



6CCS3PRJ Final Year Platform for Managing Coding Problems and Submissions

Final Project Report

Author: Ali Zaini

Supervisor: Dr Christopher Hampson

Student ID: 1806270

April 6, 2022

Abstract

This report will cover the development of Proto, a full-stack web application for creating, solving, and automatically assessing programming problems and submissions. It also includes features to build classrooms, invite users, set assignments, and give feedback. Developers can utilise the modular backend in various contexts such as university assignments, coding challenges, hackathons, and employability examinations. The accompanying frontend aims to showcase how this backend can be implemented with a user-friendly web application.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Ali Zaini

April 6, 2022

Acknowledgements

I would like to thank Dr Christopher Hampson for his supervision and advice throughout this project. His reassurance of the direction of this project and feedback for improving on my work helped me immensely. I would also like to thank the many lecturers, teaching assistants, and others involved in supporting my education these last couple of years. Finally, I would like to thank my friends for making things a little less boring and my family for supporting me throughout everything.

Contents

1	Introduction	3
1.1	Introduction and Motivations	3
1.2	Aims	3
2	Background	5
2.1	Relevant Literature	5
2.2	Existing Solutions	12
3	Specification	19
3.1	Functional Requirements	19
3.2	Non-functional Requirements	24
4	Design	25
4.1	Technology	25
4.2	Frontend Prototyping	36
5	Implementation	38
5.1	Tooling & Methodology	38
5.2	Architecture	41
5.3	Testing	58
6	Evaluation	61
6.1	Functional Requirements	61
6.2	Non-functional Requirements	64
6.3	Limitations	67
7	Legal, Social, Ethical and Professional Issues	68
7.1	Licensing	68
7.2	Ethics	68
7.3	British Computer Society	69
7.4	General Data Protection Regulation	70
8	Conclusion & Future Work	71
8.1	Conclusion	71
8.2	Future Work	71
	Bibliography	77

A	User Guide	78
A.1	Instructions	78
A.2	Requirements	78
A.3	Setup + Installation	78
A.4	Testing	80
A.5	Walkthrough	80
B	Source Code	82
B.1	Declaration	82

Chapter 1

Introduction

1.1 Introduction and Motivations

There are many contexts where it is important to assign a coding assignment to many people, have their code automatically run against a suite of tests, and be able to give feedback. In talent acquisition, technical selection tests are a common way of assessing candidates as it lowers costs, improves efficiency, and provides objectivity and standardisation [39]. As the popularity of programming courses increases, teachers will face additional difficulties in marking code and providing individual feedback in a timely manner [16].

1.2 Aims

This project aims to address these issues by empowering assessors through a platform that allows for creating programming questions that can be automatically assessed against a suite of test cases and enable assessors to mark and provide feedback quickly. This paper will outline the development of this platform alongside a web application implementation in the context of a university course to showcase how it could be implemented. It should then be possible to implement the same backend platform in other contexts such as Moodle plugins [31], games, and mobile applications.

Users should be able to log in, find problems to work on (or those they have been assigned), write code using an online IDE (Integrated Development Environment), submit their code, run code against test cases, and receive feedback on their results. Assessors should be able to view the submissions and provide feedback. Within the university context, teachers should be

able to create classrooms and invite students to join them to then set assignments, monitor which problems students struggle with and export the results for other pedagogical purposes. A similar process is performed when assessing software engineers during hiring.

Chapter 2

Background

This chapter covers relevant literature to introduce the project space and terminology used throughout the report, and areas of interest and relevant issues are summarised. It ends with a critical evaluation of some existing systems, which aim to tackle adjacent problems and explore their approaches, strengths, and weaknesses to assist in defining the specification for this project.

2.1 Relevant Literature

2.1.1 A Survey on Online Judge Systems and Their Applications

Summary

Wasik et al. [42] introduces the types of systems that interest this project. It focuses on online judge systems used for the reliable evaluation of code submitted by users and explores their different applications. Notable areas of use include education, recruitment, and competitive programming. The paper discusses various existing systems, which we will explore in more detail in the next section of this chapter.

The paper defines online judges as systems designed for evaluating source code submitted by users by compiling and testing the code in a homogeneous environment. Online judges will be an essential component of the platform built, as users will submit code that is evaluated against a suite of test cases.

A crucial component of online judge systems is the evaluation procedure. It consists of submission, assessment, and finally, scoring. During submission, the code is compiled (if needed) and executed in a homogeneous evaluation environment. The submission is then tested against

a set of test cases for that problem and scored based on the test cases it passes. Successful execution must proceed without errors, without exceeding resource limitations and that the outputs match the expected outputs for the problem.

Although the paper discusses some methods of developing an online judge system, it clarifies the complexities of such a system, such as handling different languages and versions. Because of this, it is worth exploring the existing open-source online judge system such as Judge0.

The paper identifies combinatorial problems as the most commonly used with online judge systems. It defines them as relying on discovering values for variables that satisfy specific constraints. They are further divided into decision problems, where it has to be verified that a solution exists, and search problems, where a particular solution has to be found. These definitions help lay out the mechanism by which we can build a system for creating problems by writing them as combinatorial problems with set inputs and expected outputs for the test cases.

The paper also explores the topic of competitive programming, which is the most popular use of online judge systems. There are many implementations, such as CodeForces, which has over 3,000 problems and over 325,000 users. We will explore similar websites such as HackerRank in the next section of this chapter.

Recruiting faces a similar problem as in education, where they have to assess many applicants and doing so manually is not always feasible. Often an online judge will be used as the first step of screening before a human will interview and examine the applicant. For recruitment, the paper explores platforms such as Codility, HackerRank, and Leetcode.

The paper explores online judge systems in Massive Open Online Courses (MOOCs) and games for teaching programming. It describes the benefits of online judges in education as allowing staff to assess student code automatically. By giving assessors the freedom to write comprehensive tests covering all edge cases, they can quickly be sure if the code is correct. Since code is automatically assessed, the time for evaluation is much shorter, so assessors could assign more problems. Students also get instant feedback on how their code performs on test cases. This project aims to include these benefits.

Online judge systems are used in games and many other domains. This highlights the importance of developing a platform that can be implemented for various contexts, not just in education or recruiting.

2.1.2 Coderunner: A Tool for Assessing computer Programming Skills

Summary

Handwritten code is hard to write and read and even harder to grade. This paper discusses the successes of CodeRunner; a Moodle [31] plugin which provides a tool to assess programming skills and give immediate feedback to students based on the correctness of their answers [27]. It can impose penalties for submissions and adjust input values, making it customisable. It supports many text-based programming languages, including C, Java, Python, PHP, and JavaScript. These extensibility measures and support for multiple languages are essential for this project too. Initially, the project may just support one language and make it simple to expand to support more.

This Moodle plugin is free and open-source and can be integrated seamlessly with other Moodle features, such as “description” questions that might include tutorial information. This integration with another platform is helpful as it lets the results from the plugin be stored within the school’s management system. As this project aims to be applicable in many contexts, adding a method to export results so they can be imported into other systems would be an excellent way to provide similar functionality. An advantage of this is that this project will not necessitate the use of Moodle and its limitations. One other limit of CodeRunner is that it only works for single file code, which can be troublesome for larger projects.

CodeRunner serves an important subset of the target demographic for this project. The paper discusses the benefits of automated code judges. For example, students are motivated to take the time to work on problems to achieve the “green ticks” rather than move to different questions. Instant feedback is something students like and further maintains motivation. It has become a core part of how many computer science courses are taught and assessed. Students get used to the platform, so it is not as stressful when used for online tests. Teachers enjoy devising and implementing good questions. finding it both challenging and rewarding. This project should include a seamless and straightforward solution for creating new problems.

2.1.3 Optimization and Improvements of a Moodle-Based Online Learning System for C Programming

Summary

Similar to CodeRunner, this paper is about a Moodle-based interactive teaching platform focused on teaching the C programming language [40]. It has an online judge system to grade

students' code automatically. This paper is focused on optimising and improving that system, mainly to address five problems according to feedback from teachers and students:

1. Logical errors in code cannot be located.
2. Cheating by directly outputting answers cannot be detected.
3. Evaluation results lack statistics and visualisation.
4. Code submitting procedure is complicated.
5. Feedback for incorrect answers is not detailed.

Similar to others, this paper outlines some more of the benefits of an online judge system, mainly the following:

- Access to an all-day online judge system means users can complete assignments anywhere & anytime. This enables users to self improve in their own time.
- Trains students in the rigour of programming by having to pass test cases.
- Teachers can easily view the code students submit and assess their mastery of knowledge.
- Teachers do not need to spend as much time and effort evaluating student code, which saves time and money that can be focused on other aspects of teaching.
- It is automatic and allows students to quickly and accurately obtain feedback, which is more objective.

The paper discusses other uses of online judge systems; however, we have already covered this with the first paper.

The paper outlines its architecture and solutions to the five initial problems. The statistics they collect are interesting, such as the pass rates for each test case and the overall problem. Their solutions for other issues are not as relevant to our project. For example, their old submission procedure involved uploading a file with the code. They improved on this by including a text box to paste the code instead. This is irrelevant as this project will have a text editor with useful features such as syntax highlighting as the primary way of submitting code. The paper gives a great example of how they map the online judge results to messages shown to the user. Our messages will have to depend on the capabilities of our online judge system.

Although our project will likely depend on an existing online judge system (i.e. Judge0), this paper does raise some critical problems and suggests some valuable solutions to consider. Here is an outline of how this project may address these five initial problems:

1. I will rely on an existing online judge system to handle the errors and provide error messages to the user through the frontend.
2. Cheating by directly outputting answers will be handled by having hidden test cases. The submitter will not see the solution to the test case and so cannot directly output the result.
3. I can generate statistics for our problems and assignments. These will be helpful for assessors in monitoring which problems are difficult or the common pitfalls of some problems. Visualisations can be done using third party frontend frameworks for graphs.
4. The code submission procedure will be simple, with clear buttons and navigation inspired by other well established coding platforms.
5. As well as the feedback such as which test cases passed and any errors, the feedback system can have a mark and comments provided by an assessor and some general advice based on the types of errors users get.

2.1.4 Building a Comprehensive Automated Programming Assessment System

Summary

This paper develops Edgar, an Automated Programming Assessment Systems (APAS), which is a type of online judge system [28]. Much like the other papers, it highlights the increasing demand for computer science courses by students and the low supply of teachers, leading to various challenges.

It is very similar to CodeRunner but focuses on improving how it handles SQL assignments. This feature is not of much interest to this project as it is currently intended to support mainly only text-based programming languages such as Python or JavaScript. This paper distinguishes between “standalone” languages, which come with their own compiler/interpreter and “managed” languages. “Managed” languages such as SQL or HTML are executed within a context (such as a database or web DOM.)

Whereas CodeRunner simply adds a question-type to Moodle, Edgar is a web application built using the Angular framework. It also supports cross-platform mobile applications for Android and iOS, written in React Native.

It supports creating assignments and exams with various question types, including multiple-choice, free text (which is not evaluated), SQL, JSON and code.

Question type	Comparison method	Comparison parameters
Multiple choice	trivial: comparison with stored correct answers;	
Free text	non-evaluated questions, used in questionnaires for gathering feedback	
SQL	questions evaluated against a database returning a record set (comparison of two record sets)	ignore tuple or columns order and/or column names
JSON	questions evaluated against some engine returning JSON (comparison of two objects)	deep or strict comparison
Code	programming language questions, e.g. Java, C, C#, C++, Python, that get compiled and executed in a sandbox environment (comparison of expected and actual standard output)	ignore case, trim whitespaces, use regular expression matching, use random input data

Figure 2.1: TABLE 2 of the paper, showcasing different question types

Like many projects which utilise an online judge, Edgar partially relies on Judge0, further supporting it as a good option for this project too.

Edgar also allows for conducting exams on the platform. It provides a dashboard for teachers to monitor students as they do the exams to check for cheating, such as losing focus from the exam window. Although their approach to this feature is very interesting, it is also outside the scope of what is initially intended for this project.

The paper also highlights many of the benefits of an online judge system and the issues with their implementation. They highlight some studies which show that human graders are less accurate than automatic evaluators and that they can increase students' scores. They also enable students to have more actual programming assignments since teachers will not need to spend time marking them. The web application for this project could allow students to work on programming problems even if a teacher has not set them a classroom assignment, allowing students to practice more coding in their own time. This would showcase how the backend platform is agnostic to how developers want to utilise it.

Some of the issues with such systems are the kind of feedback they provide, the submission policies (such as only allowing one submission or multiple submissions), and the unwanted student behaviour that might be enabled. The paper highlights that opinions about unlimited submissions vary. Some argue that unlimited submissions are essential for assessments done for formative reasons. In contrast, others say that multiple submissions allow students to exploit feedback to do a trial-and-error strategy. Again, this highlights the importance of the backend platform to be agnostic to most concrete use cases and instead be modular so that developers can choose to implement whatever strategies they want.

The paper also discusses some strategies for the feedback given to students after a submission. For example, feedback about which test cases passed or failed shows the difference between the actual and expected outputs. Hints can also be given as feedback. These feedback

types would be relevant to this project and could be implemented for public test cases and maybe even hidden test cases. The paper suggests that feedback generation is an area where automated systems lack compared to human evaluators. In some contexts (such as recruiting), it is preferred not to provide any feedback directly related to the submission. Because of this, the backend platform should overdeliver by providing feedback data and then developers can opt not to include that feedback when integrating the backend into their application.

2.1.5 Developing a self-regulated oriented online programming teaching and learning system

Summary

This paper develops LETJS (Learning with Entertainment and Training in JavaScript), a web application similar to Edgar [22]. Students and teachers can log in and access the resources. For example, learners can access any chapter to study at their own pace.

One feature from this paper that is of interest is the ability to share code and get feedback. Students can share their code, and peers can comment and rate it out of five stars. This code review and sharing could be beneficial for our project since users will be able to create problems, and it would make sense for it to be possible to share solutions in some capacity.

However, the design of the application is outdated and very much focused on a single language and is not as extensible as we would like. Beyond that, this paper goes over some studies that suggest that their platform enhances students' programming ability.

2.1.6 Robust and Scalable Online Code Execution System

Summary

This paper focuses on the development of an online judge system called Judge0 [14]. This will be an essential part of the system that this project develops.

The authors reiterate the importance of programming, which is recognised as a core competency skill and the high demand for computer science education. It also acknowledges the expansion of distance learning and online courses, which greatly need online judge systems.

The paper goes over the details of their implementation and architecture. They give an example of how the ecosystem architecture of competitive programming platforms, e-Learning platforms or interview preparation platforms would be structured. Typically, the platform has a frontend with an online IDE that the user can write code into and then send it to an online

judge. It is run through the code execution system in a sandboxed environment. Understanding this system helps us know its limitations and any issues that may arise.

Judge0 is the product of this paper, an open-source online code execution system that is used as an online judge. The system is scalable, robust, easy to use (by having a simple JSON web API, deployed using Docker), extendable, and open-source, making it a perfect fit for the needs of this project. It is also easily integrated into web applications and has been used by more than 3,000 students at the University of Zagreb. Judge0 also deals with security issues associated with executing arbitrary code submissions by considering all submissions as untrusted and running them in a sandboxed environment.

Although Judge0 fills the role of actually executing code, it does not provide the other features that would make the tool directly useful for an application, such as recruiting assessment or education platform. Judge0 only takes in code and executes it. This project will include additional functionality such as allowing for the creation of problems, users, classrooms, assignments and feedback and utilise Judge0 for the code execution portion of the project.

2.2 Existing Solutions

2.2.1 Codecademy

Codecademy is an online platform that offers free coding lessons in various languages to over 45 million users. It provides courses on computer science, web development, data science, machine learning and interview preparation. These courses typically include instructions and some coding environment to write code in a chosen language and then run it.

Relevant Features

Codecademy is very relevant to this project in that it includes many features that this project also aims to implement. It is an example of an application that the backend platform should be able to be used to create.

It includes a Catalog which has all the different courses they offer, including for various languages and subjects. Users can view the popular and new courses.

Of particular interest to this project is the ‘Challenges’ page which includes various coding challenges based on technical interviews. This is currently available in Python, JavaScript, and Java and questions are labelled by difficulty and topic. When users open a question, it has a description and some pre-written template code to get them started and a section to see the

result of running their code against some test cases and the current output of their code, such as for logging/testing purposes. Their code editor also adapts for web development projects by including a simple browser to preview code changes.

Codecademy also allows users to create a ‘Workspace’ to write and run their code in any language.

Completing courses awards users badges that can be viewed on their profile, which is a great motivator. It also allows users to set goals and weekly targets for practising and earning badges.

Codecademy also offers community features such as forums and chatrooms via Discord.

Drawbacks

Codecademy is a full-stack solution, not a backend platform that can be implemented for different use cases.

One drawback of Codecademy is the cost. Many of its courses require a PRO membership, which can cost up to £31.99 per month. This can be a prohibitive cost for many people, particularly students and people from developing countries looking to learn to code.

The platform also does not have a way to create and share coding challenges, as only Codecademy themselves can do this. This project’s platform will provide the functionality to create new problems.

Codecademy no longer offers ‘Codecademy Teacher Resources’, which was used to enable sending assignments to students and tracking their progress. Now, Codecademy recommends that teachers use Google Classroom and simply send the assignment link to students and manage the entire classroom environment using Google Classroom. [11] This has a few obvious drawbacks, most notably that it is not a native implementation with Codecademy and so lacks many of the valuable insights that could be generated, such as completion rate, quickly seeing where students struggle and so on. Secondly, it requires already having students on Google Classroom. Not all schools use this platform and may use something like Microsoft Teams or Moodle instead. This project aims to be agnostic of the specific integrations developers want to make. However, at a minimum, it should be possible to export data about how users have performed on assignment problems.

Summary

Codecademy is an excellent resource for learning to code and is an example of the kind of application this platform should be able to facilitate by providing key functionality such as the ability to create problems and submit code to be executed and get feedback. Codecademy, along with the other existing applications to be examined, inspires the frontend portion of this project to give an example of how to build a fully functioning coding application based on the backend platform.

2.2.2 HackerRank

HackerRank is an online platform for learning to code and doing competitive programming challenges. It focuses on providing solutions for recruiting and is a popular option for companies to send out automated online assignments for their applicants, often as an early stage of the interview process. It supports many languages, frameworks and topics, and it provides many features and products we will explore.

Relevant Features

HackerRank provides the option to create a set of questions and send them to applicants for jobs. These can have various question types, such as multiple choice or custom coding questions or use questions from HackerRank's existing library.

CodePair is a feature that allows interviewers to do real-time video calls while working in a shared coding environment. This is a compelling feature and can also be helpful for students too.

Outside of interviewing, HackerRank allows users to solve many different questions on their site, such as for particular programming languages or specific skills such as algorithms, data structures, mathematics and databases. They also have various 'Tutorials', collections of coding challenges to teach the basics. For example, the 'Interview Preparation Kit' is a curated list of challenges broken down into various topics commonly asked in job interviews.

They also have competitions and leaderboards, which is how they use gamification to make programming more fun.

Their code editor is very customisable, as it can be resized, have the themes changed and includes syntax highlighting. Users can upload their code as a file rather than use the provided IDE. When running code, users can use public test cases to check their code. There are also hidden test cases to help mitigate cheating. Users can also enter custom input to test their

code.

Some questions include an ‘Editorial’, a walkthrough of a solution provided by HackerRank. Although it is not free, it can be unlocked using points that users earn by completing questions. There is also a discussion page for users to discuss questions and solutions. The leaderboard for challenges shows who has completed the question, their username, rank, score, language, countries, and the option to view their actual code.

All these features are great, and HackerRank even provides a teaching environment where teachers can create assignments with due dates [20]. These can have multiple different questions with different question types and is a very similar experience to the environment used for job interview candidates. Teachers can then view reports for their assignments to see how users answered questions and what scores they got and be alerted about any potential plagiarism. Assignments are automatically graded but can be manually reviewed.

All their coding environments include plagiarism detection, which is very powerful because of the data they have collected from their over 16 million users. This is very important in education and recruiting.

HackerRank provides an editor that walks users through the steps, with code stubs and test cases for creating new questions.

Drawbacks

HackerRank offers various products, but developers cannot use HackerRank’s features in their custom use cases as they are not open-source.

HackerRank has a complex pricing model based on the features and number of people. However, most of its core features for users looking to complete coding challenges are free.

Their system for coding assignments is very powerful and helpful but is not integrated with any learning platforms such as Moodle.

Summary

HackerRank is a compelling platform that has it all. It’s an excellent platform for learning to code, doing coding challenges, interviewing candidates and setting student assignments. It does come with a price for these more advanced features, but it is entirely free for most users. Its teaching tools could be improved by having a way to export results easily and integrate with learning platforms such as Moodle.

2.2.3 Leetcode

LeetCode is another online coding platform and is very popular for interview preparation as it has many questions which companies typically ask. It has some other features which make it an interesting online judge system. However, it is not built for school education environments.

Relevant Features

LeetCode has over 2,000 questions, broken up into three difficulty categories (Easy, Medium, Hard) and various topics such as Arrays, Strings and Hash Tables. They provide solutions to some questions, but some of these solutions require a paid ‘Premium’ account. Some questions also include hints. LeetCode also suggests similar questions and related topics.

They provide basic statistics to free users, such as likes/dislikes, acceptance rate and the number of submissions for a question. These statistics are nice to have as it allows users to see if a question is worth attempting as some may be of lower quality. Anyone can submit a question to LeetCode, which can be added once approved by moderators. This limits users from being able to create and share new questions. On the other hand, it ensures that the questions on the platform are of a slightly higher quality and don’t overlap too much.

Users can also see statistics for their submissions, such as how their run-time and memory distribution compares to other submissions (Figure 2.2).

Two Sum

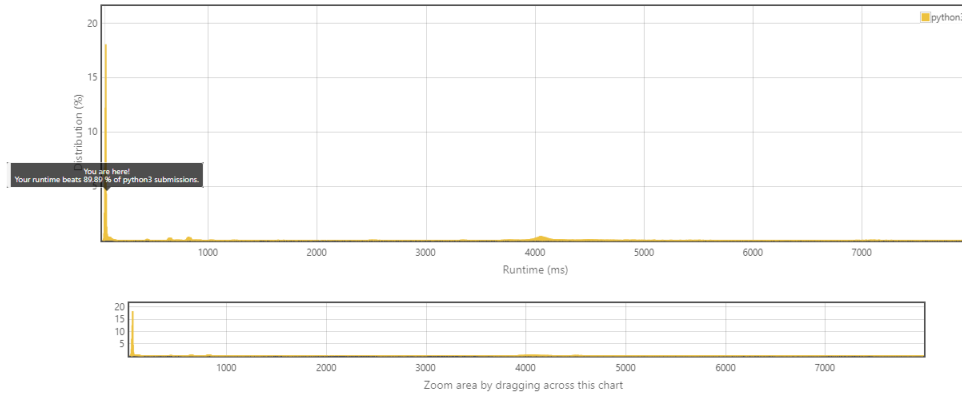
Submission Detail

55 / 55 test cases passed.
Runtime: 56 ms
Memory Usage: 15.4 MB

Status: Accepted

Submitted: 2 months, 2 weeks ago

Accepted Solutions Runtime Distribution



Accepted Solutions Memory Distribution

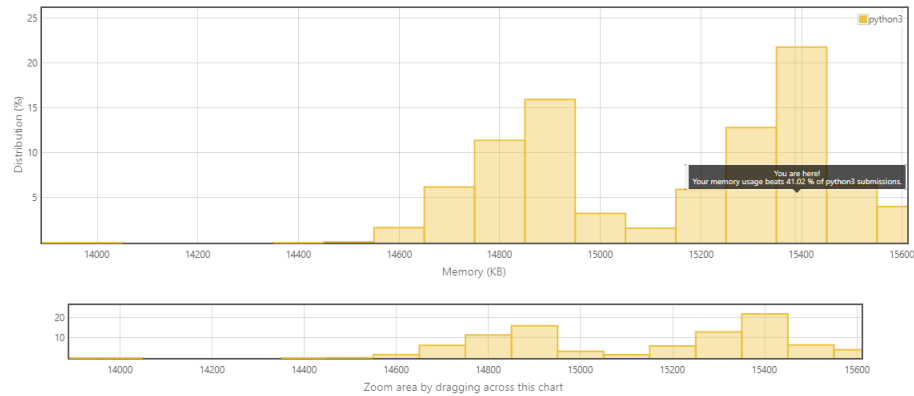


Figure 2.2: Example of submission details to a question.

Their IDE is simple and easy to use with customisation features such as themes and auto-completion, although some also require a ‘Premium’ account.

LeetCode also provides a simple forum for each question where users will typically discuss the question and solutions for various languages.

LeetCode has begun to expand into technical recruiting by providing some features for interviewing. Most relevant to our project is the “Assessment” feature, which allows you to complete some questions in a timed manner and get some basic statistics back.

Drawbacks

LeetCode Premium is required for many features such as solutions, and even many questions are locked behind Premium. It costs up to \$35 per month and unlocks features such as a

faster online judge (so users have to wait less time between submission), autocompletion in the IDE and a debugger. Wait time between submissions is imposed because of the computational resources required to facilitate that feature and is something to consider when scaling this project to more users.

Furthermore, the platform is not focused on school education and doesn't provide any features for managing a classroom.

Summary

Overall, LeetCode is an online judge system with many great features helpful in preparing for interviews and completing coding challenges, even if some of it is locked behind the requirement of a paid 'Premium' account. It is not focused on school education and is more concentrated on interview preparation but still has many valuable features that make it stand out, such as the statistics it provides.

Chapter 3

Specification

This chapter establishes the functional and non-functional requirements for this project, user stories and a specification based on the requirements. It is based on the background research completed in the previous chapter and aims to collect the good ideas that could work well together for this project and add features that existing projects did not implement or could be improved. The requirements consider both the frontend and backend of the system since the frontend will be made to demonstrate the capabilities of the backend platform, which is the primary service that this project aims to develop.

This process began with brainstorming and writing all ideas that came into mind on a whiteboard and then selecting those most appropriate for this project. Some of the ideas not directly used in this specification are still good and will be mentioned later in the report when discussing future work.

3.1 Functional Requirements

Functional requirements are mostly mandatory requirements that capture a use case of the system, often describing a product feature. We will explain most of our functional requirements as user stories, with some functional requirements being described separately. They are provided in the tables below, accompanied by a brief specification detailing how they could be implemented and what they might rely on.

3.1.1 User Stories

User stories are a small unit of work that describes an end goal from the user's perspective. [7]

These are to be used as milestones for the development of this project.

Users

Requirement	Specification
1. As a user, I would like to be able to log in to the web application	The frontend needs a login page, and the backend needs to provide the functionality to support that. This can be done using OAuth. This project could use GitHub OAuth to sign in and store basic data on the application's database. Some token can then be stored on the browser for the login session and cleared when they log out.
2. As a user, I would like to be able to switch between a Student and Teacher mode	A single user should be able to act as both a student who is in classrooms and has assignments, as well as a teacher who can create classrooms, invite students and set assignments and manage them. This is because some people might need to perform both roles with a single account, such as teaching assistants or just for testing purposes.

Students

Requirement	Specification
3. As a student, I would like to see all the programming questions available	The frontend needs a page with a list of all the questions. Some kind of table with information for each question would be helpful, including a button to redirect to the question. If there are a lot of questions, pagination would be needed and methods to sort or filter the table.

4. As a student, I would like to use an online IDE to code my solution to a problem and submit it to see the result	The frontend needs an IDE for the user to code. This could be implemented using CodeMirror [12] or other online code editors. When the code is submitted, it should be judged using the Judge0 API [25], and relevant results should be sent to the frontend and shown to the user. The code should be tested against that problems test cases.
5. As a student, I would like to be able to give feedback to a programming question	The backend should support storing feedback for problems, such as ratings. Users should see the average ratings for a problem and be able to add their own. This data would be stored in the database for each problem.
6. As a student, I would like to see the solutions submitted by other people for a particular problem	On each question, there should be a tab to view the submissions of other users. For example, a leaderboard showing the top 10 submissions with the quickest run-times.
7. As a student, I would like to see a history of the questions I have completed	On a user's profile, there should be a list of all their recent submissions for each question they have completed. Clicking on a submission should show more details about that submission. Like GitHub/LeetCode, we could include a calendar/heatmap of their submission history as an additional motivator.
8. As a student, I would like to see which assignments I have for the classrooms I am in	There should be a tab to view everything related to classrooms, including the classrooms users are in and any assignments they have due. It should be easily accessible and include a list of all classrooms, which users can click to get more details. Users should also have the option to leave a classroom.
9. As a student, I would like to see the grade/results I got for each assignment	Viewing an assignment page in a classroom should include the user's grades for each assignment alongside any additional feedback.

Teachers

Requirement	Specification
10. As a teacher, I would like to create a classroom	There should be a button to create classrooms and enter relevant details for that classroom, such as the name. New classrooms should then be stored in a database with the teacher as the creator.
11. As a teacher, I would like to invite students to join my classroom	On the classroom page, it should be possible to generate and copy an invitation link or send it out by email to a list of emails. Students can then use that link to join the classroom. The backend will need some infrastructure to allow for the creation of these links and possible expiry.
12. As a teacher, I would like to be able to create an assignment and set it for students in a classroom with a due date.	There would need to be a button on the classroom page for which the teacher would like to create an assignment. The teacher can then enter details such as the due date for that assignment and which problems it will include.
13. As a teacher, I would like to see statistics for how students are performing on the assignments	On each assignment, there should be a tab to view statistics. It should include statistics such as completion rate, submissions made, average marks etc. Graphs could also be generated using a frontend library such as Chart.js \cite{chartjs}.
14. As a teacher, I would like to export data for a classroom/assignment in a standard format such as CSV	Teachers should be able to go to any assignment and have a button to download the statistics in a standard format. This could then be used for other purposes, such as importing them into a central data store that the university may use, such as Moodle.

3.1.2 Other Requirements

Requirement	Specification
16. Developers must be able to use documentation and instructions to set up the system in their local environment and deploy it	Code must be adequately documented throughout the development of the application. Instructions for setting up the project and deploying it should be available, ideally in a README file written in Markdown at the root of the project directory
17. Developers must be able to run tests to ensure the application is working correctly	Tests must be written for the frontend and backend and any other manual tests. Unit tests, integration tests and end-to-end tests should be used where appropriate. It should be possible to easily run these tests to ensure that all the core functionalities of the application still work when changes are made. Some of these tests can be based on the user stories written in this chapter. Jest [24] can be used as a testing library as this project will involve a lot of JavaScript.
18. The application must support at least one programming language	This includes the online judge system, problem creator and IDE. They must support at least one language while still being extensible to add more languages in the future. Examples of support include creating a question with the language, writing code in an IDE with syntax highlighting, and running the code through the online judge system. Example languages that could be supported involve Python, JavaScript and Java due to their popularity [32].
19. The application must have authentication and authorisation, particularly for accessing classrooms	As mentioned, with the ability to log in, it must be possible to log into the application securely. There will be pages that will require you to be authorised to view, such as specific classrooms and data.

3.2 Non-functional Requirements

Non-functional requirements describe the system's qualities we are developing and are focused on delivering a better user experience. They are things to keep in mind while designing the application to ensure it is high quality.

1. The application must be usable. Usability is critical as the backend platform should be easy to use and allow for the development of well-designed applications (in this case, a web application). The design must be appealing and be usable in typical desktop environments.
2. The application must be secure. Through the use of authentication and authorisation, user data should be protected, particularly in classrooms. Databases should also be adequately protected. This project has an exceptionally high risk for remote code execution since users can submit code. The application will have user information such as username, organisation identifiers (such as school emails) and their feedback and marks for assignments.
3. The application must be performant. For example, users should not need to wait a long time to access different pages or wait too long to get the results for their submissions.
4. The application must be reliable. It must be available when needed, and the results it gives are accurate
5. The application should be extensible and maintainable. It should be possible to add new features and pages quickly and, importantly, support more programming languages in the future. The way the architecture is set up and testing is done should enable this
6. The application should be easily deployable. This makes the development of the application more straightforward and allows issues to be fixed more quickly.
7. The application should be scalable. If there are many users and more submissions occur, the system could be under a higher load which could reduce the performance. It should be possible to scale the system to deal with increased (or decreased) demand
8. The system should be interoperable. The system's APIs should use standard formats such as JSON or CSV. This is particularly for any data exporting functionality that may be added.

Chapter 4

Design

The design phase is an opportunity to look at how the specification could be implemented, explore technical decisions, and find areas of particular difficulty to plan for. In this project, both frontend and backend design need to be closely considered.

4.1 Technology

The application will follow a client-server model, with a frontend web application for the users to interact with and a server that stores and processes data. The frontend client provides the user interface while the server handles the many APIs used and data storage.

This section will cover the major technologies that will be used to develop the application and justify their use over alternatives (if any exist). Explaining the technology that will be used should make the descriptions of the architecture and implementation clearer.

Area	Technology	Summary justification
Language	TypeScript	Strongly typed language which can be used for both the frontend and backend with excellent support for all the other libraries to be used
Frontend	React with Chakra UI for styling	Most popular frontend framework with a UI component library with great WAI-ARIA support
Backend	Express server with GraphQL API	GraphQL includes excellent developer tools and is self-documenting based on the schema. Great types support

Database	PostgreSQL with Prisma as the ORM	Project lends itself well to a relational database. Prisma is an ORM made with TypeScript in mind and comes with great tools such as Prisma Studio
Code executor	Judge0	Rather than create such a system from scratch, Judge0 can be used, and any missing functionality can be made with wrapper functions
Authentication	Passport.js with GitHub OAuth strategy and JWTs	Rather than add a custom username/password system, I will use OAuth. Passport.js makes implementing that simple, and it is extensible so that new login strategies can be easily added

TypeScript

TypeScript is a strongly typed programming language developed and maintained by Microsoft [29]. It builds upon JavaScript, which currently does not support types or have the sophisticated tooling available for TypeScript. Microsoft's VS Code IDE has robust support for TypeScript, making them a strong combination for development [30]. The choice of TypeScript also helps achieve some of the non-functional requirements set out for this project by making the code more maintainable and easier to write because of things such as autocompletion and type checking.

TypeScript code is then transpiled into JavaScript with the TypeScript compiler. The great thing about this is that programmers can write TypeScript using the latest features. The compiler can transpile the code into any version of JavaScript, including versions that do not support the latest features, by generating alternative code to do the same thing. This can be important for having code running on older browsers or servers.

Of course, the main benefit of TypeScript is still in the name, and that is the support for types. Producing type-safe code reduces the scope of some of the tests we would need to perform. JavaScript does have some very recent proposals to begin support types, although this is not even close to being usable, unlike TypeScript [38].

Popular frameworks for developing frontend web applications support JavaScript/TypeScript, making this language a good choice. I could still write the backend with Python or Java instead; however, using a single language for both the frontend and backend has benefits. Firstly, new contributors to the project will only need to be familiar with a single language to

understand the entire project. Some code can be reused for both ends of the project, such as types. This means both the frontend and backend can agree on the types for certain objects, most notably the data objects we will define in GraphQL. More about this in later sections.

React

React is a free and open-source frontend JavaScript framework created by Meta (formerly Facebook) [37]. It allows user interfaces to be developed using encapsulated components of UI code. They allow developers to break down each part of a web application into its component with its own logic. Any component that may require another component can add it to its body (this is known as called component composition). React uses JSX, a syntax extension to JavaScript that acts as a templating language that allows developers to describe UI elements similar to HTML. React is powerful because it is written in JavaScript and allows developers to easily embed JavaScript into the templates, such as other React components [36]. It also automatically rerenders the UI when changes are made to the state of a component, such as when the user interacts with some elements or new data is fetched from an external API.

Using a frontend framework is vital as they improve the development process significantly and encourage better coding practices. Reusable UI components make the frontend more maintainable and modular. Many of them have large communities providing excellent documentation and tutorials. Deciding on a frontend framework early is important as changing later would involve a major rewriting of the codebase. Other popular frontend frameworks include Vue.js, Angular and Svelte [21]. This project opts to use React because it is the most popular framework with the most support. Many JavaScript modules also provide React versions that integrate nicely with the React ecosystem. This point is incredibly important as it can save a lot of developer time as there is no need to build common UI elements from scratch. Furthermore, a rich ecosystem of libraries and tools is available through npm (Node Package Manager) [33].

Chakra UI

Chakra UI is a component library that provides various components that help in building React applications [41]. It removes the need for creating some simple but common UI elements, and they are also already styled to look appealing, user friendly and consistent. Chakra UI also makes it easy to add custom changes to the UI elements without having to write CSS directly. It also has many other great benefits, such as adhering to WAI-ARIA (Web Accessibility Initiative – Accessible Rich Internet Applications) standards, which means the components are

more accessible to people with disabilities. This library will be used alongside fully custom components to style the web application.

Many other component libraries such as Material UI, Bootstrap or Blueprint are available, but this project opts for Chakra UI due to its popularity, support, customisability, and commitment to WAI-ARIA support in all its components.

CodeMirror/Monaco Editor/Ace

The frontend application needs to include a code editor for users to be able to write code to submit. Basic features such as autocompletion and syntax highlighting are incredibly helpful to have. CodeMirror is a popular choice, designed for writing code and supports various languages and themes and comes with many addons and features such as autocompletion, linter integration and split views. There is also a React component built around CodeMirror, making it much easier to integrate into this project.

There are some alternatives, such as the Monaco Editor by Microsoft, which is used by Visual Studio Code, a popular IDE. Another editor is Ace (Ajax.org Cloud9 Editor). For now, this project aims to use CodeMirror because it has much support for React. However, so does the Monaco Editor, so it may be a good option too because of the familiarity many users may have with it from VSCode. It will be important to experiment with both during development and select the preferred choice.

Express

Express is a fast, unopinionated, minimalist web framework for Node.js. We use TypeScript for both the frontend and backend, we need a framework to run our backend server, and Express is a popular choice. Express is easy to set up and has many utility methods and middleware to make creating APIs easier. It is also very performant, and as with any popular framework, it has excellent documentation and a plethora of tutorials available online.

Express will be used to run the backend server and connect all the backend services, such as authentication, database connections, the online judge system and any other APIs which our backend will utilise.

We are not interested in non-JavaScript alternatives such as Flask, Django or Spring Boot as they would not integrate with the rest of the JavaScript modules that this project intends to use. JavaScript alternatives to Express do exist, such as Koa and Fastify. Koa was developed by some of the original developers of Express, and Fastify was developed with a focus on fast

performance, as the name would suggest. It has been decided to still go with Express because of its ubiquity and support by many other libraries this project would use.

Prisma

Object-relational mappings (ORMs) allow you to have a “virtual object database” used within a programming language. They work by mapping entries in a database to objects in the programming language, making work with databases more manageable and maintainable. Some benefits include that it does not need developers to write any SQL statements and can instead rely on abstract functions to interact with the database. These abstract functions have often been optimised to be more performant compared to writing bespoke SQL queries. If it is necessary to write raw SQL, that option is still available. Since they are an abstract layer above a database, it is also simple to change the underlying database from Postgresql to MySQL or anything else. ORMs also come with various utility functions such as seeding and making migrations.

ORMs have two primary drawbacks. Because of the abstraction, it may not always be clear what the system is doing under the hood, making debugging some things more difficult for complex systems. The other drawback is the initial setup of an ORM and the configuration required.

Prisma is an ORM focused on Node.js and TypeScript. It is easy to set up with excellent documentation and examples, supports JavaScript/TypeScript and Go and supports many different databases such as PostgreSQL, MySQL and MongoDB. It has an extensive type system, ensuring code is safer for production and provides amazing autocompletion. Setting up the database is incredibly simple. A ‘schema.prisma’ file is defined that includes all data models, the database to be used and any additional plugins (Figure 4.1).

```

datasource db {
  url      = env("DATABASE_URL")
  provider = "postgresql"
}

generator client {
  provider = "prisma-client-js"
}

model User {
  id          Int      @id @default(autoincrement())
  createdAt   DateTime @default(now())
  email       String   @unique
  name        String?
  role        Role     @default(USER)
  posts       Post[]
}

model Post {
  id          Int      @id @default(autoincrement())
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
  published   Boolean   @default(false)
  title       String    @db.VarChar(255)
  author      User?     @relation(fields: [authorId], references: [id])
  authorId    Int?
}

enum Role {
  USER
  ADMIN
}

```

Figure 4.1: Prisma schema example for a relational database with User and Post models [35]

The `prisma migrate` command syncs the database with the schema. Prisma also used a database IDE called Prisma Studio, a graphical way of managing the database. It also helps during development as it can visualise the database and edit it (Figure 4.2).

id	githubId	username	organisationId	createdAt	ownedClassrooms	problems	classrooms	submissions
1	11	ali	null	2022-03-25T12:...	2 Classroom	3 Problem	1 UsersOnClassrooms	1 Submission
2	22	bob	bob@school.ac.uk	2022-03-25T12:...	1 Classroom	0 Problem	1 UsersOnClassrooms	1 Submission
3	33	cathy	cathy@school.ac.uk	2022-03-25T12:...	0 Classroom	0 Problem	3 UsersOnClassrooms	2 Submission
4	51179189	zaini	cathy@school.ac.ukd	2022-03-25T12:...	2 Classroom	1 Problem	1 UsersOnClassrooms	5 Submission
5	55	edward	null	2022-03-25T12:...	0 Classroom	0 Problem	0 UsersOnClassrooms	0 Submission

Figure 4.2: Prisma Studio showing the 'Users' table

Before discussing the alternatives for ORMs, it should be explained why this project will use a relational database instead of No-SQL like MongoDB. This is because the structure of the objects envisioned is very relational. For example, a user will have many classrooms, which in turn have many students and assignments, and assignments have many problems and submissions. These relationships lend themselves very well to a relational database approach over No-SQL.

Alternative ORMs for TypeScript have been reviewed. Mongoose is a popular ORM but is built only for MongoDB, but this project will be using a relational database as opposed to No-SQL, so it is not of much use. Sequelize is another popular ORM but it also lacks many of Prisma's great tools, such as Studio. Knex.js is a query builder for SQL databases and provides more fine-tuned control of the queries. However, it also lacks the excellent tooling available with Prisma.

GraphQL

Representational state transfer (REST) has been the standard for designing web APIs for over a decade. It is very robust and scalable; however, there was a demand for a new system with better performance and a richer feature set with more complex services.

GraphQL is an open-source query language developed by Meta. It involves describing the structure of data, called 'type definitions', using the GraphQL schema. Then 'resolvers' are defined, which describe the different functions that can be used on the data. Resolvers come in two types: queries and mutations (Figure 4.4). Queries typically only return data, similar to a GET request in a standard RESTful API. Mutations may change the actual state of the data and return some value, similar to a POST request. These must also be defined in the type definitions (Figure 4.3). These type definitions then automatically produce simple documentation for the input and output types, and with the use of Apollo Studio (Figure 4.5), it can be easy to write queries/mutations.

```
type Specification {
  title: String!
  description: String!
  initialCode: String!
  testCases: [TestCase!]!
  difficulty: Difficulty!
}
type TestCase {
  id: ID!
  stdin: String!
  expectedOutput: String!
  isHidden: Boolean!
}
type Problem {
  id: ID!
  creator: User!
  specification: Specification!
  solved: Boolean
  rating: ProblemRating!
}

type Mutation {
  # User Mutations
  deleteUser(userId: ID!, username: String!): Boolean
  setOrganisationId(organisationId: String!): Boolean
  # End of User Mutations

  # Classroom Mutations
  createClassroom(classroomName: String!, password: String!): Classroom
  joinClassroom(classroomId: ID!, password: String!): Classroom
  deleteClassroom(
    classroomId: ID!
    classroomName: String!
    password: String!
  ): Boolean
  removeStudent(studentId: ID!, classroomId: ID!): Boolean
  # End of Classroom Mutations
}
```

```

type Query {
  # User Queries
  getUser(userId: ID!): User!
  isLoggedIn: String!
  # End of User Queries

  # Classroom Queries
  getTeacherClassrooms: [Classroom!]!
  getLearnerClassrooms: [Classroom!]!
  getClassroom(classroomId: ID!): Classroom!
  # End of Classroom Queries

  # Assignment Queries
  getAssignment(assignmentId: ID!, classroomId: ID!): Assignment!
  getAssignments: [Assignment!]!
  getAssignmentSubmissions(
    assignmentId: ID!
    userId: ID
  ): [AssignmentSubmission!]!
  getAssignmentProblemSubmissions(assignmentId: ID!): [ProblemSubmissions!]!
  getAssignmentSubmissionsAsTeacher(
    assignmentId: ID!
  ): [UserAssignmentSubmission!]!
  getAssignmentExportData(
    assignmentId: ID!
  ): [UserAssignmentSubmissionDataRow!]!
  # End of Assignment Queries

```

Figure 4.3: GraphQL type definitions for types, mutations and queries

```

getClassroom: async (
  _: any,
  { classroomId }: QueryGetClassroomArgs,
  context: any
) => {
  logger.info("GraphQL classrooms/getClassroom");

  // Get a classroom to view. Also used when joining a classroom to get basic information.

  const classroom = await prisma.classroom.findUnique({
    where: {
      id: parseInt(classroomId),
    },
    include: {
      assignments: {
        include: { problems: { include: { problem: true } } },
      },
      creator: true,
      users: { include: { user: true } },
    },
  });

  if (classroom) {
    return {
      ...classroom,
      assignments: classroom.assignments.map((assignment) => ({
        ...assignment,
        problems: assignment.problems.map((x) => x.problem),
      })),
      users: classroom.users.map((e) => e.user),
    };
  }

  throw new ApolloError("This classroom does not exist.");
},

```

Figure 4.4: GraphQL query resolver for the ‘getClassroom’ query

GraphQL gives us more control over the data we get since queries can be defined to only return the data that is relevant to us. It also integrates well with TypeScript and provides good autocompletion. It also has various technical advantages over a REST API, particularly its solution to over and under-fetching. Queries are defined to provide the exact data required and nothing more. A REST API will often structure the endpoints based on the frontend views, which is a good idea as each view can have an associated endpoint for all its data. As the frontend views evolve, significant adjustments to the backend endpoints must be made to ensure the correct data is being fetched. With GraphQL, it is possible to adjust the query being made on the frontend to remove or add the required data without touching the backend.

This is important for the backend platform because we do not want to enforce a structure to the frontend even implicitly, as we want developers to be able to use the backend platform in any way they want.

GraphQL is just a pattern for doing things, so various implementations can be used. In this project, This project opts to use Apollo Server [6] as it works well with Express and has great tooling, including a web-based tool (Apollo Studio Explorer (Figure 4.5)) for writing queries and mutations. It also has many great features such as caching, simple to implement authentication and authorisation through the use of a context for each request. This is the server side of Apollo. Of course, we want to now be able to use this API in the frontend. Apollo has us covered with Apollo Client, a state management library for JavaScript that allows you to fetch and cache data and update our UI. It integrates seamlessly with Apollo Server and modern React, by including support for features such as hooks. As with all Apollo services, it has great tooling and many useful extensions for development in VSCode, TypeScript and Chrome/FireFox [5].

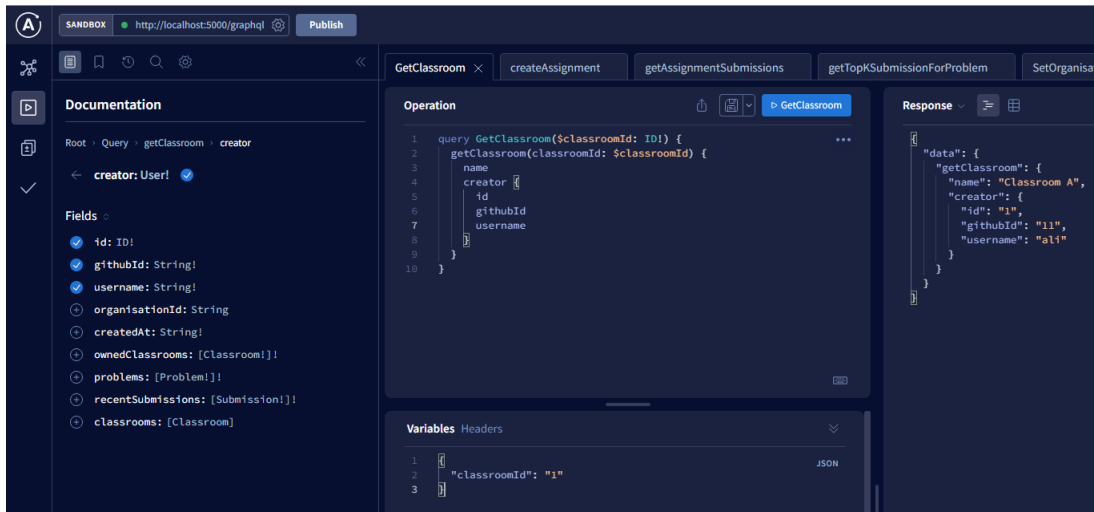


Figure 4.5: Apollo Studio example of the 'getClassroom' query

JSON Web Tokens

JSON Web Tokens (JWTs) are an open standard for securely transmitting information in the form of JSON objects. The data is signed and can be verified and trusted using a public/private key pair [26]. We will use JWTs for a few things, mainly authorising and exchanging some basic information.

For authorisation, it will be used to store the user's ID and other information such as their user object from the database. This token will then be stored in their browser's local storage, which can be accessed from any page in the application using a global React context. This

means that any page can easily access information such as the username or account type. This token is also included in all requests to the backend that requires authorisation, such as GraphQL requests.

JWTs can be easily implemented into a JavaScript project using the ‘jsonwebtoken’ library [9]. The library provides a `sign` function, which takes the data you want to in the token (the payload) and a private key. An expiration for the token can be included to invalidate it after some time, which is a useful additional security feature. It then returns the signed token. The signed token can then be verified using the `verify` function. This takes the token and a public key and returns the payload or an error if the token is not valid. The library also includes methods that can be useful, such as `decode`, which will decode the token even if its data is not valid.

Passport

Passport is simple authentication middleware for Node.js and can be easily added to an Express application. It also has various “strategies” for different platforms such as Facebook, Google, Twitter, and GitHub. I have chosen GitHub as the only strategy to support because it is a platform that many people who code already have, and it is also relevant to some future work that could integrate with GitHub. Although the Passport module allows you to create your strategies to use a custom username/password system, we have opted to rely on GitHub OAuth for now as it removes much overhead of having to support multiple different login types and offloads the security of the accounts to GitHub.

The way it works is that the frontend will redirect the user to a particular link in the backend, which takes a user to GitHub to log in. Once logged in, the user is redirected back to the backend, which finds the user’s account, stores it in a session, and redirects them back to the frontend, now logged in (Figure 4.6).

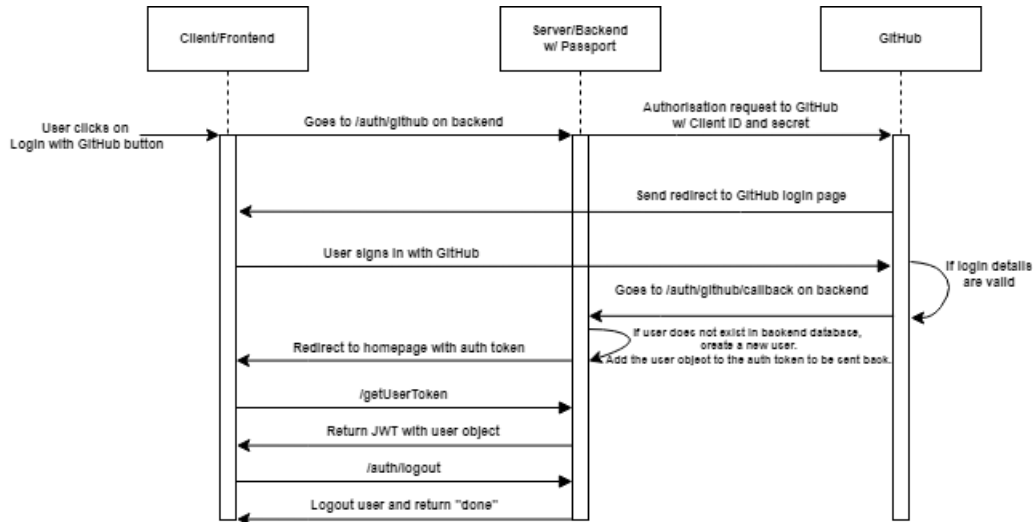


Figure 4.6: Sequence diagram for Passport authentication flow for GitHub OAuth

The alternative to this system is to implement an authentication system from scratch. Although that is a reasonable option, it is something that is not very core to our application and is easier to offload to Passport/GitHub OAuth. Another benefit is that GitHub already has all the relevant information we would ask for during account creation, such as username, email and profile pictures. Overall, it makes development and the user experience smoother.

Passport is also easily extensible as we can add new “strategies” to support new login methods or define our own in the future.

Judge0

Judge0 is an online judge system explored as part of the background literature review. Rather than reiterate all the points, we will explain a little more about why it has been chosen. The system is open-source, extensible, scalable, robust, and easy to use. It uses a simple JSON web API and can be deployed locally using Docker. It is also possible to pay for an external instance. [25].

For this project, it may be necessary to wrap the API with additional functionality to make it more usable with GraphQL and the needs of this project so that it fits into the API ecosystem being built. One crucial example of functionality that Judge0 does not directly support is running multiple test cases for a single piece of source code [1] [2]. This is a common use case when implementing code execution systems since it is often not preferred to test the code against one pair of input and output but a set of inputs and outputs. This project will have to include a function that takes a set of test cases and then does individual tests for each one, and then collates and returns the results of all of them to mimic this functionality as it is

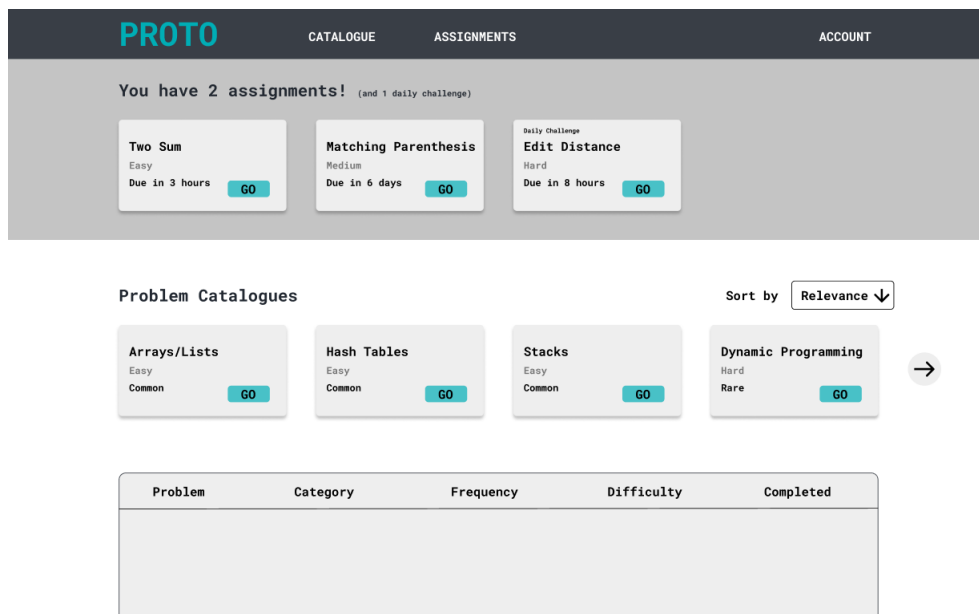


Figure 4.7: Prototype design for a dashboard

not currently available.

Alternatives exist, but none are quite suitable for this project as we want to design a custom IDE and use an API just for judging code and nothing else, which this API does well.

4.2 Frontend Prototyping

This application is called ‘Proto’, derived from standard computer science terms such as ‘Protocol’ or ‘Prototype’. Some simple designs of significant pages for the application were designed using Figma, a web-based graphics editor and prototyping tool [17] (Figure 4.7, Figure 4.8).

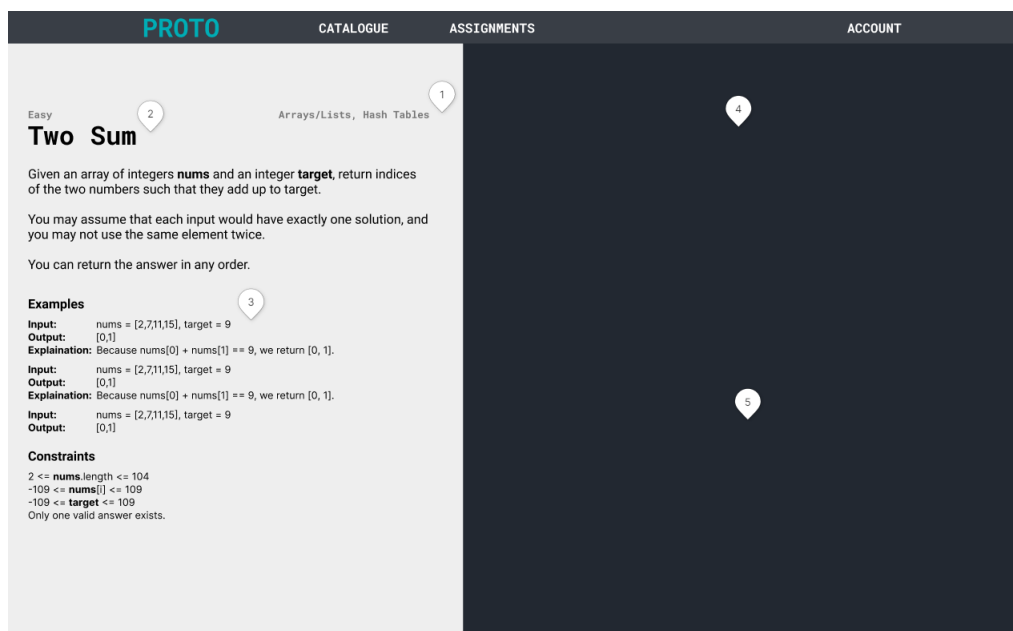


Figure 4.8: Prototype design for a coding problem

Chapter 5

Implementation

This chapter explains the non-trivial components of the system and the approach taken to develop this application. This chapter discusses some of the problems encountered, and the solutions devised and finishes the chapter with a discussion about the testing done for this project.

5.1 Tooling & Methodology

5.1.1 Agile Development

Agile software development is a mindset for developing software focusing on writing software and adapting to change [4]. The Agile practices and concepts that might be utilised in a project can vary. Although no particular framework was used, the most relevant practices utilised in this project were the following:

- User stories: Used to define the specification and inform which features would be worked on next.
- Incremental development: Each ‘version’ of the software is a working product that adds user functionality to the previous version. This was done by using branches in Git, which were only merged into the main branch once they were working and had implemented the features required to fulfil a user story. These increments were also an opportunity to review the code for errors before merging.
- Iterative development: Some functionality was revisited and rewritten, such as changing the code editor from CodeMirror to Monaco. Developing the frontend was highly iterative,

as minor adjustments to UI elements would constantly be made, and the results were previewed before the code was adjusted.

- Kanban: A Kanban board was used to list out tasks and manage which tasks were in-progress/completed/to-do. It was also used to record bugs to be fixed [8].

5.1.2 Developer Operations

Various systems were implemented early on to support the project and enable agile development. This includes logging, various common scripts, a GitHub repository with continuous integration (CI) and continuous deployment (CD), environment variables and utility functions. These additions made it easier to find and test changes during development, find bugs, navigate the code and try different configurations.

Logging

With many different resolvers and endpoints in the code, it is important to have logging to see what exactly is happening when the code is running. Logging helps us to do this by outputting relevant information such as which functions are being called and their inputs and outputs.

Winston is a popular JavaScript logging library that enables the creation of loggers with custom formats, timestamps, colours and more [43]. It also allows the logs to be output to the console and saved to a file for diagnostics. Standard logging practices suggest using different loggers for development, production and testing. The development logger is typically configured to be the most detailed and include information that might not be relevant in a production environment or too large to log in a production environment.

The logger used is dependent on the environment variable `NODE_ENV`. The logger can then be imported into any file in the backend and used. In this project, logs are included at the start and end of any method where it is appropriate. Valuable things to log include the user making the request, parameters, the result and whether the request was successful.

GitHub & CI/CD

GitHub was used as the remote Git repository for this project. The GitHub repository contains all the code for the project as well as a `README` file with instructions for setting it up. GitHub supports Pull Requests (PRs), which are a way to merge a group of commits within a branch into another branch.

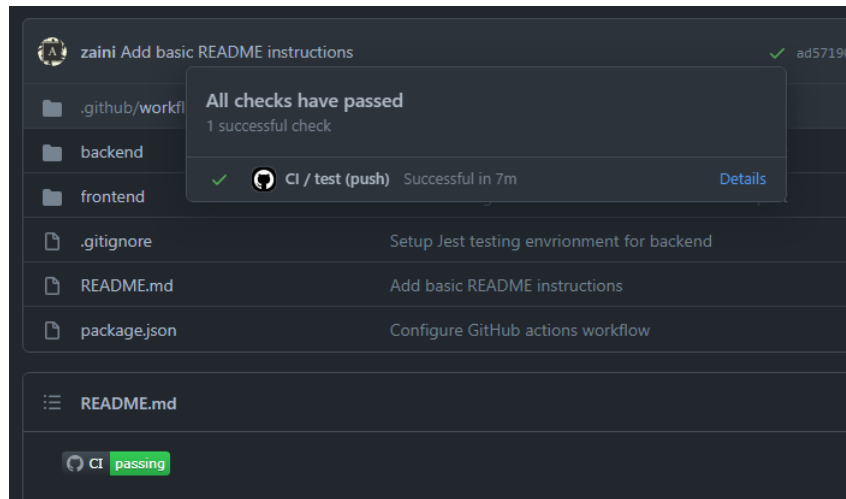


Figure 5.1: GitHub Actions for Proto, showing all checks passed and code is ‘passing’

As described in the previous section about incremental development, branches were used to group code changes related to particular user stories. Once these code changes were completed (when the features were completed, implemented and tested), a PR was created from the new branch into the main branch, and the code changes were reviewed. PRs would include a brief description or title to explain these changes, which could be helpful when looking back through the code. Once the code was reviewed and any final changes were made, it was merged into the main branch.

GitHub Actions allows automated additional actions to be performed once code was committed to the main branch [18]. Actions were set up first to build the project, run the tests, and finally deploy if the tests passed. This helped in rapid iterative development as it would be simple to check that code changes added to the main branch were not breaking the code and causing unintended effects. A continuous integration/deployment (CI/CD) system is handy for large projects with many contributors but is also good to include early on. The GitHub repository also shows some indicators for if everything is running fine (Figure 5.1).

The continuous deployment features were later disabled due to the cost of hosting the system since it required resources beyond what was available from free tiers. Continuous integration features are still enabled.

Other

To simplify setup and deployment, various scripts were added that could be executed alongside instructions in a `README` file in the root of the code repository. Scripts include those to build and start the Docker instance for Judge0, scripts to reset and seed the database, scripts to

install all the required libraries/modules, and scripts to run tests.

Variations of the scripts for development, testing and production environments were also created. This worked by setting different environment variables which were loaded in the code, such as `NODE_ENV`. These scripts can be found in the `package.json` files within the code repository.

There are various utility functions that are used throughout the application, such as those to do with creating and verifying tokens with JWT. These are separated into files for ease of access.

To ensure consistent code styling throughout the project, a linter (ESLint) and code formatter (Prettier) were used [15][34]. Since both the frontend and backend were written in TypeScript, similar styling is used for both.

To help with iterative development, `nodemon` was used to automatically restart the backend server as code changes were saved. This meant that the server did not have to be manually restarted after every change, saving time. React has such hot-reloading already built-in for the frontend.

5.2 Architecture

This section details how the frontend (client) and backend (server) structures and how the different technologies were integrated.

5.2.1 Client Architecture

The client is a React application written in TypeScript. It is structured to have different views, representing the different pages on the website, organised so that each subpage is within the folder for another page, based on how they are related (Figure 5.2).

The root of the application is wrapped with some components to enable the use of some of the libraries that this application will use (Figure 5.3). This includes:

- **ApolloProvider**, which provides a connection to Apollo Client, which is linked to our backend GraphQL instance. This wraps the root component, meaning all components in the application can connect to GraphQL and perform queries/mutations.
- **ChakraProvider**, which enables the use of Chakra UI and its associated features, such as themes and styling for components.

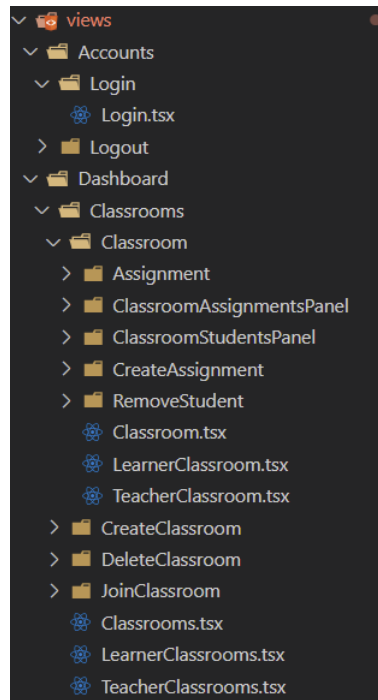


Figure 5.2: File structure for some views

- **AuthProvider**, a custom authentication provider to provide various utility functions and access to values which could be of use throughout the application. This is a React Context Provider, which makes some state and functions available throughout all the components in the application. In particular, this **AuthProvider** will provide the user data (decoded from the JWT token upon login), a logout function (which clears the local storage of the token), the current account type (either student, teacher or empty) and finally a method to set the account type. Account type data is stored in local storage so that when the user returns to the application, they continue as the same account type as before.

Routing

Nested within these three components is the main content of the page. We have a **Navbar** component, which will show up at the top of all pages, and then the **IndexRouter** component, which defines which components will load based on the current route the user is on in the browser (Figure 5.4).

```
const App = () => {
  return (
    <ApolloProvider client={client}>
      <AuthProvider>
        <ChakraProvider theme={theme}>
          <div className="main-content">
            <Navbar />
            <IndexRouter />
          </div>
          <Footer />
        </ChakraProvider>
      </AuthProvider>
    </ApolloProvider>
  );
};

export default App;
```

Figure 5.3: Root App component

```
const IndexRouter = () => {
  const { user } = useContext(AuthContext);

  return (
    <BrowserRouter>
      <Routes>
        {/* public routes */}
        <Route path="/" element={<Home />} />
        <Route path="/accounts/login" element={<Login />} />
        <Route path="/accounts/log-out" element={<Logout />} />

        {/* private routes */}
        {user && (
          <>
            <Route path="/dashboard" element={<DashboardHome />} />
            <Route path="/dashboard/classrooms" element={<Classrooms />} />
            <Route path="/dashboard/classrooms/join/:classroomId" element={<JoinClassroom />} />
            <Route path="/dashboard/classrooms/:classroomId" element={<Classroom />} />
            <Route path="/dashboard/classrooms/:classroomId/assignments" element={<Classroom />} />
            <Route path="/dashboard/classrooms/:classroomId/assignments/:assignmentId" element={<Assignment />} />
            <Route path="/problems/new" element={<NewProblem />} />
            <Route path="/problems/:problemId" element={<Problem />} />
            <Route path="/profile/settings" element={<Settings />} />
            <Route path="/profile/:userId" element={<Profile />} />
          </>
        )}

        <Route path="*" element={<NotFound />} />
      </Routes>
    </BrowserRouter>
  );
};
```

Figure 5.4: `IndexRouter` component with paths and elements defined. ‘Private routes’ are only included if the `user` variable is not empty, so only logged in users can access those routes. A catch-all route is included to render a `NotFound` component. Because this component is a child of the `AuthProvider`, it can use the `AuthContext` to access the `user` variable from the context state.

The screenshot shows a web interface with a table of problems. At the top, there is a search bar with a dropdown menu set to 'all' and a 'Search' button. Below the search bar is a table with the following data:

ID	PROBLEM	DIFFICULTY	TOTAL RATINGS	AVERAGE RATING	SOLVED
1	Addition	easy	3	★★★★☆	true
2	Subtraction	easy	0	Unrated	false
3	Double Word	easy	0	Unrated	false
4	FizzBuzz	medium	0	Unrated	true

Below the table, there is a pagination control showing 'Page 1 of 1' and a dropdown menu to show 5, 10, 15, or 20 items. The 'Show 10' option is currently selected.

Figure 5.5: Example of `CustomTable` component which is reused in multiple parts of the application. Column headers and data are passed into the component and it will be load everything appropriately, with pagination and column sorting. Used for showing problems, members of a classroom, assignments etc.

UI Components

Beyond the views and providers, another folder stores all the other components used in the application. This can include simple UI elements such as custom buttons or modal popups, but also complex components such as a code editor or tables and all their associated subcomponents (Figure 5.5). This is a good way to organise the files as it makes finding and importing components easier. It also groups associated components together rather than having them all dispersed throughout the application. Breaking things down also means they can be reused more easily and encourages better coding practices as we are decoupling all the different components.

Other commonly reused components, such as loading and error components, are used when fetching data from the backend.

‘Create New Problem’ page

Part of the specification requires users to be able to create new problems to be used in the system. With the backend resolver for this functionality defined, a frontend interface had to be created to allow users to provide the problem’s specification for the problem to be created. This was done with a form that included the following:


- Text input field for problem name.
- Drop-down select menu to select problem difficulty.
- Markdown text editor with preview so users know how their markdown would be rendered for others on the problem page. This is better than plain text since it allows for some styling.
- Test case input, reusing existing components. More about this later.

- Multi-select drop-down and search menu for selecting the languages to support the problem.
- Code editor for providing the starter code for problems. For example, defining the function signature or how inputs would be handled.

You can see screenshots of this page in (Figure 5.6) & (Figure 5.7).

This form is fully validated by the backend and tested to ensure users cannot do things such as provide no problem name or no test cases, or no language support. Users get an alert if there is an error explaining the issue, and these messages are also returned from the backend. If the submission is successful, the problem is created, and the user is redirected to the problem's page using the ID returned from the backend when the problem is created.

The boilerplate code in the code editor also comes from the backend, and this means the frontend relies on the backend to provide the default code. This improves maintainability as modifications to the boilerplate code only have to be done in one place.

[Proto](#) [Dashboard](#) [View Classrooms](#) 

Create New Problem

Create Problem

Name

FizzBuzz

Difficulty

EASY

Description

Markdown

Preview

Add two numbers and return the result.

Example 1

Input: 1 2

Output: 3

Example 2

Input: 5 7

Output: 12

Test Cases

Hidden test cases are not shown to users but their code will be tested against them to determine if they have solved a problem.

☒ Hidden test case

Input

3 4

=>

Output

7

Add Test

Test #1

Input: 3 4

Expected Output: 7

Hidden?: false

Remove

Figure 5.6: ‘Create New Problem’ page, form for problem name, difficulty and description with markdown preview. Users can also add test cases for their problem.

Language Support

Select the languages you would like to support and write the boilerplate code you want to provide for people who open this problem.

Example initial code is given for a simple problem with two inputs. Note that all inputs are given in from the standard input as a single line, so you should modify the given code to suit your inputs. Your should then write the expected output to the console as that is how the code will be assessed.

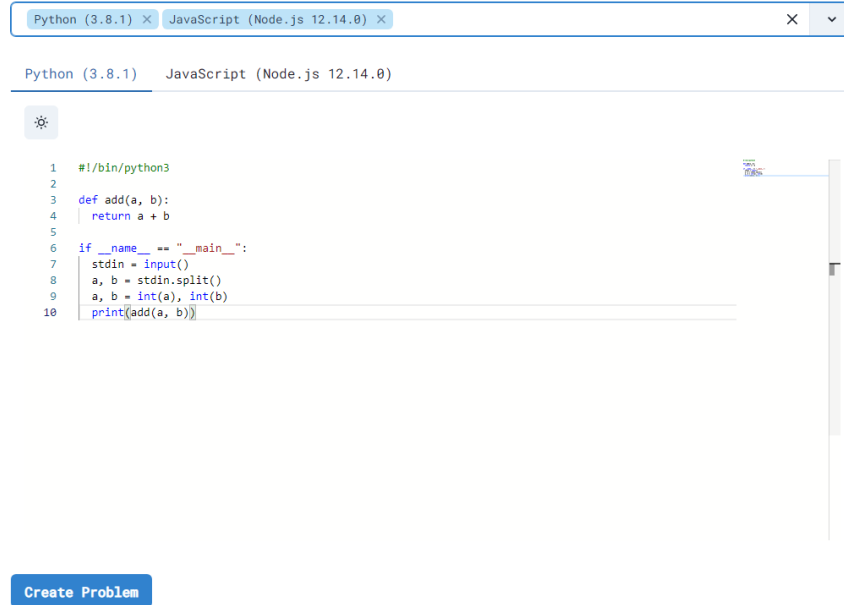


Figure 5.7: ‘Create New Problem’ page, code editor for adding language support.

‘Problem’ page/Code Editor

The problem page is one of the code pages for this project. It is where users will view a problem, write the code, and submit their solutions. This page shows various information about the question, including its description, ratings, test cases and boilerplate code for the languages the problem supports.

Users can view and rate the problem using a star rating system from the problem description tab. They can use the text editor to write their code and configure the editor’s theme and language using the settings on top (Figure 5.8).

There are other tabs the user can switch to, such as the submissions and leaderboard tab. The submissions tab shows the user’s submissions in a paginated table. They can see basic information about each submission, such as when it was made, whether it passed all test cases, its average run-time and memory usage and the language used (Figure 5.9). They also have the option to open that submission and view it in more detail (Figure 5.11). The leaderboard tab is similar, however, it shows submissions from other users as well. This tab contains the best performing submissions based on run-time, displays basic statistics, and allows the user to open the submission to view it in more detail (Figure 5.10).

Prote
Dashboard
View Classrooms

Addition

Description Submissions Leaderboard

Number of Ratings: 4 Average Rating: 3 Your Rating: 4 Created by All

★★★★☆

Add two numbers and return the result.

Example 1

Input: 1 2

Output: 3

Example 2

Input: 5 7

Output: 12

Python (3.8.1)

```

1 # Write your code here
2
3 def add(x, y):
4     return x + y
5
6 if __name__ == '__main__':
7     stdin = input()
8     x, y = stdin.split()
9     x = int(x)
10    y = int(y)
11    print(add(x, y))
12

```

All Test Cases Custom Test Cases

You must pass all these test cases before submitting.
Hidden testcases mean you cannot see the input or expected output.

Run All Tests

Test #1

Input: 30 30

Expected Output: 30

30 30 30 (30 30 30)

Description: Accepted

Your Output: 30

Passed: ✓

Test #2

Hidden Test #3

Hidden Test #4

Submit

Figure 5.8: ‘Problem’ page, with code editor, problem description and some testcases. Users can select a language or change to lightmode from the settings above the code editor.

Addition

Description Submissions Leaderboard

Latest Submission

```
{
  "time": "30/03/2022, 17:27:18",
  "passed": "false",
  "avgTime": "7.75 ms",
  "avgMemory": "3.21 MB",
  "language": "Python (3.8.1)"
}
```

Your Submissions

TIME	PASSED	AVERAGE TIME	AVERAGE MEMORY	LANGUAGE	OPTIONS
26/03/2022, 23:13:17	true	24.00 ms	6.91 MB	JavaScript (Node.js 12.14.0)	View
26/03/2022, 23:13:20	true	26.25 ms	6.96 MB	JavaScript (Node.js 12.14.0)	View
26/03/2022, 23:13:23	true	12.25 ms	4.40 MB	Python (3.8.1)	View
26/03/2022, 23:13:33	true	8.75 ms	3.21 MB	Python (3.8.1)	View
30/03/2022, 17:25:42	false	9.00 ms	3.21 MB	Python (3.8.1)	View
30/03/2022, 17:27:18	false	7.75 ms	3.21 MB	Python (3.8.1)	View

<< <
Page 1 of 1
Show 10
> >>

Figure 5.9: ‘Problem’ page, showing the current users recent submissions and basic statistics. They can click ‘View’ to open a modal to see more details.

Addition

Description Submissions Leaderboard

USER	AVERAGE TIME	AVERAGE MEMORY	LANGUAGE	OPTIONS
cathy	3.00 ms	3.00 MB	Python (3.8.1)	View
zaini	8.75 ms	3.21 MB	Python (3.8.1)	View
zaini	12.25 ms	4.40 MB	Python (3.8.1)	View
zaini	24.00 ms	6.91 MB	JavaScript (Node.js 12.14.0)	View
zaini	26.25 ms	6.96 MB	JavaScript (Node.js 12.14.0)	View

« < Page 1 of 1 Show 10 > »

Figure 5.10: ‘Problem’ page, leaderboard showing the best performing submissions for this problem.

Submission #5 for #1 Addition

[Go to problem](#)

Passed: ✓

🕒 30/03/2022, 10:22:34

📏 31.00 ms

📦 13.41 MB

🗣️ Language: JavaScript (Node.js 12.14.0)

```
1 const add = (a, b) => {
2   return a + b;
3 }
4
5 process.stdin.on("data", buffer => {
6   const ab = (buffer + "").split(" ");
7   const a = parseInt(ab[0]);
8   const b = parseInt(ab[1]);
9   console.log(add(a, b));
10 });
11
```

Test Case Results (4/4)

Test #1 ✓ ^

Input: 10 22

Expected Output: 32

📏 14.34 MB | 📏 32.00 ms

Description: Accepted

Your Output: 32

Passed: ✓

Test #2 ✓ ^

Hidden Test #3 ✓ ^

Hidden Test #4 ✓ ^

[Close](#)

Figure 5.11: Submission modal showing detailed information and statistics about a submission for a problem.

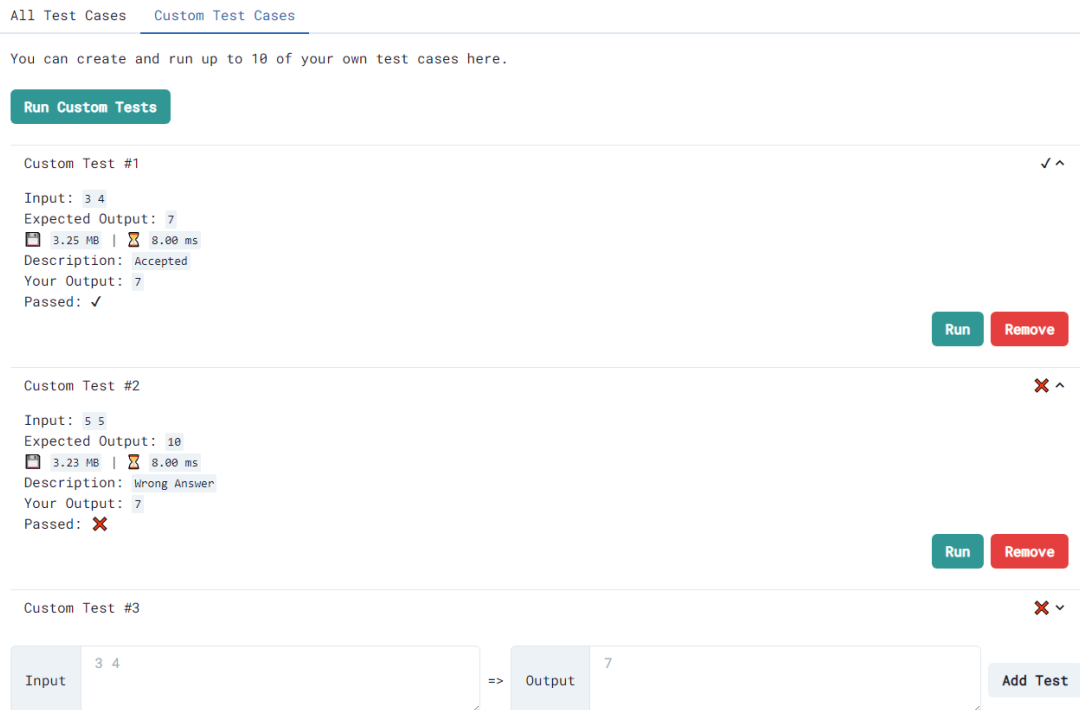


Figure 5.12: ‘Problem’ page, custom test case system. Users can add their own test cases to debug their code. Note the differences in the messages and icons for a passing and failing submission.

A common component used in a few places is the view for test cases and adding custom test cases. It was used in creating a new problem but can also be used when solving a problem, allowing users to add and run their custom test cases. Each test case includes vital information such as what the inputs were, the expected output, the memory usage and run-time, and whether the test case was passed or failed (Figure 5.12). Information about other errors is also shown to help users when debugging (Figure 5.13).

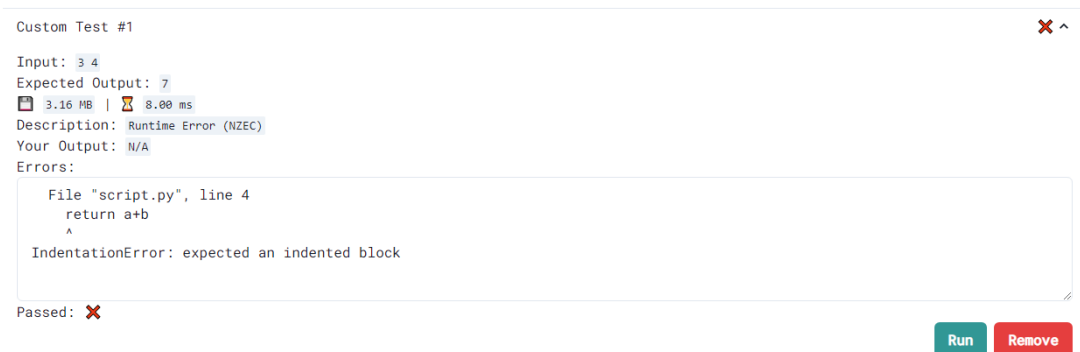


Figure 5.13: Errors for a test case. Useful for debugging.

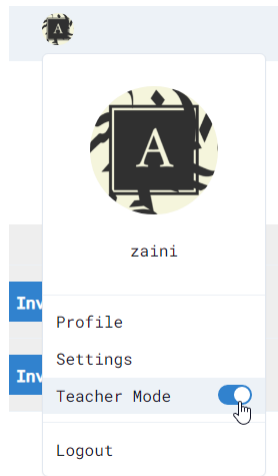


Figure 5.14: Toggle for ‘Teacher Mode’ in profile menu.

Classrooms

Create Classroom

CLASSROOM NAME	# OF STUDENTS	PRIVATE/PUBLIC	CREATED	OPTIONS
#3 Classroom Test	1	public	25/03/2022, 12:09:11	View Set Assignment Copy Invite Link
#4 Classroom Test 2	0	private	25/03/2022, 12:09:11	View Set Assignment Copy Invite Link

Page 1 of 1 Show 10

Figure 5.15: List of classrooms that a teacher owns, with buttons to directly go to set an assignment or copy the invitation link for that classroom. A similar list is visible for students.

Classrooms & Assignments

A significant set of features for this project are related to creating and managing classrooms and assignments. Any account can toggle on ‘Teacher Mode’ from their profile menu (Figure 5.14) to be able to view the classrooms they have created and create new classrooms. Users can then go to the classroom page, which shows a list of their classrooms (in teacher mode, it shows the classrooms the user has created, and as a student, it shows the classrooms the user is in) (Figure 5.15).

Teachers can also create classrooms with a simple form where they must provide a unique classroom name. They can (optionally) include a password that students must enter when attempting to join the classroom (Figure 5.16). Once created, the teacher can share a unique invitation link with their students to join the classroom. In a different context, such as hiring, it might not be called a classroom, but the basic structure would be the same.

When viewing a classrooms page, users can view the assignments they have. Teachers can also view the students in the classroom and set new assignments. Setting new assignments requires an assignment name, problem(s) selected and a due date (Figure 5.18). This is validated

Create Classroom

Name

FC2 Class A

Password

(optional)

Show

Leave the password blank if you want anyone to be able to join. If you include a password, it will be required for people to join.

Submit

Cancel

Figure 5.16: Modal for teachers to create a classroom. A password is optional but if included, students must correctly enter the password to join the classroom through the invitation link.

< All Classrooms

#3 Classroom Test

Created: 25/03/2022, 12:09:11

public

Owner: zaini

Set Assignment

Copy Invite Link

Delete Classroom

Students

Assignments

ID	NAME	SET DATE	DUE DATE	SUBMISSIONS	PROBLEMS	OPTIONS
4	HW1	25/03/2022, 12:09:11	27/03/2022, 13:09:11	0/0	1	<div>View</div> <div>Remove</div>

< <

Page 1 of 1

Show 10

> >

Figure 5.17: Teachers view of list of assignments, similar to that of students.

on the backend, and if any errors occur, an alert is shown to the teacher.

When teachers view an assignment, they can see various graphs showing statistics about the student’s submissions, which can help indicate where students are struggling (Figure 5.19). The teacher can then view all the students, open their submissions, assign marks, and give feedback. All this can then be exported as a CSV, a standard file format that is easy to work with. Once exported, teachers can then do what they want with that data and use it for other pedagogical purposes. One example use could be importing the data into an online learning management system like Moodle. An example of this is shown in (Figure 5.20). To associate each user’s submission with a user on the learning management system, such as Moodle, a unique ID associated with the users on that system must be included. To enable this, users can set their `organisationId` on their profile page, and that is what would be used when importing the data to other platforms. The CSV export can be expanded to include additional information if required.

When students view an assignment, they see which problems they must solve and which submissions they can associate with each problem in the assignment. For example, they might have made three submissions for a problem until they passed all the test cases. On the assign-

×

Create Assignment

Search for the names or IDs of existing problems you would like to add to this assignment.

You also can [create your own problems](#) and then add them to this assignment.

Assignment Name

Homework 3

Problems

#2 Subtraction × #3 Double Word ×

×

▼

Due Date

12/03/2022 21:14

📅

Submit

Cancel

Figure 5.18: Create new assignment modal. This requires an assignment name, problems to be selected and a due date in the future. All this is also validated in the backend.

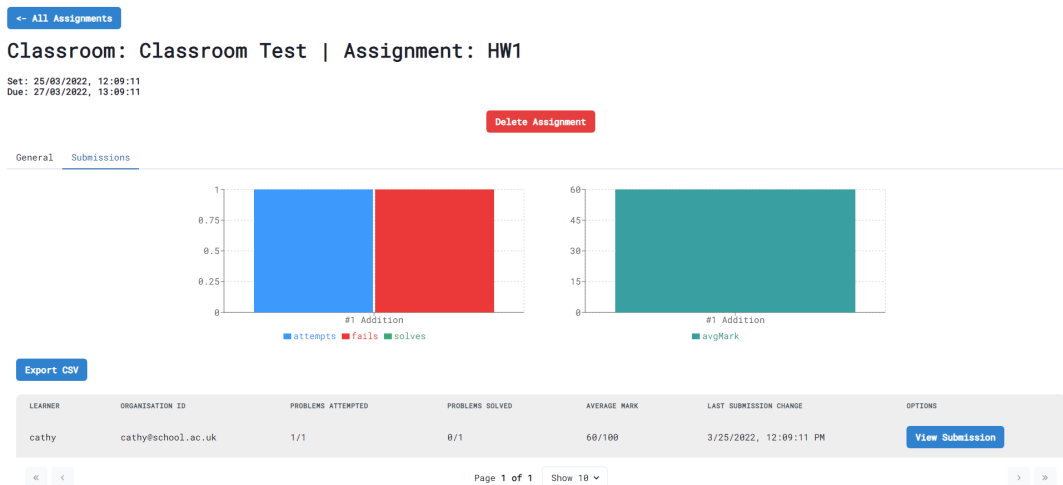


Figure 5.19: Teachers view of all the submissions students have made for an assignment, including graphs for average marks, attempts, fails and solves. Teachers can view each submission in more detail and set the mark and comments. A CSV export button is available.

CSV file

Import CSV

Import preview

organisationId	username	avgMark	solved	lastChange	comments
cathy@school.ac.uk	cathy	60	FALSE	3/25/2022, 12:09:11 PM	Addition - Great work on this

Identify user by

Map from

organisationId

Map to

Email address

Grade item mappings

organisationId

Ignore

username

Ignore

avgMark

Quiz: Test

solved

Ignore

lastChange

Ignore

comments

Feedback for Quiz: Test

Upload grades

Figure 5.20: Example of importing the CSV from Proto into Moodle. Uses the `organisationId` users can set on their profile page as the ID within Moodle.

ment page, they can choose which of those three submissions they want to have marked by the teacher (Figure 5.21).

5.2.2 Server Architecture

The backend server is run using Express. All the additional backend services are linked to the Express server, such as Apollo GraphQL and Prisma (the ORM connecting the server to the PostgreSQL database).

Server Structure

As described in the design section, the backend requires a few endpoints to enable authentication (via Passport and GitHub OAuth) (Figure 4.6). This includes `/auth/github`, `/auth/github/callback` and `/auth/logout`. Express is also configured to pass things such as the session and context to Apollo for use in GraphQL. The session/context is what contains information about the currently logged in user in the form of a JWT.

Code Execution

Judge0 is another service that our backend must handle. I have included a simple script for setting up Judge0 through Docker and included commands in `package.json` to run that script.

<- All Assignments

Classroom: Classroom A | Assignment: HW1

Set: 30/03/2022, 18:21:35

Due: 04/04/2022, 18:21:35

General

Submissions

Current submission for this assignment

(Average Mark: 0.00/100)

PROBLEM	PASSED	MARK	COMMENTS	SUBMISSION	OPTIONS
#1 Addition	true	0.00/100	N/A	Submission #6	Remove as submission for this problem
#2 Subtraction	N/A	0.00/100	N/A	N/A	You must set a submission for this problem
#3 Double Word	N/A	0.00/100	N/A	N/A	You must set a submission for this problem

<< <

Page 1 of 1 Show 10 ▾

> >>

Problem: #1 Addition

Go to problem

ID	PASSED	OPTIONS
5	true	View Set as submission for this problem
6	true	View Submission is used for this problem
7	false	View Set as submission for this problem

<< <

Page 1 of 1 Show 10 ▾

> >>

Problem: #2 Subtraction

Go to problem

You have not made any submissions for this problem

Figure 5.21: Errors for a test case. Useful for debugging.

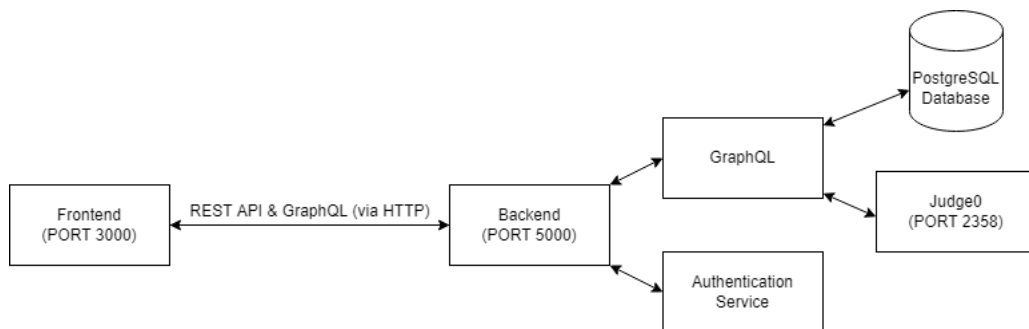


Figure 5.22: Diagram showcasing how the major services interact with one another

The Judge0 server instance is then called by our GraphQL resolvers when code execution must be performed. This way, we can interact with the code judge service the same way as anything else on the backend through the Apollo Client on the frontend, rather than treating the Judge0 API as a disjoint service.

As user stories were being implemented, it was realised that some functionality related to Judge0, namely that it would be called to run multiple test cases, was being repeated. All that differed between these calls was where the test cases were coming from. Either they were new custom test cases written by the user or the test cases associated with a problem. This use case was extracted into its own utility function called `submitTestCases` which would take the source code, language and test cases for a problem and run the test cases for each problem and collate the results (Figure 5.23). This solves the issue highlighted in the previous chapter regarding Judge0 not directly supporting the running of multiple test cases for a single piece of source code. This function is then in both the `submitTests` and `submitProblem` resolvers.

The project supports Python 3.8.1, Python 2.7.9, Java (OpenJDK 13.0.1) and JavaScript (Node.js 12.14.0). It can easily be extended to support more languages however is reliant on the capabilities of the Monaco Editor for the frontend and Judge0 for the backend to also support these languages. Initially, it was also intended to support TypeScript. However, there were significant issues with the Judge0 environment for TypeScript as it was missing libraries and types required for accepting input from the console. Because of this, support for TypeScript was removed. However, it could be quickly reintroduced if these issues are resolved [19]—more about this and related issues in the limitations section of this report.

Database

The database is hosted on PostgreSQL and set up through Prisma, the ORM. A file that defines a ‘seed’ for the database is included and used to initialise some Problems, Users and Classrooms for testing purposes. This seed is repeatedly run between tests so each test can work on a clean version of the database—more about testing in the next chapter.

Schema

The previous review of the frontend setup and various pages should give a clear picture of the functionality that the backend supports. Data relationships on the backend are defined in a schema file for Prisma to generate the PostgreSQL database. A UML diagram is provided alongside a summary of the relationships (Figure 5.24).

```

const submitTestCases = async (user: User, language: number, code: string,
  testCases: TestCase[]): Promise<(TestCaseSubmission & {testCase: TestCase;})[]> => {
  if (!(language in LanguageCodeToName)) {
    throw new ApolloError("This language is not supported.");
  }
  if (testCases.length === 0) {
    throw new ApolloError("Cannot submit tests without test cases.");
  }

  const res = await Promise.all(
    testCases.map(async (testCase) => {
      const options = {
        method: "POST" as Method,
        url: `${JUDGE_API_URL}/submissions`,
        params: { base64_encoded: "false", fields: "*", wait: true },
        headers: { "content-type": "application/json" },
        data: {
          language_id: language,
          source_code: code,
          stdin: testCase.stdin,
          expected_output: testCase.expectedOutput,
        },
      };

      const response = await axios.request(options);

      const testResult = response.data;

      return await prisma.testCaseSubmission.create({
        data: {
          testCaseId: testCase.id,
          userId: user.id,
          description: testResult.status.description,
          compile_output: testResult.compile_output
            ? testResult.compile_output
            : "",
          passed: testResult.status.description === "Accepted",
          stdout: testResult.stdout ? testResult.stdout : "",
          stderr: testResult.stderr ? testResult.stderr : "",
          time: testResult.time * 1000,
          memory: testResult.memory / 1024,
        },
        include: {
          testCase: true,
        },
      });
    })
  );
  return res;
};

```

Figure 5.23: `submitTestCases` function in `resolverUtils.ts`. Performs input validation and then runs the source code against each of the test cases before returning the results.

- Users can own multiple Problems, Classrooms, and Ratings
- Users can be members in multiple classrooms
- Classrooms can have multiple Users and a single creator, as well as many Assignments
- Assignments can have multiple Problems and Assignment Submissions
- Problems have Specifications and Ratings
- Specifications have multiple Testcases
- Assignment Submissions have multiple Submissions for each Problem and User in the Classroom for the Assignment
- Each Submission has multiple Test Case Submissions for each Testcase in the Problem

5.3 Testing

The frontend and backend of this project were tested using various methods and practices. Testing is important as it helps ensure that code works as intended and helps in debugging and fixing issues. Unit tests, integration tests and manually testing were the main types used in development.

Manual testing was used extensively throughout. React has fast-refreshing, meaning as code changes are saved, the server restarts and the changes are immediately visible. The backend server has similar functionality through the use of `nodemon`. This enabled iterative development and allowed manual testing for the user interface and backend code to be performed constantly while coding. TypeScript helped immensely with writing code that could be more easily tested by limiting the types of errors that could occur. It was possible to ensure that specific type-related errors would not occur with type-checking. It is similar to GraphQL, which has type restrictions for the queries/mutations, so it was unnecessary to test invalid types. However, tests for null inputs and boundaries were still performed.

Tests were organised into separate files in the backend, and for the frontend, each component file had a test file associated with it in the same directory. This made writing tests much more manageable.

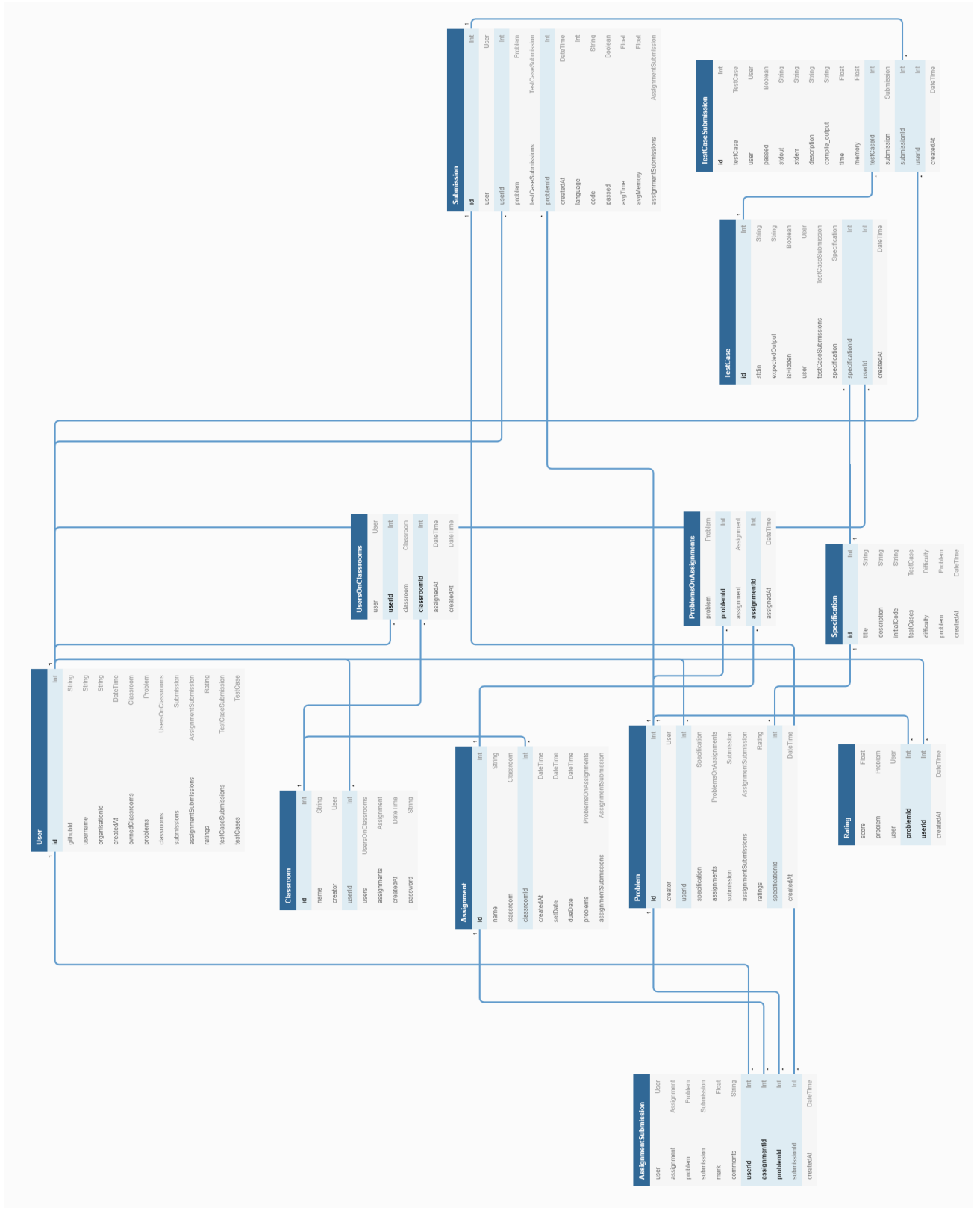


Figure 5.24: UML diagram for database schema

5.3.1 Backend

The backend consists of 118 tests, mainly covering the GraphQL resolvers, which are the primary API endpoints. Important utility functions have unit tests written, such as the method for submitting test cases for execution. The majority of the tests are integration tests written for the resolvers, and these are done by making HTTP requests with GraphQL queries to a live testing server and then checking that the data returned is as expected. These integration tests were typically written to emulate an actual use case based on the requirements for this project.

Testing was done using Jest, which is a popular JavaScript testing library [24]. There were various issues with using Jest, which required many configurations to make work. By default, Jest runs tests in parallel, causing issues as the test server was having multiple tests running on it simultaneously, causing unintended behaviour. This required Jest to be configured to run tests in series.

Prisma does not provide a simple method for deleting or restarting a database programmatically. The solution to this was to write code to execute raw SQL queries to truncate all the tables to delete them and reset the primary keys. Once this was done, the database would be seeded, and testing could continue. Essentially, each test would run, delete the database, seed it, and move on to the next test. Overall this process worked well.

5.3.2 Frontend

Frontend testing involves using React Testing Library with Jest. These tests required the Apollo GraphQL client and router used to redirect to pages to be mocked. Mock data would then be passed, such as user objects, and assertions would be made. For example, the ‘Profile’ page tests would check that the logged-in user’s username would show up on the page in the intended place. The ‘Classrooms’ page test would check that the ‘Create Classroom’ button would only appear when the user was in ‘Teacher mode’.

The frontend benefited greatly from TypeScript, and the automatic fast-refreshing from React meant manual testing was possible throughout development.

Chapter 6

Evaluation

This evaluation will review the success criteria outlined in the specification from Chapter 3. It will cover the functional and non-functional requirements and evaluate how the resulting project compares to the intended project.

Overall, all the objectives for this project were successfully achieved, with many features beyond the initial scope being implemented.

6.1 Functional Requirements

Users

Requirement	Evaluation
1. As a user, I would like to log in to the web application	GitHub OAuth has been implemented using Passport.js, making it easy to extend to support other login methods. Users can log into the web application and have a profile created successfully, and a session is stored on their browser.
2. As a user, I would like to switch between a Student and Teacher mode	Users can toggle 'Teacher Mode' from their profile menu. The account type is stored in the browser's local storage (Figure 5.14).

Students

Requirement	Evaluation
3. As a student, I would like to see all the programming questions available	The 'Dashboard' page contains all the problems on the website, with search functionality and filters by difficulty included.
4. As a student, I would like to be able to use an online IDE to code my solution to a problem and submit it to see the result	Rather than CodeMirror as initially specified, Monaco Editor by Microsoft was used as the IDE. Users can write their code for a problem in various languages and submit it and get results (Figure 5.8).
5. As a student, I would like to be able to give feedback on a programming question	Users can rate a problem using a 5-star system and see the average ratings (Figure 5.8).
6. As a student, I would like to be able to see the solutions submitted by other people for a particular problem	As specified, a leaderboard showing the top submissions for a problem is available (Figure 5.10).
7. As a student, I would like to see a history of the questions I have completed	On a user's profile, there is a heatmap of their submission history, a table of all their submissions and a list of their own created problems.
8. As a student, I would like to see which assignments I have for the classrooms I am in	On the dashboard, users can see all the assignments they have due. Users can also find this information on the classrooms page.
9. As a student, I would like to see the grade/results I got for each assignment	Students can open an assignment and look at their submission to see feedback from the teacher, including comments and marks.

Teachers

Requirement	Evaluation
10. As a teacher, I would like to be able to create a classroom	Teachers can create a classroom with a name and (optional) password. New classrooms are stored in the database with the teacher as the creator.
11. As a teacher, I would like to invite students to join my classroom	Users can copy an invitation link from the classroom page. Students can then use that link to join the classroom. Currently, it is not implemented for the link to expire.
12. As a teacher, I would like to be able to create an assignment and set it for students in a classroom with a due date.	Teachers can set assignments with a due date, name and problems (Figure 5.18).
13. As a teacher, I would like to see statistics for how students are performing on the assignments	On an assignments page, teachers can see the students' submissions along with charts with statistics (Figure 5.19).
14. As a teacher, I would like to be able to export data for a classroom/assignment in a standard format such as CSV	Teachers can download assignment submissions and feedback in a CSV. This can then be used for other purposes, such as importing into a learning management platform like Moodle (Figure 5.20).

6.1.1 Other Requirements

Requirement	Evaluation
16. Developers must be able to use documentation and instructions to set up the system in their local environment and deploy it	A detailed README file is included at the root of the project along with various scripts to make setup, running, and testing simple.
17. Developers must be able to run tests to ensure the application is working correctly	There are over 110 tests for the backend and various tests for the frontend too. Users can easily add more tests, and they are organised clearly and logically and are easy to run, with instructions included.

18. The application must support at least one programming language	The project currently supports Python 3.8.1, Python 2.7.9, Java (OpenJDK 13.0.1) and JavaScript (Node.js 12.14.0). It can easily be extended to support more languages however is reliant on the capabilities of the Monaco Editor for the frontend and Judge0 for the backend also to support these languages.
19. The application must have authentication and authorisation, particularly for things such as accessing classrooms	Users must be logged in with a valid authorisation token to use the application. This is implemented and discussed extensively in previous chapters.

6.2 Non-functional Requirements

6.2.1 Usability

Usability is concerned with the capacity of users to be able to perform their intended actions efficiently. The backend system is usable, and this is assisted by the use of tooling such as Apollo Studio (Figure 4.5). Usability with the frontend is more critical, though, as this is what users will directly interact with, and reflects the backend’s functionality that enables it.

The web application is easy to use, following standard practices and clear UI components. Places of potential confusion, such as the new problem creation page, include additional text and tooltips to help users. Placeholder text is used in many fields throughout the application to show the user examples of what they should be entering.

To avoid users accidentally performing actions that could be detrimental (e.g. deleting their account/classroom), elements require the user to properly read a warning and enter their username/classroom name with correct spelling before confirming the deletion.

Chakra UI has accessibility as a core principle and includes support for keyboard navigation, focus management, colour contrast and other practices of the WAI-ARIA (Web Accessibility Initiative – Accessible Rich Internet Applications) specification.

The application was built for desktop environments, not for mobile use. Usability concerns for mobile were not extensively considered. The beginnings of such support for the application, such as the navbar, which changes based on screen size, can expand further.

6.2.2 Security

Early in the project, it was decided to avoid implementing a custom registration/login system. The application does not store users' account passwords and avoids the associated security hurdles. Instead, OAuth is used, a common alternative for authorisation and authentication.

All the API endpoints include checks to ensure the person making the request is authorised. For example, only the teacher who owns the classroom can set an assignment and modify the marks/comments. This is done by checking the authorisation token passed with the request.

As mentioned in the specification, there were concerns about remote code execution exploits for this application as users are sending code to the backend. However, this concern is not a severe problem because the code execution occurs in an isolated environment in a Docker instance for Judge0, which cannot interfere with the rest of the project. Time and memory limits are imposed to stop users from writing malicious code that loops forever or writes to files.

Secret tokens and values are not hardcoded into the project but instead loaded from environment variables. This means that users viewing the code on GitHub cannot read any secret values.

6.2.3 Performance

The frontend of the application performs well and does not cause issues when using the application. Navigating between pages, logging in and submitting code all perform well.

As the frontend uses React, it can suffer from the issues associated with all client-side rendered applications. After receiving a large network payload from the backend, it relies on the user's hardware to render the site. Because the data on each page is not graphically intensive (mostly just text), it does not have any noticeable effects. Of course, it depends on the hardware that the system is running on as well. However, this system can scale with the hardware, and if something such as the code execution system is under a heavy load, it can be given additional resources.

The application was developed and tested on a desktop PC with an AMD Ryzen 5 5600X CPU, 32GB of RAM and NVIDIA GeForce GTX 1060 3GB GPU. When deploying the project, RAM and CPU power are important to allow the project to be quickly built without issues.

6.2.4 Reliability

Both the frontend and backend perform as expected without any unhandled errors. Messages are shown if something goes wrong, and users can correct any mistakes or just refresh the page to try again.

6.2.5 Extensibility, Maintainability & Scalability

The application is scalable to more users and submissions by increasing hardware resources to the backend. For Judge0, this can be done by changing the configuration for the Docker instance to provide more CPU threads and RAM. If it is not possible to provide more resources for code execution, it would be recommended to throttle user code submissions. This could work by only letting users submit every minute or so.

Extensibility and maintainability have been discussed throughout this report. The use of TypeScript, GraphQL and Prisma means that the code is mostly self-documenting as the types are detailed and easy to view and understand. Folders for both the frontend and backend are organised clearly. Good code should be clear without comments; however, comments are worth including for complex and language-specific operations. Comments for argument and return types were not needed because TypeScript handles that.

It is easy to add new GraphQL endpoints as developers just need to write the new type definitions and associated resolvers. The frontend is also easy to expand by simply creating new components and adding them to the router or whatever existing page needs the components.

One clear area for expansion is additional programming language support. This is discussed further in the limitations section. Adding additional language support is made simple. Developers must define a mapping from a language's name to its internal code for Judge0 (e.g. `Python 3.8.1 -> 71`) and the boilerplate code you want users to have when using the language.

Scripts are included to make the code easier to work with, such as for managing the database, tests and generate types.

6.2.6 Interoperability

Interoperability is about the ability of the system to exchange information in a useful manner. The system uses industry-standard technologies and follows standard best practices. Both the REST and GraphQL APIs allow for data exchange in a standard format that can be easily integrated with other systems, such as JSON. The assignment submission export produces a CSV

(Comma-separated Values) file, which is commonly used by other systems, as demonstrated by importing the file into Moodle (Figure 5.20).

6.3 Limitations

It is essential to highlight the system's current limitations as it shows an awareness of the system's capabilities and what can be improved or added as future work. Here is a list of the main limitations, even if they do not directly limit any of the specification points for this project.

- Language support: The project currently supports four languages and can be extended to support more easily. The frontend is limited in that it can only provide syntax highlighting for languages also supported by the Monaco Editor. Without this support, there would be no syntax highlighting; however, the editor would still work, and it technically does not impede the project from working. The backend uses Judge0 for code execution and is limited by Judge0's capabilities. As noted in previous sections, TypeScript support is not fully available because the Judge0 environment is missing features required to allow TypeScript to read user input. This also means libraries/packages/modules for some languages are also not available, such as `NumPy` for Python. This can be remedied by creating custom environments. This is possible since Judge0 is open source; however, there is not much documentation on how to do this.
- Automated frontend testing is not as extensive as it could be. Currently, most tests simply check that pages load as expected and respond to the current account type (teacher or student) and whether the user is logged in or not. More complex automated tests could be added that check that entire user workflows function correctly. However, this was not done as the frontend was rapidly being iterated and would slow down development. Manual testing was the preferred method for the frontend.
- This project does not have a mechanism to prevent all forms of cheating. The current solution only tackles one part of the problem by having hidden test cases so users cannot brute force a solution, but if they copy code from another source, there is nothing to stop that. This issue is discussed further in the conclusion.

Chapter 7

Legal, Social, Ethical and Professional Issues

This chapter discusses the legal, social, ethical and professional issues within the context of this project. It will also discuss the regulations and principles related to GDPR and the Code of Conduct & Code of Good Practice issued by the British Computer Society.

7.1 Licensing

This project uses various libraries and resources, many of which have been referenced and credited. Many of them are open source, and efforts have been put to follow all licenses.

The code for this project is made available under the MIT license, which means the code can be modified, distributed and commercialised without source code and under different terms but also comes with no liability or warranty. This project is intended to be integrated with other applications, so this license works well.

7.2 Ethics

This platform could be used for hiring people or educational settings. The system must not misrepresent or mislead users. For example, the marks students get should be updated correctly. This is why testing the application is essential, in order to avoid bugs that could cause problems.

7.3 British Computer Society

The BCS outlines four fundamental principles, which this project abides by [10].

1. Public Interest

- This project is open source and does not discriminate against users based on any condition or requirement.
- Efforts have been made to make it accessible, including following common standards and practices and using a frontend UI framework (Chakra UI) committed to accessibility and WAI-ARIA.

2. Professional Competence and Integrity

- The project was developed solely by the author, with credit to all additional resources used. The author has avoided cutting corners to the detriment of the project.
- If the author did not know or understand something, they put in the time to learn it to ensure it was not misused.

3. Duty to Relevant Authority

- The project was conducted with the best interests of King's College London and the author in mind.
- The author takes responsibility for this project and has credited any other tools/frameworks/libraries.
- The development of this project followed all local laws and regulations. The project is released under the MIT license.

4. Duty to the Profession

- The developer has followed common standards and practices throughout the development of this project.
- This project contributes to the profession by developing an open-source project that others can utilise to build or learn from and a detailed report for the entire development process.

7.4 General Data Protection Regulation

This project also abides by UK GDPR principles [23]. The principles most relevant to this project are discussed here.

1. Lawfulness, fairness and transparency

- The application does not do anything unlawful with users' data.
- The way user data is handled is clear and reasonable, and no unexpected or unjustified processing is done.

2. Data minimisation

- The application only expects basic user information when users log in via GitHub, which is all made clear on the GitHub login page.
- The only user data stored on the database is their username and GitHub ID and any other information they give, such as organisational ID. These are all required for the basic functioning of the website.
- Users can choose to delete all their information easily and permanently from the account settings page.
- Overall, the data stored is adequate, relevant and limited to the necessary functioning of the application.

3. Storage limitation

- Users have a right to be forgotten and can choose to delete all their data whenever they want.

4. Integrity and confidentiality (security)

- Security measures have been discussed at length for this project to protect all personal user data.

5. Accountability

- The author is accountable for all data collected on any version of the application they deploy; however, they are not accountable for what other people do with this software, as per the MIT licence.

Chapter 8

Conclusion & Future Work

8.1 Conclusion

The project began with researching automated code assessment systems in various areas, such as education, hiring and competition. This highlighted the lack of a comprehensive framework that could be applied to all of these use cases and that could be implemented into other applications.

This project has successfully developed such a framework and shown how it could be used in an education setting to create programming problems, create classrooms, set assignments, write and submit code and test cases and receive feedback. As well as this backend platform, a comprehensive frontend that showcases all the features has been developed.

The development followed good coding practices and is ready for developers to integrate it into other applications, as demonstrated by this project.

The project completed all the requirements outlined in the specification and achieved the aims outlined in the first chapter. The literature review supported many of the decisions made when designing the platform. Challenges were solved and documented in this report, and extensive backend testing was performed to ensure the application performed as expected. The legal, social, ethical, and professional issues were addressed.

8.2 Future Work

The expectation that this project would be extended in the future was considered from the start, and extendability was an important point of consideration throughout. Although all

the requirements were achieved, there are areas of future work that could further improve this project:

- More language & library support: currently, the backend is reliant on Judge0 and the language environments it has set up. By default, this is lacking in a few areas, with something such as the Python environment lacking NumPy support. It should be possible to create Docker instances for more languages.
- Different ways to check code: Currently, test cases are run against the code users enter when submitting a test case. They define what the expected output is for a given input. This could be error-prone when users create new problems, as they may enter the incorrect outputs for a given input, meaning no one writing the correct code would get the right answers. This could be solved by requiring the person creating the problem to write code that solves the problem. That same code could be used as an example to run all future test cases against.
- Plagiarism checking: This application does not solve all the challenges associated with plagiarism or cheating with submissions. This is important in practically all domains, including education, hiring and competition. The current solution only tackles one part of the problem by having hidden test cases so users cannot brute force a solution, but if they copy code from another source, there is nothing to stop that. There are mechanisms to try and stop this from happening on an application, such as disabling copy-pasting or tracking if the window/tab focus changes. However, these are implementation-specific solutions, so we need a solution that can be integrated into the backend. There are various code plagiarism systems out there that could be used [3] [13]. Once plagiarism is detected, a warning could be included for any assignment submissions. Assessors can use their discretion if they think cheating is possible (as sometimes, many people just have similar answers).
- Performance: Certain test cases for problems and code could be cached to reduce stress on the backend. This is particularly important as the number of users and submissions increases. For example, with a Least-Recently-Used cache, if the user has just submitted their code and tries to submit the code again without any changes, it should simply return the same result as before since nothing has changed about the inputs. Small changes in the code, such as white space, could be ignored. Testing would need to be done to see if this is a valuable mechanism to improve performance.

- Additional question types: Although the platform is focused on programming problems, some technical assignments may want to include questions that do not directly require the user to write code (e.g. multiple-choice questions or simple text-based response questions). Currently, the schema for problems does not prohibit adding new question types, as the specification field for a problem has a JSON type that can be defined with any structure. However, the mechanisms by which problems interact with other models (such as submissions) mean some reworking would need to be done. One way to do this could be to create a new 'super'-model that can take the form of different types of questions, including 'ProblemQuestions', 'MultiChoiceQuestions' and 'TextQuestions'.
- Multi-file code: Judge0 supports multiple files for code execution, so implementing this is mostly a matter of refactoring how code is passed from the frontend to the backend. The frontend would have to allow the user to create new "files" that open new editors to which the user can write code. The code from these editors can then be bundled into an array of File objects which contain the file name and content and are passed on to the backend for execution.

References

- [1] Creating batched submissions with the same source code 254.
<https://github.com/judge0/judge0/issues/254>.
- [2] Judge multiple testcases in a single submission 223.
<https://github.com/judge0/judge0/issues/223>.
- [3] Stanford University Alex Aiken, Alcatel-Lucent Professor. A System for Detecting Software Similarity. <https://theory.stanford.edu/~aiken/moss/>.
- [4] Agile Alliance. What is Agile? — Agile 101 — Agile Alliance.
<https://www.agilealliance.org/agile101/>.
- [5] Apollo. Introduction to Apollo Client. <https://www.apollographql.com/docs/react/>.
- [6] Apollo. Introduction to Apollo Server. <https://www.apollographql.com/docs/apollo-server/>.
- [7] Atlassian. User stories with examples and a template.
<https://www.atlassian.com/agile/project-management/user-stories>.
- [8] Atlassian. What is a Kanban Board? — Atlassian.
<https://www.atlassian.com/agile/kanban/boards>.
- [9] Auth0. jsonwebtoken - npm. <https://www.npmjs.com/package/jsonwebtoken>.
- [10] THE CHARTERED INSTITUTE FOR IT BCS. CODE OF CONDUCT FOR BCS MEMBERS. <https://www.bcs.org/media/2211/bcs-code-of-conduct.pdf>.
- [11] Codecademy. Using Google Classroom As a Replacement for Teacher Resources. <https://help.codecademy.com/hc/en-us/articles/115001386634-Using-Google-Classroom-As-a-Replacement-for-Teacher-Resources>.

- [12] CodeMirror. CodeMirror. <https://codemirror.net/>.
- [13] Codequiry. Code Plagiarism & Similarity Checker - Codequiry. <https://codequiry.com/>.
- [14] Herman Zvonimir Došilović and Igor Mekterović. Robust and Scalable Online Code Execution System. In *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, pages 1627–1632, 2020.
- [15] ESLint. ESLint - Pluggable JavaScript linter. <https://eslint.org/>.
- [16] Sead Fadilpašić. Computer science degrees are more popular than ever. <https://www.itproportal.com/news/computer-science-degrees-are-more-popular-than-ever/>.
- [17] Figma. Figma: the collaborative interface design tool. <https://www.figma.com/>.
- [18] GitHub. Features GitHub Actions. <https://github.com/features/actions>.
- [19] GitHub. Unable to run a program in Typescript 285. <https://github.com/judge0/judge0/issues/285>.
- [20] HackerRank. HackerRank For School. <https://www.hackerrank.com/products/school/>.
- [21] James Hibbard. The 5 Most Popular Front-end Frameworks Compared. <https://www.sitepoint.com/most-popular-frontend-frameworks-compared/>.
- [22] Tien-Chi Huang, Yu Shu, Shu-Hsuan Chang, Yan-Zhang Huang, Sung-Lin Lee, Yong-Ming Huang, and Chien-Hung Liu. Developing a self-regulated oriented online programming teaching and learning system. In *2014 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, pages 115–120, 2014.
- [23] ICO. The principles — ICO. <https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/principles/>.
- [24] Jest. Jest. <https://jestjs.io/>.
- [25] Judge0. Judge0 - Where code happens. <https://judge0.com/>.
- [26] JWT. Introduction to JSON Web Tokens. <https://jwt.io/introduction>.
- [27] Richard Lobb and Jenny Harlow. Coderunner: A Tool for Assessing Computer Programming Skills. *ACM Inroads*, 7(1):47–51, feb 2016.

- [28] Igor Mekterović, Ljiljana Brkić, Boris Milašinović, and Mirta Baranović. Building a Comprehensive Automated Programming Assessment System. *IEEE Access*, 8:81154–81172, 2020.
- [29] Microsoft. TypeScript: JavaScript With Syntax For Types. <https://www.typescriptlang.org/>.
- [30] Microsoft. TypeScript Programming with Visual Studio Code. <https://code.visualstudio.com/docs/languages/typescript>.
- [31] Moodle. Moodle - Open-source learning platform — Moodle.org. <https://moodle.org/>.
- [32] Ellen Murphy, Tom Crick, and James H Davenport. An analysis of introductory programming courses at UK universities. *The Art, Science, and Engineering of Programming*, 1(2), 2017.
- [33] npm. npm. <https://www.npmjs.com/>.
- [34] Prettier. Prettier - Opinionated Code Formatter. <https://prettier.io/>.
- [35] Prisma. Prisma schema (Reference). <https://www.prisma.io/docs/concepts/components/prisma-schema#example>.
- [36] React. Introducing JSX - React. <https://reactjs.org/docs/introducing-jsx.html>.
- [37] React. React - A JavaScript library for building user interfaces. <https://reactjs.org/>.
- [38] Daniel Rosenwasser. A Proposal For Type Syntax in JavaScript. <https://devblogs.microsoft.com/typescript/a-proposal-for-type-syntax-in-javascript/>.
- [39] Sabina Shibalayeva and Pedro Galicia-Almanza. What are Benefits and Pitfalls of Using Technical Selection Tests During the Hiring Process? 2017.
- [40] Xiaohong Su, Jing Qiu, Tiantian Wang, and Lingling Zhao. Optimization and Improvements of a Moodle-Based Online Learning System for C Programming. In *2016 IEEE Frontiers in Education Conference (FIE)*, page 1–8. IEEE Press, 2016.
- [41] Chakra UI. Chakra UI - A simple, modular and accessible component library that gives you the building blocks you need to build your React applications. <https://chakra-ui.com/>.
- [42] Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. A Survey on Online Judge Systems and Their Applications. *ACM Comput. Surv.*, 51(1), jan 2018.

[43] winstonjs. winstonjs/winston: A logger for just about everything.
 <https://github.com/winstonjs/winston>.

Appendix A

User Guide

A.1 Instructions

The following guide can also be found in the README at the root of the source code for this project and on GitHub.

A.2 Requirements

This project requires the following:

- Node
- npm
- yarn
- PostgreSQL
- Docker/Docker Compose

A.3 Setup + Installation

A.3.1 Frontend

Environment

Set the following environment variables for the frontend. Adjust them as needed for your use case.

```
NODE_PATH=./src
```

```
REACT_APP_BACKEND_URL=http://localhost:5000
```

```
REACT_APP_GRAPHQL_URL=http://localhost:5000/graphql
REACT_APP_GITHUB_AUTH_URL=http://localhost:5000/auth/github
REACT_APP_JWT_TOKEN_SECRET=same_as_backend_JWT_TOKEN_SECRET
```

Main

1. In the frontend directory, run `yarn install` to install all the required packages.
2. Run `yarn start` to start the application. It should be accessible at `http://localhost:3000/`

A.3.2 Backend

Judge0

GitHub for Judge0¹

This project uses `judge0-v1.13.0`. It is included in the backend directory.

It requires `Docker`² and `Docker Compose`.³ More information about the deployment procedure can be found here.⁴

Environment

Set the following environment variables for the backend. Adjust them as needed for your use case.

```
DATABASE_URL="postgresql://username:password@localhost:5432/proto"
JWT_TOKEN_SECRET=same_as_frontend_JWT_TOKEN_SECRET
CLIENT_ID_GITHUB=GH_CLIENT_ID
CLIENT_SECRET_GITHUB=GH_SECRET_ID
REDIRECT_URI_GITHUB=http://localhost:3000/accounts/login
FRONTEND_URL=http://localhost:3000
SESSION_SECRET=some_secret_session_password
JUDGE_API_URL=http://localhost:2358
NODE_ENV=development
```

Information about getting a GitHub client ID and secret can be found here.⁵

Set the Homepage URL to `http://localhost:5000/`.

¹[<https://github.com/judge0/judge0>](https://github.com/judge0/judge0)

²[<https://docs.docker.com/>](https://docs.docker.com/)

³[<https://docs.docker.com/compose/>](https://docs.docker.com/compose/)

⁴[<https://github.com/judge0/judge0/blob/master/CHANGELOG.md#deployment-procedure>](https://github.com/judge0/judge0/blob/master/CHANGELOG.md#deployment-procedure)

⁵[<https://docs.github.com/en/rest/guides/basics-of-authentication>](https://docs.github.com/en/rest/guides/basics-of-authentication)

Set the Authorisation callback URL to `http://localhost:5000/auth/github/callback`.
Adjust these according to your deployment.

Main

1. In the backend directory, run `yarn install` to install all the required packages.
2. Run `yarn run judge` to start Judge0.
3. Run `yarn run test:db:setup` followed by `yarn run test:db:seed` to set up the database.
4. Run `yarn run dev` to start the backend. It should be accessible at `http://localhost:5000/`.
You can open Apollo Studio Explorer at `http://localhost:5000/graphql`.

A.4 Testing

To run the backend tests:

1. Go to the backend directory.
2. Start a test instance of the server with `yarn run test:dev`
3. Run the tests with `yarn run test`

To run the frontend tests:

1. Go to the frontend directory and run `yarn test`. Press `a` to run all tests.

A.5 Walkthrough

The frontend interface is intuitive and simple to use and it is encouraged that users simply try different things. The implementation chapter covers many of the uses, but they will be reiterated here.

- Once set up is complete and you open the application in browser, press on 'login' to log into the application with GitHub.
- You can click on 'Dashboard' on the navigation bar at the top to view existing problems and any assignments you have.
- Open any problem to attempt it. You can try out the different settings and tabs. Refer to the figures for chapter 5, Implementation, for example screenshots.

- From the dashboard, you can also create a new problem and follow the instructions on that page.
- On the classrooms page, you can see the classrooms you are in or the classrooms you own. You can toggle 'Teacher Mode' by clicking on your profile picture and pressing on the toggle.
- You can create new classrooms or view classrooms. You can set or complete assignments, view submissions and so on.
- You can open your profile page by clicking on your profile picture and viewing your account. You can also go to your account settings to delete your account or set an organisation ID.

Appendix B

Source Code

All the source code for this project is available on GitHub, as a PDF of all the source code and a ZIP file downloaded from GitHub.

B.1 Declaration

I verify that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary. - Ali Zaini, April 6, 2022.