functional Requirements (FURPS+)

Functionality

- The system shall support at least one programming language (Java) for code execution.

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.
- Lecturers with existing accounts should be able to log in, create a new code problem, write something, and upload it with no prior training in ≤ 120 seconds.

Reliability

- The system shall have 99% uptime.

Performance

- The system shall return code execution results in ≤ 5 seconds for problems ≤100 lines of code and ≤10 test cases.

Security

- The system shall ensure submissions cannot access the server file system or network.
- The system shall ensure that only authorized users (lecturers and students) can access their respective dashboards.

Supportability

- The system shall remain usable (all previous usability tests pass) and readable on the top 5 most common desktop resolutions in Europe (1920x1080, 1536x864, 1366x768, 1280x720, 2560x1440) (StatCounter, 2025).

Compatibility

- The system shall be supported on the latest 2 versions of Firefox, Google Chrome, and Microsoft Edge.

Technologies

Front-End

- React – Component-based UI framework based on JavaScript. Chosen for its support of all modern browsers (Meta Platforms, 2025).
- TypeScript – Modern version of JavaScript, chosen for its type safety and easy integration with React (Microsoft, 2025).
- SCSS – A stylesheet language that is compiled to CSS which supports variables and allows nesting of styles. This reduces code duplication leading to better maintainability (Sass Team, 2025).
- CodeMirror – In browser code editor, chosen for its ease of integration, rich feature set, and high performance (Haverbeke, 2025).

Back-End

- Node.js – Server-side runtime for handling HTTP requests and API integration, chosen for its easy integration with TypeScript and React (OpenJS Foundation, 2025).
- Express – Lightweight web framework for Node.js (OpenJS Foundation, 2025).
- PostgreSQL – Relational database to store users, problems, submissions, and progress (PostgreSQL, 2025).
- Docker – Containerization platform used to safely sandbox submitted code. Containers will be given limited resources and no network access to ensure the highest level of security (Docker, 2024).
- DigitalOcean – VPS (Virtual Private Server) hosting provider (DigitalOcean, 2025).

Context Diagram

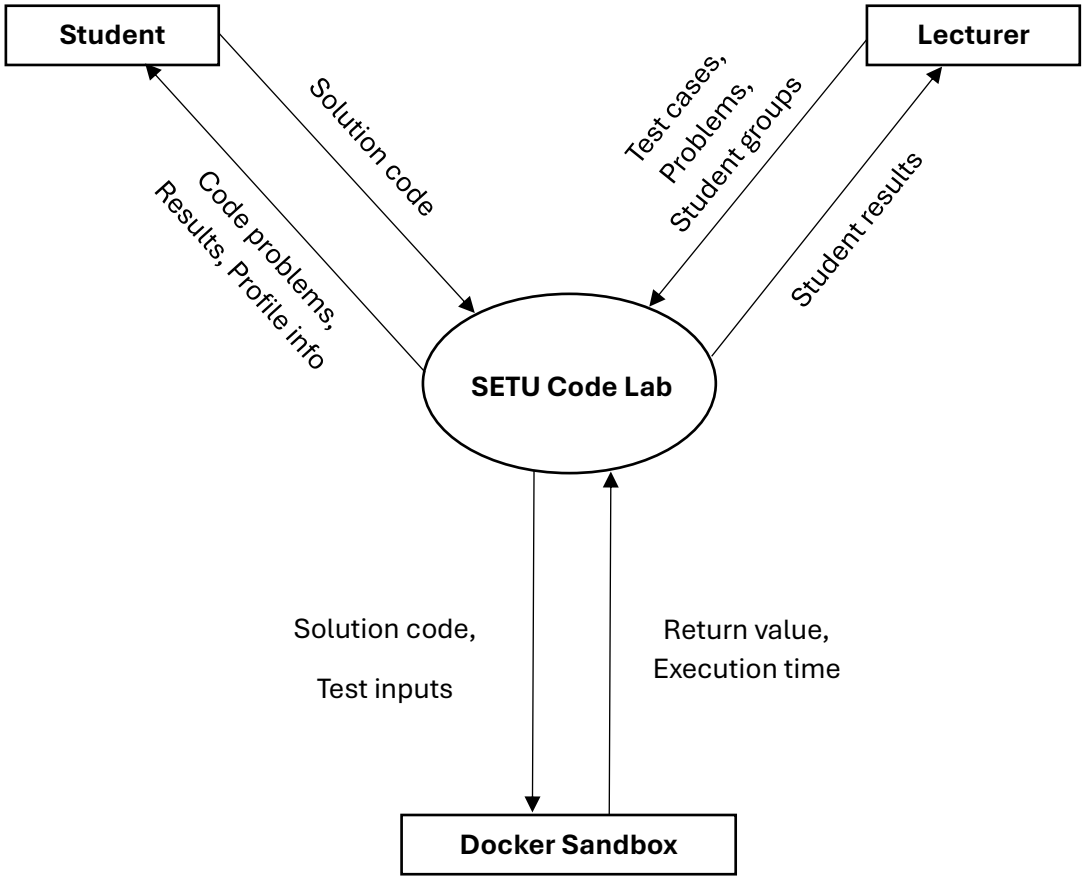

Fig. 1. Context Diagram

Use Cases

Use Case Diagram

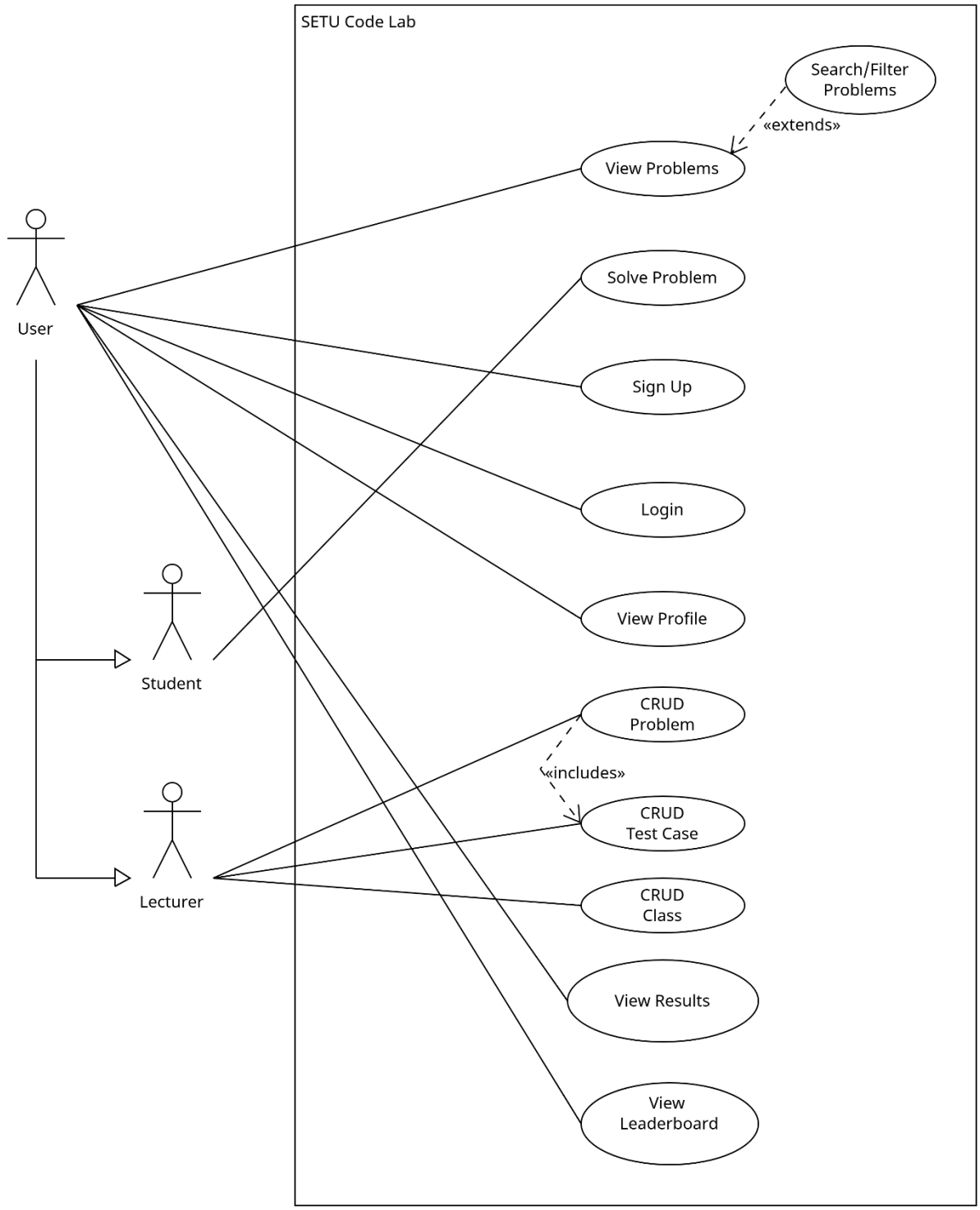

Fig. 2. Use Case Diagram

Brief Use Cases

View Problems

Actors: **Student, Lecturer**
Description: **This use case begins when a logged in user (student or lecturer) wishes to view the list of available coding problems. The use case ends when the system displays a list of available problems with relevant details such as title, difficulty level and topic.**

Search/Filter Problems

Actors: **Student, Lecturer**
Description: **This use case begins when a logged in user (student or lecturer) wishes to search or filter for specific problems. After viewing available problems, the user enters a search query into the search bar and presses enter. The user can also apply a filter by checking boxes (such as Easy, Medium or Hard for difficulty). The use case ends when the system displays the problems that user is looking for.**

Solve Problem

Actors: **Student**
Description: **This use case begins when a logged in student selects a coding problem they wish to solve. Upon selecting this problem, a detailed description of the problem is displayed, a built-in code editor, test cases, and buttons to run or submit the code. The student writes their solution into the code editor and may run it against the sample test cases. The use case ends when the student clicks submit which runs the code a final time, returns the results and saves the solution to the student's profile.**

Sign Up

Actors: **Student, Lecturer**
Description: **This use case begins when a user (student or lecturer) wishes to sign up to the platform. The user selects the "Sign Up" option on the homepage, selects whether they are a lecturer or a student and enters details such as email, name and password. The use case ends when the form is submitted. An account is then created using their details and they now are eligible to log in.**

Login

Actors: **Student, Lecturer**
Description: **This use case begins when a user (student or lecturer) wishes to log in to their existing account. The user enters their login credentials (email and password) into a form on the homepage. If the details are correct, they are logged into their account. This use case ends when the user completes the log in process.**

View Profile

Actors: **Student, Lecturer**
Description: **This use case begins when a logged in user (student or lecturer) wishes to view profile information. Students can view their own profile by clicking on "Profile" in the navigation bar to see past problems attempted and their results.** **Lecturers can view their own profile by clicking on "Profile" in the navigation bar. This shows them any problems and test cases they have created. They can also view student profiles by clicking on "My Classes", selecting which class they want to view and then selecting which student profile they want to see. This allows them to see student results and monitor their progress. The use case ends when the user can see the profile they are looking for.**

CRUD Problem

Actors: **Lecturers**
Description: **This use case begins when a logged in lecturer wishes to create, retrieve, update or delete a coding problem. The lecturer clicks on "Manage Problems" in their sidebar. Here they can see a list of their existing problems with options such as: "Create Problem", "Update Problem", "Delete Problem" and "Manage Test Cases". Upon selecting "Create Problem" They can then specify a title, description, type and difficulty. Additionally, at least one test case must be added before the problem can be submitted. Once created or updated the problem becomes available to students. Deleted problems are removed from the repository. The use case ends when the lecturer has created, retrieved, updated or deleted a coding problem.**

CRUD Test Case

Actors: **Lecturers**
Description: **This use case begins when a logged in lecturer wishes to create, retrieve, update or delete test cases for coding problems. The lecturer clicks on "Manage Problems" in their sidebar and then "Manage Test Cases" to create, retrieve, update or delete test cases on existing problems. Alternatively, if they are in the process of creating a new problem they can simply click "Add Test Case" to add a test case to the current problem. When creating a test case, they can specify an input and an expected output for the given problem. Once created or updated the test case becomes available with the associated code problem. Deleted test cases are removed from the database. The use case ends when the lecturer has created, retrieved, updated or deleted a test case.**

CRUD Class

Actors: **Lecturers**
Description: **This use case begins when a logged in lecturer wishes to create, retrieve, update or delete a class group. The lecturer clicks "Manage Classes" in their sidebar. Here they can see a list of their existing classes with options such as: "Create Class", "Update Class" and "Delete Class". Upon selecting "Create Class" lecturers can add multiple students and assign problems to the new class. The "Update Class" option allows lecturers to add new students to existing classes by entering their email and lets them assign new problems to the class. Assigned problems are visible to students in the class at the top of their problem list on their homepage. The use case ends when the lecturer has created, retrieved, updated or deleted a class.**

View Results

Actors: **Students, Lecturers**
Description: **This use case begins when a logged in user (student or lecturer) wishes to view results of past submissions. The user clicks on the profile of the user whose results they wish to view. Students can access other students' profiles through the public leaderboard. Lecturers can see students' profiles by navigating to their classes and clicking on student profiles from there. Results are then displayed on the student's profile with most recent submissions appearing first. Results include the problem description, the student's solution code, their test case outcomes, time taken, and execution time. The use case ends when the user has successfully viewed the desired results.**

View Leaderboard

Actors: **Students, Lecturers**
Description: **This use case begins when a logged in user (student or lecturer) wishes to view the public leaderboard. The user clicks “Leaderboard” in their sidebar. This takes them to the public leaderboard where they can see student profiles ranked by number of problems solved and solutions ranked by shortest execution time. The use case ends when the user has successfully viewed the desired leaderboard.**

Detailed Use Cases

View Problems

Actors:	**User (student or lecturer)**
Pre-conditions:	• The user is logged in.
Main Success Scenario:	1. The user navigates to the homepage after logging in. 2. The system retrieves available problems from the database. 3. The system displays the available problems and their relevant details (title, difficulty and topic). 4. The user can successfully view and browse available problems.
Post Conditions:	The user can successfully view a list of all available problems on their home page.
Alternative(s):	1a. There are no problems available to view. The system displays a message indicating that there are no problems currently available.
Difficulty	★

Search/Filter Problems

Actors:	User (student or lecturer)
Pre-conditions:	• The user is logged in. • There are problems available to view.
Main Success Scenario:	**Search Query** 1. The user enters a search query into the search bar. 2. The system queries the database for coding problems whose titles closely match the search query. 3. The system only displays problems that closely match the search query. **Filter Problems** 1. The user checks boxes to apply filters. (e.g. easy, medium and hard for difficulty). 2. The system queries the database for coding problems which have the desired tag. 3. The system only displays coding problems which have the selected tags.
Post Conditions:	The user can successfully view a filtered list of available problems on their homepage.
Alternative(s):	3a. There are no problems matching the search or filter query. The system displays a message indicating there are no problems available matching the current search query or filters.
Difficulty	★

Solve Problem

Actors:	Student
Pre-conditions:	• The student is logged in. • There are problems available to view.
Main Success Scenario:	1. The student selects a problem. 2. The system retrieves the problem details from the database (description and test cases). 3. The system displays the problem description, test cases, a built-in code editor, a run button and a submit button. 4. The student writes a solution into the code editor. 5. The student presses the submit button. 6. The system spins up a temporary docker container to safely compile and execute inputted code. 7. The docker container runs the code with test inputs and returns each output. 8. Docker container is destroyed 9. The system compares these outputs with expected outputs defined in test cases and compiles final results (pass/fail, and execution time). 10. The system displays a report containing pass/fail, time taken, execution time, and test case results to the student. 11. The results are saved to the student's profile.
Post Conditions:	The user can successfully view a report containing pass/fail, time taken, execution time and test case results for the problem they selected.
Alternative(s):	5a. The student presses the run button. The system will perform steps 6 and 7 as normal however the system will display test case results only, and the student will be able to continue editing their solution.
Difficulty	★★★★★

Sign Up

Actors:	User (student or lecturer)
Pre-conditions:	• User does not have an existing account.
Main Success Scenario:	1. A new user clicks sign up on the log in page. 2. The system displays a form requesting an email address, name, password, and account type lecturer/student. 3. The user inputs their details and submits the form. 4. The system validates the entered information. 5. If the details are valid the system creates a new account for the user and stores the information in the database. 6. The system confirms that the account has been successfully created and prompts the user to log in.
Post Conditions:	The system successfully creates an account using the entered details.
Alternative(s):	4a. the user enters invalid details. The system does not create an account and an error message telling the user what details are invalid is visible.
Difficulty	★

Login

Actors:	User (student or lecturer)
Pre-conditions:	• The user has an existing account.
Main Success Scenario:	1. The user navigates to the log in page. 2. The system prompts the user to log in. 3. The user enters their log in credentials (email address and password). 4. The system validates the credentials. 5. If the credentials are valid the system logs the user in and takes them to the homepage.
Post Conditions:	The user is successfully logged in and redirected to the homepage.
Alternative(s):	4a. the user enters invalid details. The system does not log the user in and an error message indicating that the email or password is incorrect.
Difficulty	★

View Profile

Actors:	User (student or lecturer)
Pre-conditions:	• The user is logged in. • There are other profiles available to view.
Main Success Scenario:	**Viewing Own Profile** 1. The user clicks on "Profile" in the navigation bar. 2. The system redirects to the user's own profile page. **Viewing Other Profiles** 1. The user clicks "Leaderboard" in the sidebar. 2. The system displays a leaderboard containing student names and links to their profiles. 3. The user clicks on a student name. 4. The system retrieves profile information including results from the database and displays this information to the screen. **Lecturer Viewing Student Profile from Class** 1. A Lecturer clicks on "My Classes" in the sidebar. 2. The system displays class groups associated with that lecturer. 3. The lecturer clicks on one of their classes. 4. The system displays a list of all students in that class with links to their profiles. 5. The lecturer clicks on a student name. 6. The system retrieves profile information including results from the database and displays this information to the screen.
Post Conditions:	The user can successfully view the selected profile (their own or another student's).
Alternative(s):	3a. The selected student profile cannot be retrieved. The system displays an error message indicating that the profile is unavailable.
Difficulty	★★

CRUD Problem

Actors:	Lecturer
Pre-conditions:	• The lecturer is logged in. • The lecturer has previously created problems.
Main Success Scenario:	**Create Problem** 1. The lecturer clicks “Manage Problems” in their sidebar. 2. The system redirects to the lecturer’s problem page. 3. The lecturer clicks “Create Problem”. 4. The system prompts the lecturer with a form requesting a title, description, type and difficulty. 5. The lecturer fills the “Create Problem” form out. 6. The lecturer clicks on “Add Test Case”. 7. The system prompts the lecturer with another form requesting an input and an expected output. 8. The lecturer fills the “Add Test Case” form out. 9. The lecturer submits the “Add Test Case” form. 10. The lecturer submits the “Create Problem “ form. 11. The system creates a problem entry in the database. 12. The system displays a success message. **Update Problem** 1. The lecturer clicks “Manage Problems” in their sidebar. 2. The system redirects to the lecturer’s problem page. 3. The system returns a list of problems created by the lecturer. 4. The lecturer clicks on “Update Problem” on a particular problem listing. 5. The system prompts the lecturer with a pre-filled form allowing them to edit the title, description, type and difficulty. 6. The lecturer makes some edits to the form. 7. The lecturer submits the “Update Problem” form. 8. The system updates the problem entry in the database.

	9. The system displays a success message. **Delete Problem** 1. The lecturer clicks “Manage Problems” in their sidebar. 2. The system redirects to the lecturer’s problem page. 3. The system returns a list of other problems created by the lecturer. 4. The lecturer clicks “Delete Problem” on a particular problem listing. 5. The system prompts the lecturer to confirm if they really want to delete the problem. 6. The lecturer selects confirm. 7. The problem and associated test cases are deleted from the database. 8. The system displays a success message.
Post Conditions:	**Create Problem** The lecturer successfully creates a problem that can be viewed by them and other users. **Update Problem** The lecturer successfully updates a problem. The updates can be seen by them and other users. **Delete Problem** The lecturer successfully deletes a problem. The problem is no longer visible to them or other users.
Alternative(s):	**Create Problem** 9a. The lecturer fills in invalid details. The system will display an error message showing exactly where and why the error is occurring. The form is not submitted. 10a. The lecturer fills in invalid details. The system will display an error message showing exactly where and why the error is occurring. **Update Problem** 7a. The lecturer fills in invalid details. The system will display an error message showing exactly where and why the error is occurring. The form is not submitted. **Delete Problem** 5a. The lecturer cancels the deletion of a problem instead. The system will not delete the problem.
Difficulty	★★★

CRUD Test Case

Actors:	Lecturer
Pre-conditions:	• The lecturer is logged in. • The lecturer has previously created problems. • The lecturer has previously created test cases on some problems.
Main Success Scenario:	**Create Test Case** 1. The lecturer clicks “Manage Problems” in their sidebar. 2. The system redirects to the lecturer’s problem page. 3. The system returns a list of problems created by the lecturer. 4. The lecturer clicks on “Manage Test Cases” on a particular problem. 5. The system returns a list of test cases created by the lecturer for that problem. 6. The lecturer clicks “Add Test Case”. 7. The system prompts the lecturer with a form requesting an input and an expected output. 8. The lecturer fills in the “Add Test Case” form. 9. The lecturer submits the “Add Test Case” form. 10. The system creates a test case entry on that problem. 11. The system displays a success message. **Update Test Case** 1. The lecturer clicks “Manage Problems” in their sidebar. 2. The system redirects to the lecturer’s problem page. 3. The system returns a list of problems created by the lecturer. 4. The lecturer clicks on “Manage Test Cases” on a particular problem. 5. The system returns a list of test cases created by the lecturer for that problem. 6. The lecturer clicks on “Update Test Case” on a particular test case entry. 7. The system prompts the lecturer with a pre-filled form allowing them to edit the test input and test output values. 8. The lecturer makes some edits to the form.

	9. The lecturer submits the “Update Test Case” form 10. The system updates the test case entry in the database. 11. The system displays a success message. **Delete Test Case** 1. The lecturer clicks “Manage Problems” in their sidebar. 2. The system redirects to the lecturer’s problem page. 3. The system returns a list of problems created by the lecturer. 4. The lecturer clicks on “Manage Test Cases” on a particular problem. 5. The system returns a list of test cases created by the lecturer for that problem. 6. The lecturer clicks “Delete Test Case” 7. The system prompts the lecturer to confirm if they really want to delete the test case. 8. The lecturer selects confirm. 9. The test case is deleted. 10. The system displays a success message.
Post Conditions:	**Create Test Case** The lecturer successfully creates a test case for the specified problem. **Update Test Case** The lecturer successfully updates a test case entry for a specified problem. **Delete Test Case** The lecturer successfully deletes a test case for a specified problem.
Alternative(s):	**Create Test Case** 9a. The lecturer fills in invalid details. The system will display an error message showing exactly where and why the error is occurring. The form is not submitted. **Update Test Case** 9a. The lecturer fills in invalid details. The system will display an error message showing exactly where and why the error is occurring. The form is not submitted. **Delete Test Case** 8a. The lecturer cancels the deletion of a test case. The system will not delete the test case.
Difficulty	★★★

CRUD Class

Actors:	Lecturer
Pre-conditions:	• The lecturer is logged in. • There are student profiles in the database. • The lecturer has previously created problems. • The lecturer has previously created classes.
Main Success Scenario:	**Create Class** 1. The lecturer clicks “Manage Classes” in their sidebar. 2. The system redirects to the lecturer’s classes page. 3. The lecturer clicks on “Create Class”. 4. The system prompts the lecturer with a form requesting a class name, student emails, and problems. 5. The lecturer fills in these details. 6. The lecturer selects problems from a list of all problems to assign them to the class. 7. The lecturer submits the “Create Class” form. 8. The system creates a class using the inputted details. 9. The system displays a success message. **Update Class** 1. The lecturer clicks “Manage Classes” in their sidebar. 2. The system redirects to the lecturer’s classes page. 3. The system returns a list of classes created by the lecturer. 4. The lecturer clicks on “Update Class” on a particular class. 5. The system prompts the lecturer with a pre-filled form allowing them to edit the class name, student emails and problems that are assigned. 6. The lecturer makes some edits to the form. 7. The lecturer submits the “Update Class” form. 8. The system updates the class entry in the database. 9. The system displays a success message.

	Delete Class 1. The lecturer clicks “Manage Classes” in their sidebar. 2. The system redirects to the lecturer’s classes page. 3. The system returns a list of classes created by the lecturer. 4. The lecturer clicks on “Delete Class” on a particular class. 5. The system prompts the lecturer if they really want to delete the class. 6. The lecturer selects confirm. 7. The class is deleted from the database. 8. The system displays a success message.
Post Conditions:	**Create Class** The lecturer successfully creates a class with the specified details. **Update Class** The lecturer successfully updates a class with the specified details. **Delete Class** The lecturer successfully deletes the specified class.
Alternative(s):	**Create Class** 7a. The lecturer fills in invalid details. The system will display an error message showing exactly where and why the error is occurring. The form is not submitted. **Update Class** 7a. The lecturer fills in invalid details. The system will display an error message showing exactly where and why the error is occurring. The form is not submitted. **Delete Class** 6a. The lecturer cancels the deletion of a class. The system will not delete the class.
Difficulty	★★★

View Results

Actors:	User (student or lecturer)
Pre-conditions:	• The user is logged in. • There are other profiles with results.
Main Success Scenario:	1. The user navigates to a student's profile page. 2. The system retrieves student's submission history and related results from the database. 3. The system displays these results on the student's profile page.
Post Conditions:	The user can successfully view that student's results on their profile page with the most recent submissions appearing first. Results include the problem description, the student's solution code, their test case outcomes, time taken, and execution time.
Alternative(s):	2a. The student has not completed any problems yet. The system displays a message indicating this on the student's profile.
Difficulty	★

View Leaderboard

Actors:	User (student or lecturer)
Pre-conditions:	• The user is logged in. • There are other profiles with results
Main Success Scenario:	**Number of Problems Solved Leaderboard** 1. The user clicks “Leaderboard” in the sidebar. 2. The system navigates to the leaderboard page and retrieves the top ten student profiles ranked by number of problems completed. 3. The system displays this ranking with profile names and number of problems solved. **Fastest Solutions Leaderboard** 1. The user clicks “Leaderboard” in the sidebar. 2. The user clicks “Fastest Solutions” 3. The system retrieves and displays a list of problems. 4. The user selects a particular problem. 5. The system retrieves the top ten fastest solutions for the selected problem and the profiles of the students’ who submitted them. 6. The ranked information is displayed to the screen.
Post Conditions:	The user can successfully view the desired leaderboard.
Alternative(s):	**Number of Problems Solved Leaderboard** 2a. There are no student profiles. The system will display a message indicating this. **Fastest Solutions Leaderboard** 3a. There are no problems. The system will display a message indicating this. 5a. No student has completed this problem yet. The system will display a message indicating this.
Difficulty	★★

Precedent and Inspiration

The design of SETU Code Lab is inspired by existing platforms such as LeetCode, HackerRank and CodeWars. These platforms have built-in code editors, code problems and test cases that submitted code must pass. The code problems on these platforms come from commonly asked coding interview questions. These features work well to gamify the learning and make practicing coding concepts easier.

SETU Code Lab aims to incorporate similar features with an added emphasis on education and student lecturer engagement. Lecturer created problems can be tailored towards computing modules in SETU increasing their relevance and usefulness for students. This would also allow Lecturers to use the platform as a tool for practical lab work and simplify the grading process as they could see every student's solution in one place. This functionality is not available in LeetCode, HackerRank or CodeWars and aims to save lecturers and students time and effort.

Success Metrics

The criteria that the system must meet to be considered a success are as follows:

Functional Metrics

- The system allows students to select and complete coding problems
- The system supports at least one language (Java) for code execution and results generation
- The system allows Lecturers to create, update and delete coding problems and test cases
- Users can view problems, results, profiles and leaderboards
- Lecturers can create, update and delete classes

Non-Functional Metrics

- The system shall ensure submissions cannot access the server file system or network
- The system shall ensure that only authorized users (lecturers and students) can access their respective dashboards
- The system shall be supported on the latest 2 versions of Firefox, Google Chrome, and Microsoft Edge
- The system shall remain usable (all previous usability tests pass) and readable on the top 5 most common desktop resolutions in Europe (1920x1080, 1536x864, 1366x768, 1280x720, 2560x1440)[1]
- 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
- Lecturers with existing accounts should be able to log in, create a new code problem, write something, and upload it with no prior training in ≤ 120 seconds
- The system shall have 99% uptime
- The system shall return code execution results in ≤ 5 seconds for problems ≤100 lines of code and ≤10 test cases

Project Timeline

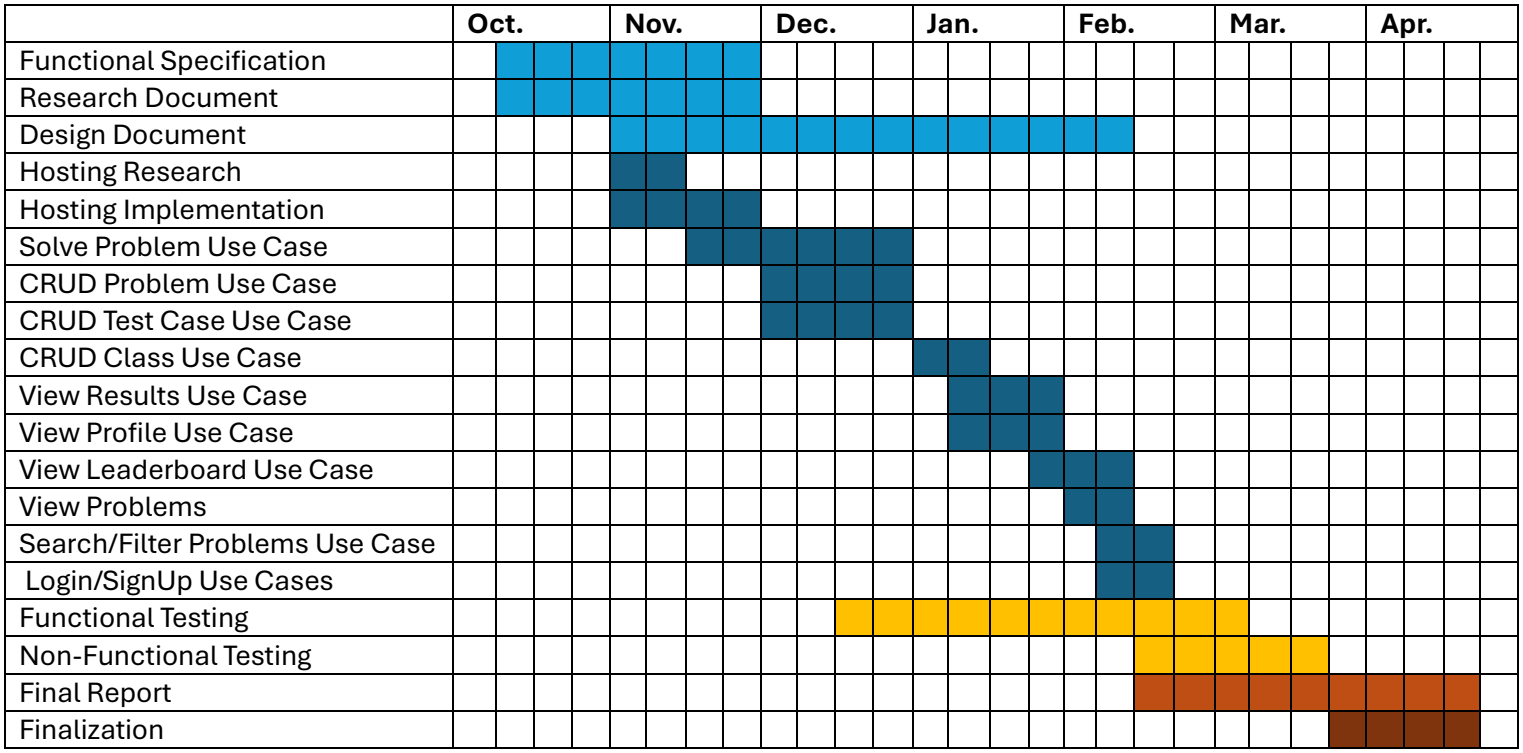

Fig. 3. Project Timeline Gantt Chart

Task	Start Date	End Date	Duration (Days)
Research	**06/10/2025**	**01/12/2025**	**56**
Functional Specification	06/10/2025	01/12/2025	56
Research Document	06/10/2025	01/12/2025	56
Development	**01/11/2025**	**23/02/2026**	**115**
Design Document	01/11/2025	23/02/2026	115
Hosting Research	01/11/2025	16/11/ 2025	16
Hosting Implementation	01/11/2025	01/12/2025	30
Solve Problem Use Case	16/11/2025	01/01/2026	46
CRUD Problem Use Case	01/12/2025	01/01/2026	31
CRUD Test Case Use Case	01/12/2025	01/01/2026	31
CRUD Class Use Case	01/01/2026	15/01/2026	15
View Results Use Case	08/01/2026	01/02/2026	24
View Profile Use Case	08/01/2026	01/02/2026	24
View Leaderboard Use Case	26/01/2026	14/02/2026	19
View Problems	01/02/2026	14/02/2026	14
Search/Filter Problems Use Case	09/02/2026	23/02/2026	14
Login/SignUp Use Cases	09/02/2026	23/02/2026	14
Testing	**19/12/2025**	**16/03/2026**	**88**
Functional Testing	19/12/2025	08/03/2026	81
Non-Functional Testing	14/02/2026	16/03/2026	30
Finalization	**14/02/2026**	**24/04/2026**	**69**
Final Report	14/02/2026	24/04/2026	69
Finalization	23/03/2026	24/04/2026	32

Fig. 4. Project Timeline Dates Chart

Bibliography

Statcounter, 2025. *Desktop Screen Resolution Stats Europe* [Online].
Available at: https://gs.statcounter.com/screen-resolution-stats/desktop/europe
[Accessed 4 October 2025].

Meta Platforms, 2025. *Quick Start - React*. [Online]
Available at: https://react.dev/learn
[Accessed 7 November 2025].

Microsoft, 2025. *TypeScript Documentation – The Starting Point for Learning TypeScript*. [Online]
Available at: https://www.typescriptlang.org/docs/
[Accessed 7 November 2025].

SASS Team, 2025. *SASS Documentation*. [Online]
Available at: https://sass-lang.com/documentation/
[Accessed 7 November 2025].

Haverbeke, M., 2025. *CodeMirror Documentation*. [Online]
Available at: https://codemirror.net/
[Accessed 7 November 2025].

OpenJS Foundation, 2025. *Node.js - Introduction to Node.js*. [Online]
Available at: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs
[Accessed 7 November 2025].

OpenJS Foundation, 2025. *Express - Node.js Web Application Framework*. [Online]
Available at: https://expressjs.com/
[Accessed 7 November 2025].

PostgreSQL, 2025. *PostgreSQL: Documentation*. [Online]
Available at: https://www.postgresql.org/docs/current/
[Accessed 7 November 2025].

Docker, inc., 2024. *What is Docker? | Docker Docs*. [Online]
Available at: https://docs.docker.com/get-started/docker-overview/
[Accessed 7 November 2025].

DigitalOcean, 2025. *VPS Hosting Plans – Starting at $4/mo | DigitalOcean*. [Online]
Available at: https://www.digitalocean.com/solutions/vps-hosting
[Accessed 12 November 2025].