

A Level Programming Project Report

Junrong Chen

November 13, 2021

Contents

1	Analysis	4
1.1	Problem identification	4
1.2	Stakeholders	4
1.2.1	Computer Science teachers	4
1.2.2	Computer Science students	5
1.3	Why it is suited to a computational solution	5
1.4	Solve by computational methods	5
1.4.1	Thinking abstractly	5
1.4.2	Thinking ahead	6
1.4.3	Thinking procedurally and decomposition	6
1.4.4	Thinking concurrently	7
1.5	Interview	7
1.5.1	Design interview	7
1.5.2	Conduct the interview	8
1.6	Research	11
1.6.1	LeetCode	11
1.6.2	Codeforces	16
1.7	Features	18
1.8	Limitations	19
1.9	Hardware and software requirements	21

1.10	Success criteria	22
2	Design	24
2.1	Decomposition	24
2.1.1	NavigationView	26
2.1.2	HomePage	27
2.1.3	ProblemsPage	28
2.1.4	CodingPage	29
2.1.5	AssignmentsPage	30
2.1.6	PlaygroundPage	32
2.1.7	AccountPage	33
2.1.8	SettingsPage	34
2.1.9	CreateNewProblemPage	35
2.1.10	CreateNewProblemListPage	36
2.1.11	CreateNewAssignmentPage	37
2.1.12	Judger module	37
2.1.13	Database module	37
2.1.14	REST API module	38
2.2	Algorithm design	38
2.2.1	Searchbox searching algorithm	38
2.2.2	Judger RunCode algorithm	39
2.2.3	Judger Judge Problem algorithm	39
2.2.4	Judger Judge Assignment algorithm	39
2.3	Data structure design	39
2.3.1	Class design	39
2.3.2	Settings design	50
2.3.3	Database design	52
2.3.4	Import/Export design	56

2.4	Development testing	56
2.4.1	Milestones	56
2.4.2	Milestone 1: Create the main interface	56
2.4.3	Milestone 2: Handle settings	56
2.4.4	Milestone 3: Implement the Judger	57
2.4.5	Milestone 4: Implement data structures	57
2.4.6	Milestone 5: Create the database	57
2.4.7	Milestone 6: Handle data import/export	57
2.4.8	Milestone 7: Handle API calls	57
2.5	Post-development testing	57
2.5.1	Alpha testing	57
2.5.2	Beta testing	57
2.5.3	Integration testing	57
3	Development	58
4	Evaluation	59

Chapter 1

Analysis

1.1 Problem identification

A Level Computer Science students need to learn many algorithms and data structures during the course. In the final exam, they need to write pseudocode to solve computational questions. Many students find it is hard to achieve a high score on those questions due to the lack of efficient training. The general method used by students to learn and revise for Computer Science is to attempt and self-mark past paper questions. This works well for ordinary questions. However, for the algorithm questions, different students may produce completely different code solutions. This makes their self-marking very unreliable. It is also too much work for the teacher to mark their solutions one by one. So, in the end, students do not know whether they get things right, and teachers do not know how the students perform and how they can help, especially in this lockdown online learning era where no direct contact between teachers and students is possible.

Both the students and the teachers are looking for a more efficient method to learn and practice.

1.2 Stakeholders

There are two types of stakeholders, Computer Science teachers, and Computer Science students.

1.2.1 Computer Science teachers

Computer Science teachers find it is difficult to monitor their students' ability to design and implement algorithms, so they cannot provide efficient help to

their students. This software allows them to create coding questions and send them to the students. After the students hand their solutions back, the software will automatically mark their answers and provide detailed statistical data with simple visualizations. This helps the teachers saving a lot of time and allows them to help the students better.

The stakeholder is Mr Grimwood, who is an experienced A Level Computer Science teacher who teaches a Year 12 CS group and a Year 13 CS group.

1.2.2 Computer Science students

Computer Science students find that they tend to lose marks on the algorithms coding questions, so they want more practice. But unlike ordinary questions, they may take a completely different approach towards the questions comparing to the mark scheme, so they do not know whether they get it correct. Students may also think they have got things right, but actually, they have made some mistakes. The software provides a free practice space that automatically marks their solutions and points out their mistakes in real-time. So the students can learn and revise more efficiently.

The stakeholders are Timofei and PCloud. They are both Year 13 students studying A Level Computer Science.

1.3 Why it is suited to a computational solution

The original problem, ‘understand and mark a student’s answer’ is a very difficult question for a computer to solve. But I transform the question into ‘compare the output of the students’ code with pre-generated test cases’, which makes the problem solvable using a computational method since a computer is good at ‘executing a piece of code’ and ‘comparing two strings’. This approach solves the ‘marking’ question from another angle and makes the question suited to a computational solution.

1.4 Solve by computational methods

1.4.1 Thinking abstractly

In reality, students use pens and paper to write their code solutions. This can be simplified into a code editor, and the students can use their keyboards to type in the code. In this way, no ‘text scanning’ or ‘handwriting recognition’ is needed which makes the design and programming much easier. The code editor will also provide a better user experience. Features such as syntax highlighting cannot exist on paper but are possible in a code editor.

In reality, the students' answer is sent to a teacher to mark it against the mark scheme. The teacher needs to read the code line by line and check whether it is correct. This process is abstracted into a judger that marks the code against pre-generated test cases, which transforms a problem that originally cannot be solved by computational method into one which is very easy to be solved by a computer while saving time and costs. When creating a new question, instead of creating a mark scheme for marking, the teacher needs to provide test cases with the correct input and expected output. The judger will run the students' submissions with the input and check whether their output matches the expected one.

1.4.2 Thinking ahead

For teachers, the software requires them to enter questions and test cases. A question editor containing input boxes is needed for this purpose. For students, the software requires them to enter their code solutions. A code editor is needed for this purpose. A relational database is needed to store all the data. For all users, the software requires input data from the mouse and keyboard to navigate between different windows and menus. Users will also need a monitor for the program to display all the information and outputs.

1.4.3 Thinking procedurally and decomposition

The program can be decomposed into several parts. Each part can be designed and maintained individually. Different components can interact with each other using custom APIs.



1.4.4 Thinking concurrently

When judging the students' solution, many test cases can be executed at the same time to reduce the judging time. The number of parallel judges needs to be set carefully based on the user's hardware. Running too few test cases concurrently may result in a very long judging time while running too many test cases at the same time may use up computing resources and cause issues.

1.5 Interview

1.5.1 Design interview

Interview for teachers

1. Do you find your students tend to lose marks on programming questions in exams?
2. Do you find marking the programming question takes a lot of time and effort?
3. Compare to the knowledge-based Computer System section, do you find it is more difficult to monitor students' skill level on the Algorithm and Programming section?
4. Have you ever heard about some online programming platforms?
5. Have you ever tried some of the online programming platforms?
6. If yes, what do you think about these platforms? Have you ever considered using them for teaching and training?
7. Do you think a similar solution can help improve the efficiency of learning and training?
8. If no, do you think the idea of a software that can mark students' answers on programming questions and provide analysis data can help improve the efficiency of learning and training?
9. Do you have anything else to add?

Question 1 to 3 is a series of proof-of-concept questions, which I expect my stakeholders to answer 'Yes' to all of them. They confirm that the problem I am trying to solve exists and there is a need for such a solution. Question 4 to 5 asks about the teachers' knowledge of existing solutions. Question 6 to 8 ask about their experiences and opinions about these existing solutions, which gives me insights on the problems with existing solutions and how my solution can fit their need better.

Interview for students

1. Do you find the programming questions difficult?
2. Do you find yourself lacking efficient practising in algorithm designing and programming?
3. Have you ever heard about some online programming platforms?
4. Have you ever tried some of the online programming platforms?
5. If yes, what do you think about these platforms?
6. Do you think a similar solution can help you learning and practising?
7. If no, do you think the idea of software that provides coding questions and marks your answer instantly can help you learn and practice better?
8. Do you have anything else to add?

Question 1 and 2 are similar proof-of-concept questions to confirm such a problem exists. The following questions ask about students' knowledge of existing solutions. If they have used an existing product before, I ask whether they think it helps. Otherwise, I ask whether they think it will be useful.

1.5.2 Conduct the interview

Computer Science teacher - Mr Grimwood

1. Do you find your students tend to lose marks on programming questions in exams?
They do. Many of them don't understand the algorithms.
2. Do you find marking the programming questions takes a lot of time and effort?
Yes. Because some students produce partially correct answers, so it takes a lot of time to identify the correct part and award them the corresponding mark. Some students may take completely different approaches which takes a lot of effort to understand and mark them.
3. Do you find it is more difficult to monitor students' skill level on the Algorithm and Programming section and more difficult to provide sufficient help?
Yes.
4. Have you ever heard about some online programming platforms?
I have. Emm... But I forget the names.
5. If yes, have you ever tried some of the online programming platforms?
I have.

6. If yes, what do you think about these platforms?

I think the idea is quite interesting and I find them working quite well.

7. Have you ever considered using them for teaching and practising?

No. Because most of them require a paid subscription, and their content is more likely to be something like 'Learning Python' which is irrelevant to the A Level Computer Science content.

8. Do you think a similar solution can help improve the efficiency of learning and training?

Yes. The students can learn at their own pace and they can keep practising by themselves.

9. Do you have anything else to add?

No.

Mr Grimwood has several valuable points here. He points out that the 'partially correct' answers are the most difficult ones to mark. For my solution, if a student submits a 'partially correct' code answer, then its output will certainly not match the expected output. This means my solution might not be able to tell the difference between a 'partially correct' answer and an 'incorrect' answer. This is a potential limitation I need to watch out for. He also says the price is one of his concerns. My solution will be free and open-source, which will meet his need perfectly. By adding the function to create custom questions and share them with others, users will be able to create and find A Level Computer Science content, or any content easier. It is also a good idea for me to create some A Level Computer Science content that comes with the software to make it easier to use.

Computer Science student - PCloud

1. Do you find the programming questions difficult?

I find some of them quite complex and difficult, especially the graph algorithms such as Dijkstra.

2. Do you find yourself lacking efficient practising in algorithm designing and programming?

Absolutely. Although I code a lot in my spare time, normal projects are quite different from the exam questions. There are not many past papers and exam-style questions for practising, so I usually don't feel confident of those questions.

3. Have you ever heard about some online programming platforms?

Yes. Such as AcWing, LeetCode, and TopCoder.

4. Have you ever tried some of the online programming platforms?

Yes. I am an active user of AcWing.

5. If yes, what do you think about these platforms?

I enjoy the experience. They can provide instant feedback for my submissions. It provides very strong positive feedback when I solve a new question. I find myself learning faster and more efficiently with such platforms.

6. Do you think a similar solution can help you learning and practising?

Absolutely. The existing platforms do not provide A Level related content. So if a software solution can be altered for A Level Computer Science course, that will help a lot.

7. (*) How do you think it should be optimized for A Level CS content?

You can add past exam questions practising. Add a timed practice mode will be helpful.

8. Do you have anything else to add?

No.

PCloud confirms that such a solution will help him learning and practising more efficiently. The instant feedback of whether he gets the question correctly is very important to him. Instead of sending the user's submission to a remote server, my solution should judge the user's answer on their computer. This can avoid the instabilities caused by the remote server's availability and the network connection. He also gives me some good ideas about the content. I can add past exam questions for users to do timed practice, which enables users to practice algorithms and exam techniques at the same time.

Computer Science student - Timofei

1. Do you find the programming questions difficult?

Yes. I generally lose marks because of some careless syntax mistakes I made.

2. Do you find yourself lacking efficient practising in algorithm designing and programming?

Yes. I find I cannot find many materials to practice.

3. Have you ever heard about some online programming platforms?

Codewar. Something like that.

4. Have you ever tried some of the online programming platforms?

Yes.

5. If yes, what do you think about these platforms?

I think they are quite helpful. But I find their marking is too specific, if I get a single character wrong in my output, it gets marked incorrect.

6. Do you think a similar solution can help you learning and practising?

Yes.

7. Do you have anything else to add?

No.

Timofei points out that the marking system in existing products is not very sensible. This may be a potential limitation of my solution as well. It is easy to directly compare the users' output and the expected output. But if they are different, it is difficult to figure out whether that difference is caused by a wrong code solution or just some formatting error. I can partially solve this by allowing the users to pre-test their code against examples before formal submissions, so they can check the output format.

1.6 Research

There are many coding training websites on the market, most of them share a similar idea, so I will investigate two of the most popular ones.

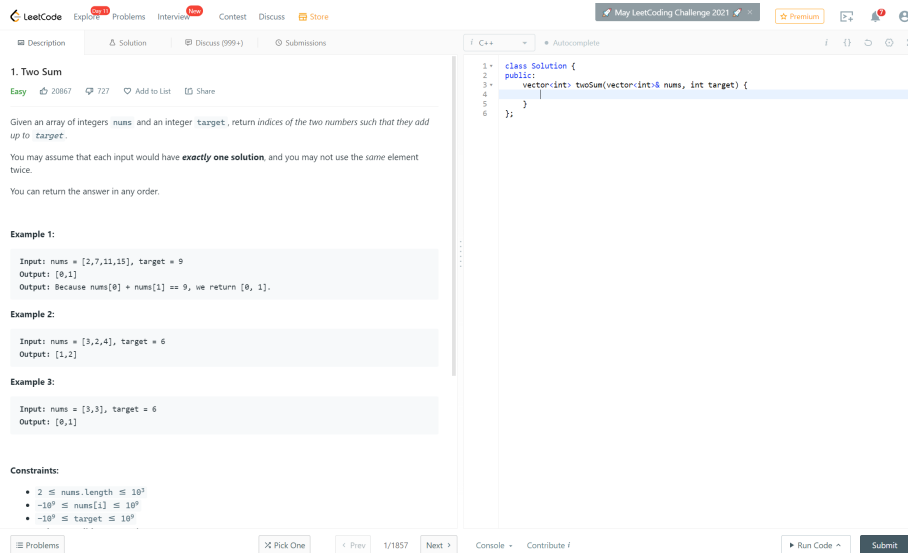
- LeetCode
- Codeforces

1.6.1 LeetCode

LeetCode is a platform for interview coding training, many large companies (Google, Facebook, ...) use it as a part of their interview.

LeetCode provides a database containing more than 1000 coding questions.

Main coding layout



This is LeetCode's main coding area. The user's screen is split into two parts - the question section and the code editor for inputting answers. Users can drag the splitter in the middle to adjust the size of each section.

The question section contains 4 tabs, 'Description' tab displays the content of the question. 'Solution' tab displays the solutions from the community. 'Discuss' tab displays the discussions in the community. 'Submission' tab lists the user's previous submissions. Since I am not adding social functions in my solution, I will ignore the 'Solution' and 'Discuss' tabs. Under the 'Description' tab, LeetCode provides the context of the question, followed by 3 examples, and constraints for this question. The examples allow the user to run and check their solution before formal submission for marking, this can help them avoid silly mistakes. My solution should also provide similar examples for each question. Under the 'Submission' tab, LeetCode records every history submission, so the user can revise old questions more efficiently. My solution should provide a similar function as well.

The screenshot displays the LeetCode web application. On the left, the 'Two Sum' problem is shown with its success status, runtime (8 ms), and memory usage (9 MB). Below this is a table of submission history.

Time Submitted	Status	Runtime	Memory	Language
05/11/2021 22:34	Accepted	8 ms	9 MB	cpp
02/06/2020 23:30	Wrong Answer	N/A	N/A	cpp
01/31/2020 12:16	Wrong Answer	N/A	N/A	cpp
01/31/2020 12:16	Wrong Answer	N/A	N/A	cpp
01/31/2020 12:15	Wrong Answer	N/A	N/A	cpp
01/31/2020 12:12	Accepted	360 ms	9.2 MB	cpp
01/31/2020 12:12	Wrong Answer	N/A	N/A	cpp

The code editor on the right shows a C++ solution for the 'Two Sum' problem. The code uses a hash map to find two numbers that add up to a target. The test case runner at the bottom shows the input [2, 7, 11, 15] and the expected output [0, 1].

The code editor provides line number and syntax highlighting functions. User can change their programming language with a drop-down box. LeetCode supports all mainstream programming languages. My solution should be able to support multiple programming languages as well, which allows students with different backgrounds to use it easily.

On the button, the user can 'Run Code' to test their code against the examples before submission, and then click the 'Submit' button to submit their solution formally.

The split view design is clean and handy. The user can see the question and write their solution on the same page without switching between different windows. The design of examples and the 'Run Code' button is useful as well. I can refer to LeetCode's coding layout when designing my solution's interface.

Question database

Array940

String466

Hash Table332

Dynamic Programming331

Math319

Depth-First Search230

Sorti

Expand

All Topics

Algorithms

Database

Shell

Concurrency

Lists

Difficulty

Status

Tags

Search questions

Pick One

Status	Title	Solution	Acceptance	Difficulty	Frequency
	1189. Maximum Number of Balloons		62.7%	Easy	
	1. Two Sum		47.6%	Easy	
	2. Add Two Numbers		36.8%	Medium	
	3. Longest Substring Without Repeating Chara...		32.2%	Medium	
	4. Median of Two Sorted Arrays		32.7%	Hard	
	5. Longest Palindromic Substring		31.2%	Medium	
	6. ZigZag Conversion		39.6%	Medium	
	7. Reverse Integer		26.1%	Easy	
	8. String to Integer (atoi)		16.0%	Medium	
	9. Palindrome Number		51.3%	Easy	
	10. Regular Expression Matching		27.9%	Hard	
	11. Container With Most Water		53.1%	Medium	
	12. Integer to Roman		58.1%	Medium	
	13. Roman to Integer		57.6%	Easy	
	14. Longest Common Prefix		37.7%	Easy	

Every question in LeetCode has many different attributes (Lists, Difficulty, Status, Tags, Title, Acceptance), so it is very easy for a user to find a suitable question to practice. My solution can similarly organize the question database and provide a corresponding query interface for a better user experience. The ‘Pick One’ button on the top right is a very handy feature as well. Users can simply click that button to start working on a quick random question. The idea of ‘a list of questions’ is great. Users can organize a series of questions to practice and share.

Pricing

The screenshot displays the LeetCode pricing page. At the top, there are two subscription cards: a 'Monthly' card with an orange header and a 'Yearly' card with a black header. The Monthly card shows a price of \$35/month, with a note that it is down from \$39/month. The Yearly card shows a price of \$159/yr, with a note that it is the most popular plan, previously sold for \$299 and now only \$13/month, saving over 60% compared to the monthly plan. Below the cards, there is a list of premium features, each with an icon and a brief description:

- Video Solutions** (NEW): Unlock elaborate premium video solutions like [this](#). Each video includes a detailed conceptual overview and code walkthrough that will efficiently guide you through the problem.
- Access to Premium Content**: Gain exclusive access to our latest and ever-growing collection of premium content, such as questions, Explore cards, and premium solutions, where detailed explanations are written by our team of algorithm and data structure experts.
- Autocomplete**: Not interested in memorization? With premium access, you receive intelligent code completion inside the LeetCode code editor based on language and an analysis of your source code.
- Debugger**: Tired of `System.out.println(val)`? Set breakpoints and debug your code interactively line by line right inside our code editor.
- Lightning Judge**: Tired of waiting? Premium users get priority judging using an exclusive queue, resulting in a 3X shorter wait time, up to 10X during peak hours.
- Sort Questions by Prevalence**: Find out which questions turn up most frequently in interviews so that you know where to focus your personal studying. Invaluable data collected from thousands of samples.
- Interview Simulations**: Mock assessments provide you with a way to test your abilities in a timed setting, just like a coding challenge or on-site interview. You choose the company and we will select an appropriate question from our constantly growing database.
- Unlimited Playgrounds**: Premium users can create an unlimited number of Playgrounds, up from 10! You also get the ability to organize your Playgrounds in folders.

The basic functions of LeetCode are free to use for all users and it charges a fee for premium subscriptions. The premium subscription provides a larger question database, better code editor, faster judger, and more.

Analysis

LeetCode is a fully web-based solution, which means it works on any device. However, it also means you will not be able to use it without a stable Internet connection. I decide to make my solution a desktop application since most students practice coding with a computer. It also save me a lot of cost from running and maintaining a server. LeetCode runs a large community for users to discuss questions with each other. I am not adding such a function to my solution. Teachers and students can use existing platforms they have been familiar with, it is unnecessary for me to develop a new platform and for the users to migrate from mature solutions. LeetCode has an easy-to-use graphical interface, which is important so new users can get their hands on very easily.

LeetCode does not support custom questions or any functions for educators. It is mainly designed for self-learners. My solution is designed for school use, so

it must support functions like custom questions, custom assignments, statistics data visualizations. LeetCode charges a subscription fee for essential functions. My solution will be free and open-source so everyone can benefit from it.

1.6.2 Codeforces

Codeforces is a competitive coding platform, it is mainly used by people to the held coding competition.

Main question layout

A. Fox And Snake

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Fox Ciel starts to learn programming. The first task is drawing a fox! However, that turns out to be too hard for a beginner, so she decides to draw a snake instead.

A snake is a pattern on a n by m table. Denote c -th cell of r -th row as (r, c) . The tail of the snake is located at $(1, 1)$, then it's body extends to $(1, m)$, then goes down 2 rows to $(3, m)$, then goes left to $(3, 1)$ and so on.

Your task is to draw this snake for Fox Ciel: the empty cells should be represented as dot characters (\cdot) and the snake cells should be filled with number signs $(*)$.

Consider sample tests in order to understand the snake pattern.

Input
The only line contains two integers: n and m ($3 \leq n, m \leq 50$).
 n is an odd number.

Output
Output n lines. Each line should contain a string consisting of m characters. Do not output spaces.

Examples

input

```
3 3
```

output

```
***
..*
***
```

input

```
3 4
```

output

```
****
...#
****
```

input

```
5 3
```

output

```
***
..#
***
#..
***
```

input

```
9 9
```

output

```
*****
.....#
*****
.....#
*****
.....#
*****
.....#
*****
```

Codeforces Round #290 (Div. 2)

Finished

Practice

★

Virtual participation

Virtual contest is a way to take part in past contest, as close as possible to participation on time. It is supported only ICPC mode for virtual contests. If you've seen these problems, a virtual contest is not for you - solve these problems in the archive. If you just want to solve some problem from a contest, a virtual contest is not for you - solve this problem in the archive. Never use someone else's code, read the tutorials or communicate with other person during a virtual contest.

[Start virtual contest](#)

Practice

You are registered for practice. You can solve problems unofficially. Results can be found in the contest status and in the bottom of standings.

Clone Contest to Mashup

You can clone this contest to a mashup.

[Clone Contest](#)

Submit?

Language: GNU G++17 7.3.0

Choose file: Choose File No file chosen

Be careful: there is 50 points penalty for submission which fails the pretests or resubmission (except failure on the first test, denial of judgement or similar verdicts). "Passed pretests" submission verdict doesn't guarantee that the solution is absolutely correct and it will pass system tests.

[Submit](#)

Last submissions

Submission	Time	Verdict
66675479	Dec/12/2019 14:07	Accepted
66675379	Dec/12/2019 14:05	Wrong answer on test 1
66675290	Dec/12/2019 14:03	Wrong answer on test 3

Problem tags

Implementation 800

No tag edit access

The questions and the examples take up nearly all the spaces on the question page. There is no online editor or online runtime environment provided. Users are expected to write and test their code in their IDEs and only submit the solution for judging. Custom IDEs may be more powerful than a built-in one. My solution will provide an editor, it is much more convenient to use. Even if a user decides to use his environment, he can paste his code into the editor for submission. It sets the time and memory limits for users' submissions, if a piece of code takes too long to run, or takes up too much memory while running, it will be terminated and marked wrong.

Submission

PROBLEMS SUBMIT CODE MY SUBMISSIONS STATUS HACKS ROOM STANDINGS CUSTOM INVOCATION

General

#	Author	Problem	Lang	Verdict	Time	Memory	Sent	Judged		
66675479	Practice: PCloud_	510A - 15	GNU C++17	Accepted	31 ms	8 KB	2019-12-12 14:07:49	2019-12-12 14:07:49	★	Compare

→ Source

Copy

```
#include<iostream>
using namespace std;
int main()
{
    int m = 0;
    int n = 0;
    cin >> n >> m;
    for(int i = 0; i < n; i++)
    {
        if(i%2==0)
        {
            for(int j = 0; j < m; j++)
            {
                cout<<"#";
            }
            cout<<endl;
        }
        else if(i%4==1)
        {
            for(int j = 0; j < m-1; j++)
            {
                cout<<".";
            }
            cout<<"#<endl;
        }
        else if(i%4==3)
        {
            cout<<"#";
            for(int j = 0; j < m-1; j++)
            {
                cout<<".";
            }
            cout<<endl;
        }
    }
}
```

When the user submits the code, the code enters a queue waiting for judging, then the user can look up their result. Users can check their source code, performance stats, and more importantly, when they have not passed all test cases, they can see what they have got wrong. The judgment protocol provides detailed information about each test case, so users can debug easily.

—Judgement Protocol

Test: #1, time: 15 ms., memory: 0 KB, exit code: 0, checker exit code: 0, verdict: OK

Input

aaabb

Output

6

Answer

6

Checker Log

ok 1 number(s): "6"

Test: #2, time: 15 ms., memory: 4 KB, exit code: 0, checker exit code: 0, verdict: OK

Input

usac0

Output

1

Answer

1

Checker Log

ok 1 number(s): "1"

Test: #3, time: 15 ms., memory: 0 KB, exit code: 0, checker exit code: 0, verdict: OK

Input

l0l

Output

2

Answer

2

Checker Log

ok 1 number(s): "2"

Test: #4, time: 15 ms., memory: 0 KB, exit code: 0, checker exit code: 1, verdict: WRONG_ANSWER

Input

qdpinbncrfwxpdbfgozvocenjructoadewegtvtbvfmrpgyeaxgddrwnlqnygmhmhrhaizpyxvgaFlrsvzhzhrouvprixkfza

Output

-763363328

Answer

37

Checker Log

wrong answer 1st numbers differ - expected: '37', found: '-763363328'

Analysis

Codeforces is optimized for coding competition, so it has a lightweight and complex interface for better performance. It is completely free to use. Users can create their questions but it is very complicated to do so. My solution needs to enable users without experience to create questions easily. There is no function for education - there is no way for a teacher to 'create a class' and monitor his students. Codeforces is an online platform, so it also works across all devices and requires an Internet connection. Users have to use their IDEs to write and debug their code. Codeforces sends all submissions to a central 'judging queue' for marking. My solution will mark all submissions locally, which makes judging a lot faster and save me from running a server. By limiting time and space allowance, Codeforces effectively prevents malicious code from running.

1.7 Features

A useful homepage interface with shortcuts to different functions in the software and other resources outside the software. This allows the user to get into practising faster and makes the software easy to use. Details about the design of the homepage will be discussed in the Design chapter.

A problem database with a graphical user interface. The GUI will have a search box and several drop-down menus for the user to input information to search for a problem. This provides the user with a simple way to find the problems they want, and it also ensures the users can manage and backup the data easily.

An interface for users to create new problems and share them with others. This interface will have multiple text fields for the user to input the descriptions to the problem, the expected input/output data. Then the problem is saved to the database and can be exported to a JSON file allowing the user to share it with others. Users can also create a 'list' of problems and export the entire list into a JSON file and share it with others. This enables teachers to create custom problems and share them with the students. It is a core feature that differentiates my solution from the existing ones.

An interface for teachers to create assignments. An assignment is a 'list' of problems with some extra data, such as the due date and total mark. It can be exported to a JSON file and share with students. The student's submission will be exported into a JSON file as well and can be sent back to the teacher. The solution will also integrate with the assignment function in Microsoft Teams for Education, which makes it even easier to do. The students' submissions will be automatically marked by the software and detailed data will be provided to the teacher. A simple data analysis interface will be provided to the teacher so they can have a brief look at the result. The teacher can also export the data into a CSV file so they can import it to their school system or analysis it with professional software. This automates the entire process from creating assignments, distribute assignments, collecting assignments, and marking

assignments. Teachers will have more time analysis the student's performance and provide corresponding help timely. It is a core feature that differentiates my solution from the existing ones.

An interface displaying the problem and the code editor. The solution provides a 'Run Code' button for the student to pre-run their code before submission, a 'Submit' button for the student to submit their code. This allows the students to read the question and write their code solution without switch between different windows. The 'Run Code' function also makes it easier to debug their code.

A playground with a code editor and runtime environment. This allows the software to be used in class teaching as well, the students can experiment with different algorithms and programming languages in the playground easily.

A settings page contains all the setting options for the software. Users can adjust settings such as their preferred programming language, syntax highlighting settings, colour themes, and so on. This allows the users to customize the software to fit their needs and allows users with different backgrounds to use it without issues.

1.8 Limitations

The software will be written in C# instead of web-based which means extra software needs to be downloaded by the user. I plan to use .NET 5 runtime and WinUI 3 library for my solution, so only the Windows 10 1809 or newer Windows operating systems will be supported. This should not cause many compatibility issues since most school computers are running the required version of the operating system. Downloading an extra software is inconvenient and may violate the IT security policy of some schools.

The judger can only accept code submission in limited programming languages and the user may require to configure their runtime environment. Creating a compiler for 'OCR Pseudocode Programming Language' is too complex for this project. I will attempt to allow the user to add their preferred programming language and write documentation for them to make the process easier.

Unlike LeetCode, there are no Discussion pages for users to discuss questions because it is a desktop program instead of a web one. But this is not a big problem, students and teachers should use an existing product such as Microsoft Teams which has very good support in sharing code snippets. It is unnecessary to rebuild the wheel.

Distribute the questions and assignments is still something inconvenient. Currently, distributing questions and assignments requires the teacher to first export the questions and assignments, then send them to the students through email or file-sharing platforms. When the students finish working, they need to send their results back through email or other apps. I have attempted to integrate the file-sharing function with the existing platform - the Microsoft

Teams Assignment function. But unfortunately, the Graph API required for this operation is still in beta version, which means it can only be tested in the development environment and cannot be used in production. So for now, the users still have to use this inconvenient way to share questions and submissions. But in the future, the integration with some existing platforms may improve the experience. (Update: the Microsoft Teams API is out of beta, now it is possible to integrate with it)

There are no good ways to maintain and distribute a large question database. Computer Science teachers are required to maintain a database for their students. But this is difficult work. Creating good test cases is much time consuming than writing a mark scheme, it is very likely for a wrong solution to pass the judging if the test cases are not good enough. It relies on the teacher who creates the questions to consider everything clearly to minimize its impact.

The judge can only simply compare the students' output with the expected output if there is a format error such as trailing space and extra newline in their output, which will not be considered as a mistake in a real exam, will be marked as a wrong answer by the judge. So students may need to spend extra time debugging their output format. It cannot judge "partially correct" answers as well. It does not care which line did the student get correct or wrong, if the final output doesn't match, the submission will be marked wrong.

1.9 Hardware and software requirements

Hardware and software requirements	Justification
Standard mouse, keyboard, and monitor.	Standard I/O devices are required for the user to interact with the software. Users need a mouse to navigate around different menus and pages, they need to use a keyboard to input their code solutions and use a monitor to get the output from the software.
Operating system: Windows 10 (1809 or later), Windows 11.	The software is designed with the WinUI 3 library and .Net 5 runtimes, which require such an operating system to run.
x86 64-bit CPU (Intel / AMD architecture) with 2 or more cores and 1 GHz or higher clock speed.	A modern CPU is required for the software. 1 core will be used to run the main program and at least 1 spare core is required for the judger to judge the submitted code. A clock speed higher than 1 GHz is required to ensure the software is running smoothly.
1GB free memory or more.	Around 512MB RAM is required to run the software, and another 512MB RAM is required for the judger to judge the submissions.
256MB free disk space or more.	256MB free disk space is required to store and run the program itself, the user may need extra disk space to store extra cache data and the database.
A modern dedicated or integrated graphics card.	The software has very little graphical demand, if the user's graphics card can run their operation system, it should be able to handle software as well.

1.10 Success criteria

Criteria	Justification
Users can use different links, menus, and buttons to navigate around the software easily.	This ensures the program is easy to use and allows the user to find the function they want to use quickly.
Users can use different drop-down menus and the search box to find a problem from the problem database.	This allows users to search for questions easily in the database.
Users can add new questions to the database.	This allows teachers to create new algorithm problems.
Correctly validate the new questions before adding them to the database.	Make sure correct data is input and prevent SQL injection.
Users can create lists of questions.	This allows teachers to organize problems better by creating lists to manage them.
Users can create assignments.	This allows teachers to create new assignments for their students.
Users can export/import questions, lists, and assignments from/to their problem database.	This allows the users to share questions and data with others easily.
Users can work on a problem and their submissions can be automatically marked by the judger.	This allows the users to practice and get feedback on the software.
Users can create submissions for assignments and export them into a file.	This allows the students to complete and hand in assignments easily.
Correctly access and interact with the Microsoft Teams API	Allowing teachers and students to manage assignments through Microsoft Teams.
The software can mark the assignments automatically.	This automates the marking process and reduces teachers' work.
The software can perform simple data analysis to the assignments data.	This provides teachers with an overview of student's performance on their assignments and allowing them to help their students better.
Users can export the assignment data to CSV files.	This allows the teachers to use advanced data analysis tools and import the data into their school system.
Users can use the playground to test any code.	This allows the users to experiment with new algorithms and programming languages and allows the software to be used in class teaching.

Users can customize the software.	This allows users to work in their favourite environment and makes the solution suitable to users with different backgrounds.
Split the core functions and class into a core library.	Allowing easier maintenance.
Using sensible variable names and add comments to each function.	Makes the code easy to understand and make maintenance easier.
Create CI/CD pipelines to build and deploy the application automatically.	It allows the software to be tested and deployed automatically so users always receive the latest features and security updates.
Unit tests with coverage higher than 90%.	It makes sure all code is well tested.

All the success criteria will first be tested through the unit tests created during the development. Then they will be tested and improved by me during the Alpha Testing stage. Finally, they will be tested by the stakeholders in the Beta Testing stage, and the evaluation will be based on their user experience and opinions.

Chapter 2

Design

2.1 Decomposition



2.1.1 NavigationView



The Navigation View provides a global menu for the user to navigate between different pages in the software. There are six tabs in the navigation view, “Home”, “Problems”, “Assignments”, “Playground”, “Account”, “Settings”. By clicking on different tab, the main frame will display the corresponding page. The current selected tab will be highlighted.

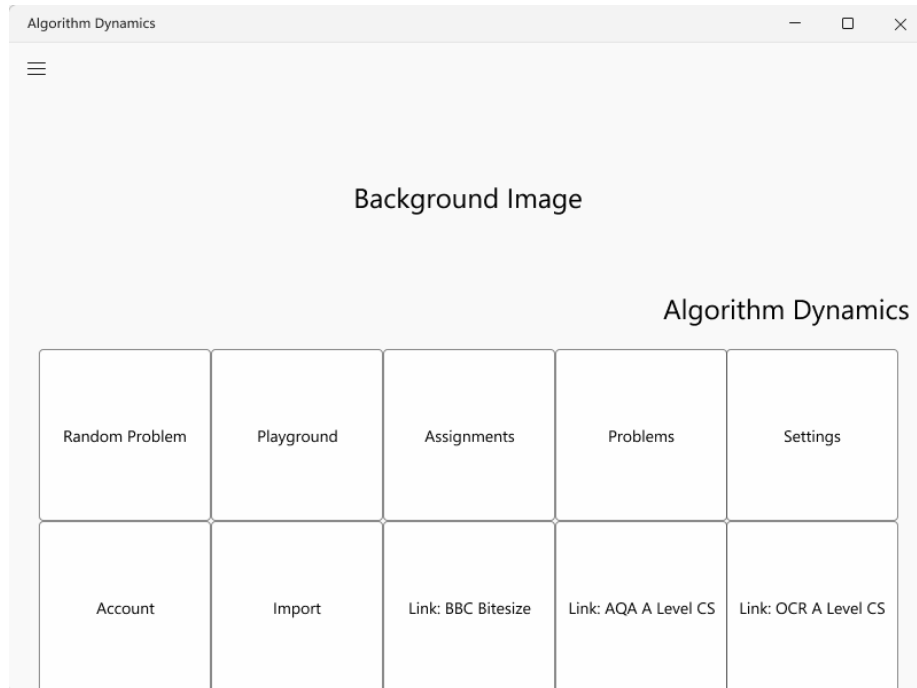
Usability feature

The entire NavigationView is a usability feature. Users can always look it up by clicking the top left button. They will be able to know where they are and navigate to other pages by one click, which makes the program easier to use.

Validation

There are only buttons in the NavigationView for the user to click, so only valid actions can be taken, no further data validation is required.

2.1.2 HomePage



On the top of the HomePage, there will be a beautiful background image under the 'Algorithm Dynamics' title. This makes the interface looks pretty. Let users have a good mood every time they open this software.

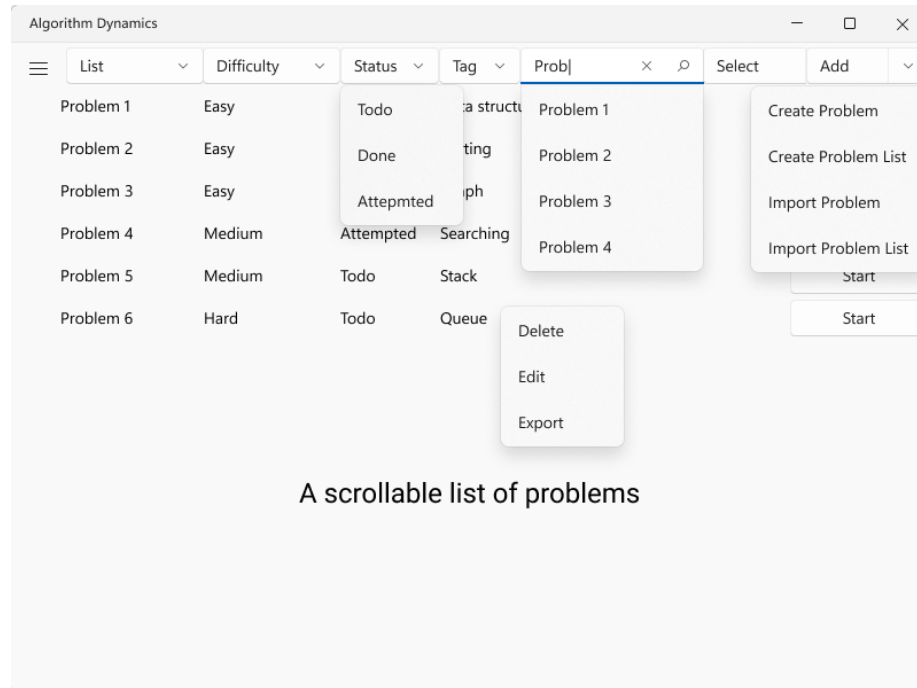
Usability feature

The bottom half of the HomePage will be filled with buttons link to different functions of the software. The 'Random' button starts a random problem for the user. The 'Import' button calls the system file explorer for the user to import problems, problem lists, or assignments. The 'Playground', 'Problems', 'Assignments' and 'Account' button links the user to the corresponding page. The three buttons at the end link to three useful websites, when users click the button, the default web browser will be called and direct to these websites, which makes it easy for the user to look up speifications and revise content. All the useful functions of the software are grouped on the home page, which makes them easily accessible and makes the software easy to use.

Validation

There are only buttons in the HomePage for the user to click, so only valid actions can be taken, no further data validation is required.

2.1.3 ProblemsPage



The ProblemsPage displays problems in the database. The user can search, create, edit, delete, import, export or start working on one or multiple problems on this page.

Usability feature

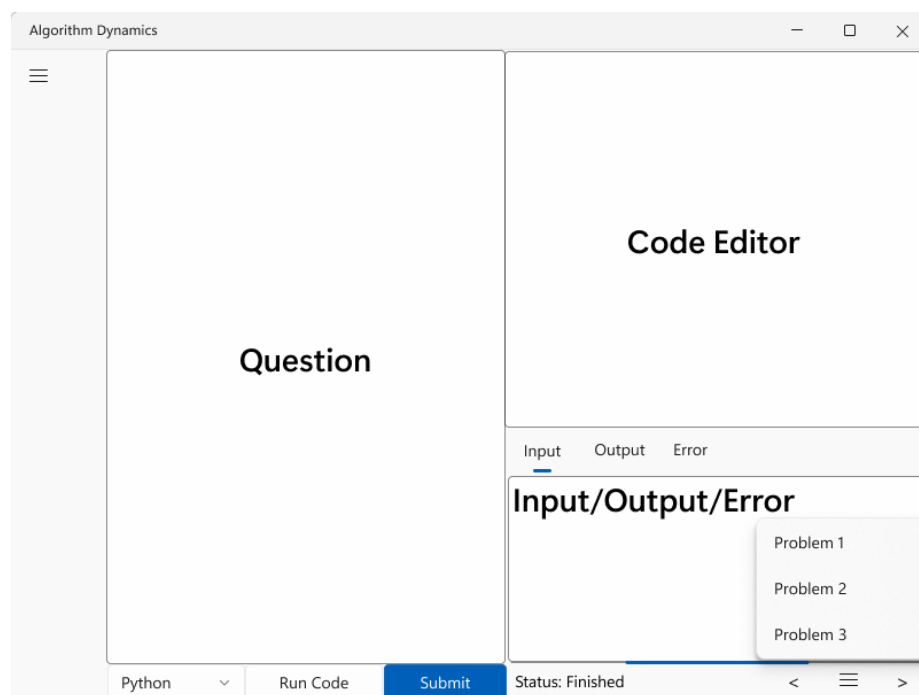
The user can apply selection condition by either select different fields in the dropdown boxes or directly type into the search box. A scrollable list of problems that match the conditions will be shown bellow, with detailed information. The user can start working on a problem by clicking the start button on the right. They can also right click the problem to call a context menu that includes more actions for them to delete, edit or export the problem. By clicking the select button on the top, the user can select multiple problems at once and applie the same action to them at once. By clicking the Add button on the top right, a context menu will be displayed, allowing the user to create or import new problems or problem lists. When the user is typing into the search box, a flyout will display matching results to save typing.

Validation

Most components on this page are still buttons, user can only click them and no invalid data can be input. The search box is where the user can input some

text only, first, a flyout will be displayed to promote the user to click the button instead of inputting data themselves. Next, a length check will be applied, The user can only enter a maximum of 32 characters so they cannot crash the search box and the searching algorithm. Instead of passing the search keywords directly to the database, a custom searching algorithm will be used to search and sanitise the search keyword, which prevent SQL injection and provide a better searching experience.

2.1.4 CodingPage



The CodingPage is where the user works on a programming problem.

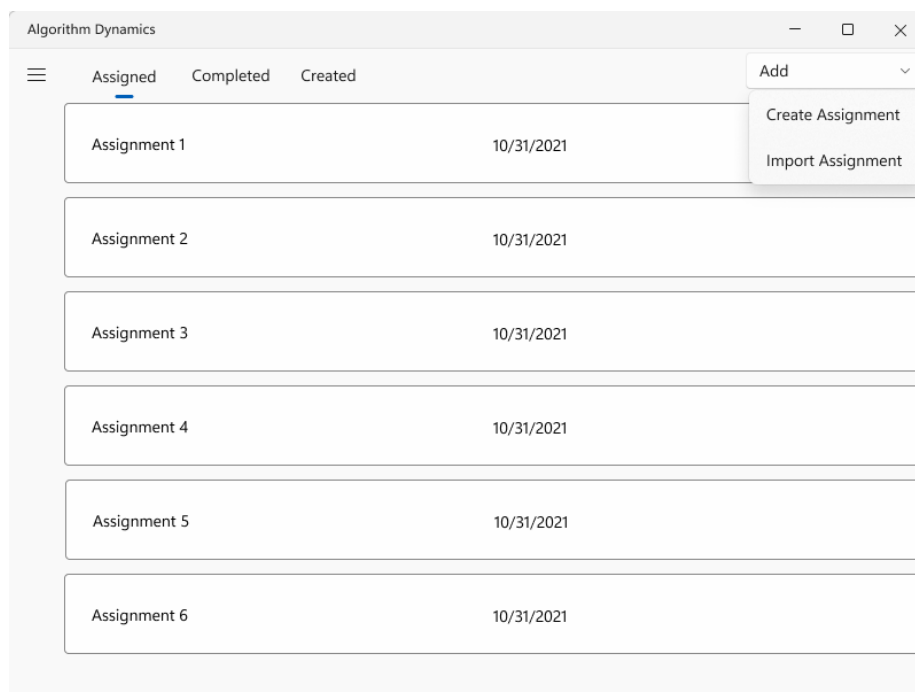
Usability feature

The question is displayed on the left and a code editor will be displayed on the top right. The input, output and error messages will be displayed on button right, the user can switch between them by clicking the corresponding tab. On the button, the user can select programming language use a dropdown menu, run code by clicking the run code button and submit their code for judging by clicking the submit button. The status field shows the status of the judge and 3 navigation buttons on the button right to make it easy navigating between different problems. There is a progress bar between the input output error section and the navigation buttons, it displays the judging progress.

Validaton

Again, most of the components on the page are either readonly (such as the question section) or buttons. The code editor is the only place for the user to input text, and the text inside will be validate by the compiler or ther intrepreter of the selected programming language. However, the input, output and error panel requires further validation. To prevent the user from printing out a huge amount of data which might results in poor performance, the input output error panel will perform a length check and only display the first 2048 characters of the output.

2.1.5 AssignmentsPage



The AssignmentPage is where the user interact with their assignments. For student users, they can work on their assignments, for teacher users, they can create and mark assignments.

Algorithm Dynamics

Assignment 1

10/31/2021

Submit

Assignment Descriptions

Problem 1

Problem 2

Problem 3

Problem 4

Problem 5

Algorithm Dynamics

Assignment 1

10/31/2021

Import Submissions

Edit

Export

Assignment Descriptions

Student 1 Submission

95/100

Student 2 Submission

Not marked

Student 3 Submission

Not marked

Problem 1

Problem 2

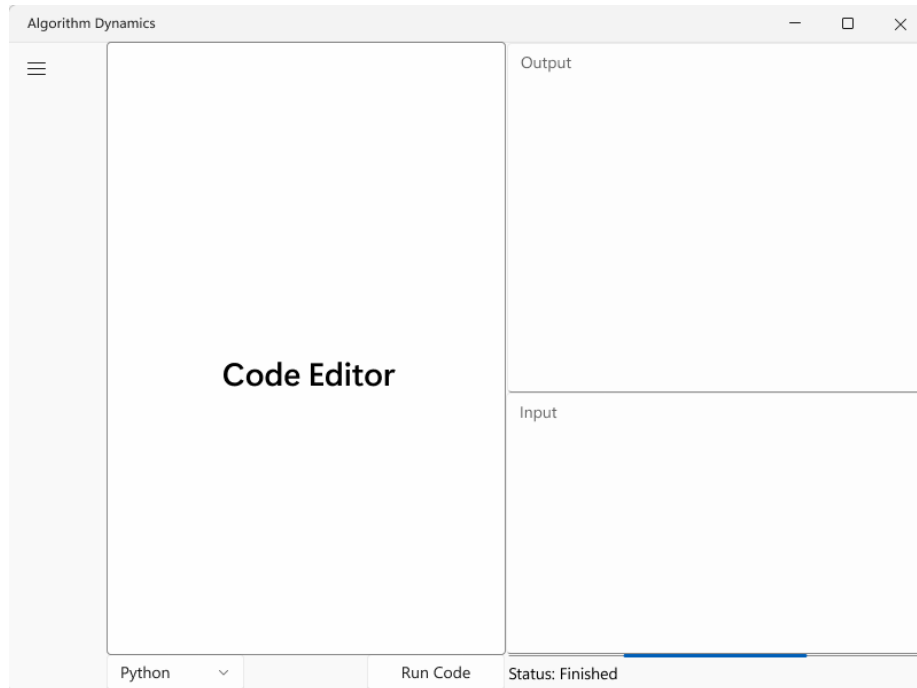
Problem 3

Problem 4

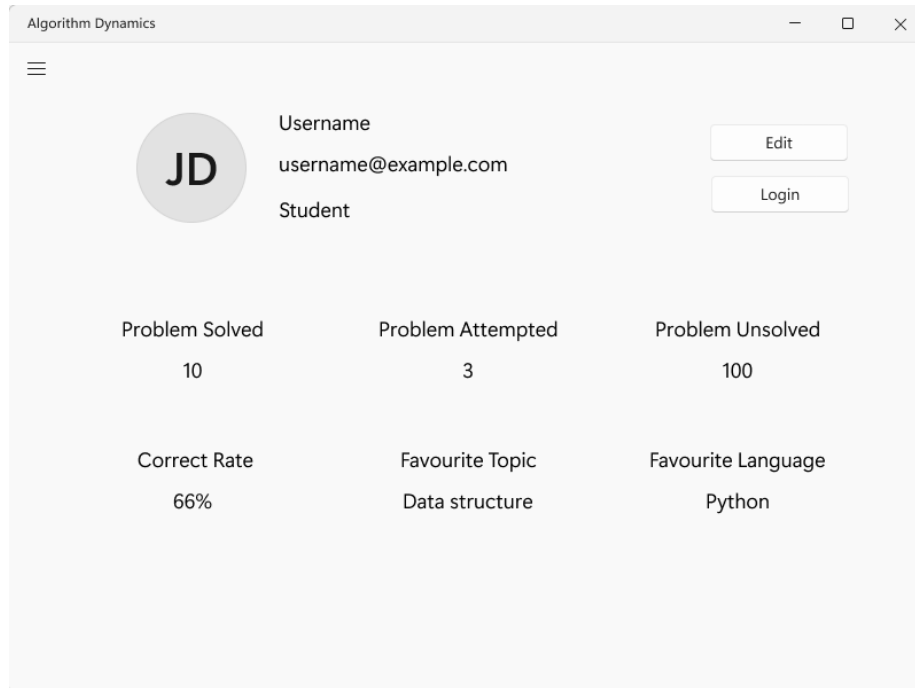
Problem 5

Mark

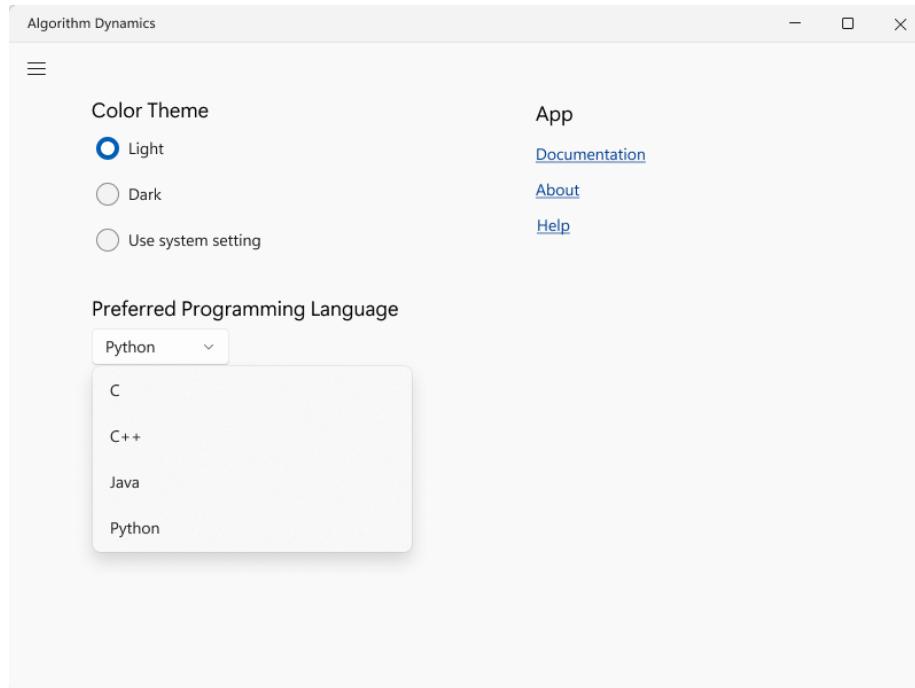
2.1.6 PlaygroundPage



2.1.7 AccountPage



2.1.8 SettingsPage



2.1.9 CreateNewProblemPage

Algorithm Dynamics

Create a New Problem

Problem Name: Problem 1 ×

Tags: Sorting

Difficulty: Easy ▾

Time Limit (ms): 1000 ▾

Memory Limit (MB): 128 ▾

Add Test Case

1 Input Data 1 Output Data 1 ☒ Example ×

2 Input Data 2 Output Data 2 ☐ Example ×

3 Input Data 3 Output Data 3 ☐ Example ×

Cancel Save

Description

Description

Problem description...

Input

Input format...

Output ...

Algorithm Dynamics

Create a New Problem

Problem Name: Problem 1 ×

Tags: Sorting

Difficulty: Easy ▾

Time Limit (ms): Easy ▾

Memory Limit (MB): Medium

Add Test Case

1 Input Data 1 Output Data 1 ☒ Example ×

2 Input Data 2 Output Data 2 ☐ Example ×

3 Input Data 3 Output Data 3 ☐ Example ×

Cancel Save

Description

Description

Problem description...

Input

Input format...

Output ...

No changes will be saved. Do you want to continue?

Yes

2.1.10 CreateNewProblemListPage

The screenshot shows the 'Create a New Problem List' page in the 'Algorithm Dynamics' application. The page has a light gray background and a white header bar with the title 'Algorithm Dynamics' and window control buttons. A hamburger menu icon is in the top left corner.

The main content area is titled 'Create a New Problem List'. It contains the following elements:

- List Name:** A text input field with the value 'Problem 1' and a close button (X).
- Description:** A text input field with the value 'Problem List description'.
- Tags:** A text input field with the value 'Data structure'.
- Add Problem:** A button with a search icon and a dropdown menu. The dropdown menu is open, showing a list of problems: 'Problem 1', 'Problem 2', 'Problem 3', 'Problem 4', 'Problem 5', and 'Problem 6'. 'Problem 5' is highlighted, and its difficulty level 'Easy' is visible.
- Problem List:** A table of existing problems with their tags and difficulty levels.
- Buttons:** 'Cancel', 'Export', and 'Save' buttons at the bottom right.

Problem	Tag	Difficulty
Problem 1	Data structure	
Problem 2	Data structure	
Problem 3	Data structure	
Problem 4	Data structure	
Problem 5	Data structure	Easy
Problem 6	Data structure	Easy

2.1.11 CreateNewAssignmentPage

The screenshot shows a web application window titled "Algorithm Dynamics". Inside, there's a form titled "Create a New Assignment". The form has several input fields: "Assignment Name" (containing "Problem 1"), "Description" (containing "Assignment List description"), "Tags" (containing "Data structure"), and "Due Date" (containing "Pick a date"). Below these is a list of problems. The first problem is "Problem 1" with tag "Data structure". A dropdown menu is open, showing a list of problems: "Problem 1", "Problem 2", "Problem 3", "Problem 4", and "Problem 5" (with tag "Easy"). At the bottom of the form are three buttons: "Cancel", "Export", and "Save".

2.1.12 Judger module

The Judger module will handle all the judging and marking tasks. It takes in a **Submission**, compile and execute the user's code, judge whether it gives the correct output.

2.1.13 Database module

Because this software will store many complex relational data, a relational database is needed to store all the data. Since the software runs locally, and every user will have different data, instead of a central SQL Server, a local SQL Database engine is required.

I choose to use SQLite to power this software. It is a small, self-contained, high-reliability, full-featured SQL Database engine, which will be enough to handle all the data storing and querying requests. Details about the design of the database will be described in the Data structure design section.

2.1.14 REST API module

2.2 Algorithm design

2.2.1 Searchbox searching algorithm

This algorithm will be used to power all the search boxes in the software. It will be packed into a function, taking in a list of item and the searching keyword input by the user, returning a list of results. To make the software easy to use, instead of simply using a linear search and returning all matching results, the algorithm will perform some fuzzy searching, so even the word is not typed in completely, matching results will be returned.

```
1 function search(List<string> sourceList, string keyword)
2     // Create an empty list to store the results
3     resultList = new List<string>()
4     // Split the keyword into pieces by space
5     splitKeyword = keyword.ToLower().Split(' ')
6     // Compare each piece of keyword with the sourceList
7     // Add the matching result into resultList
8     for i=0 to sourceList.Length - 1
9         for j=0 to splitKeyword.Length - 1
10            sourceKey = sourceList[i].ToLower()
11            if sourceKey.Contains(splitKeyword[j]) then
12                resultList.Add(sourceList[i])
13            endif
14        next j
15    next i
16
17    // If no result is found,
18    // add an "not found" notice to the resultList
19    if resultList.Length == 0 then
20        resultList.Add("No results found")
21    endif
22 endfunction
```

In this algorithm, I first create an empty list to store the result. Then I split the keyword into a list by space. So the keyword "A Long Problem Name" will be splitted into ["A", "Long", "Problem", "Name"].

Next, I use two nested loops to perform linear search on each splitted keyword. I choose to use linear search here because the list is not sorted, so only it will work. The overall time complexity here is $O(nm)$ where n is the length of the `sourceList` and m is the length of `splitKeyword`. This is not very fast, but in real word use cases, both n and m will be very small (less than 1000), so this algorithm will be fast enough to handle most of the cases. The increase in complexity brings a better searching experience. In this way, the algorithm

will be able to match keywords like `prob` to result `A Long Problem List`, while normal linear search will not be able to do this.

At the end, if no item is found, a not found notice is added to the list, which will be displayed to the user.

2.2.2 Judger RunCode algorithm

This algorithm is designed for the `Judger` to run a piece of code, pass input to the code and receive all the output and error. This is the basic function of the `Judger` and further judging will all based on this algorithm. An `async` function will be used so it will not block the main UI thread. It takes three input, `UserCode` for the code to be executed, `Input` for the input data, and `TODO` for the programming language. There are two types of programming languages, interpreted programming languages and complied programming languages, the procedure to run them is quite different, so I need to handle them separately. The configuration for each language is stored as `LanguageConfig`.

```
1  async function RunCode(string UserCode, string Input, Language
   ↪   language)
2
3
4  endfunction
```

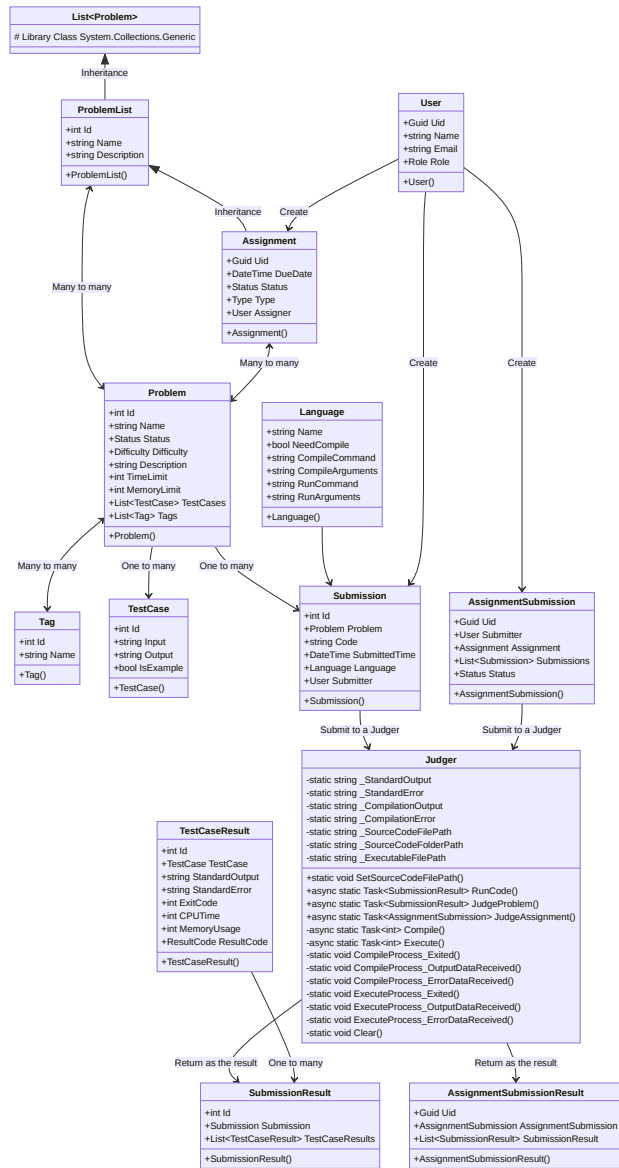
2.2.3 Judger Judge Problem algorithm

2.2.4 Judger Judge Assignment algorithm

2.3 Data structure design

2.3.1 Class design

I am taking an object-oriented approach to the design of the software. This is the class diagram for all classes with their attributes and methods.



User

User is an object used to represent the current user. Its data will be attached to each **Submission**, **Assignment** and **AssignmentSubmission**, so the user can track identities and manage the data easily. Users will be asked to enter these information when they first open the software, an instance of **User** will be created and stored into the database.

Variable	Data type	Justification
Uid	Guid	The User uses a Guid value for its Uid instead of a normal int value to ensure the Uid is unique globally. Every user will have a unique Uid .
Name	string	Name stores the user's name.
Email	string	Email stores the user's Email address.
Role	enum Role	Role is a custom enumeration type which has two possible values, Student or Teacher . It is used to determine the role of the current user, and the user interface will be adjusted accordingly. I choose to use a custom enumeration type instead of some magic numbers to make the code more readable and easier to maintain.

Tag

A **Tag** is a label used to categorize a **Problem**.

Variable	Data type	Justification
Id	int	The Id is the local unique identifier for a Tag . It is used to uniquely determine a Tag in the local database.
Name	string	Name stores the name of the Tag . A Name must be unique, so data needs to be validated and two Tags with the same Name is not allowed.

TestCase

A **TestCase** is one set of test input and output, which will be used by the **Judger** to judge user's submission.

Variable	Data type	Justification
Id	int	The Id is the local unique identifier for a TestCase . It is used to uniquely determine a TestCase in the database.
Input	string	The Input will be feed to the user's submission code by the Judger .
Output	string	Output stores the expected Output of the corresponding Input . The Judger will compare user's output with the Output to judge whether the user's code is correct.
IsExample	bool	Define whether this TestCase is an example. An example TestCase will be displayed to the user for debugging, and distribute with an Assignment . A non-example TestCase will be used to judge the solution and will not be displayed to the user or distribute with an Assignment . A boolean value is suitable to store the two-state data.

Problem

The **Problem** is used to store and organize the data for each individual programming question.

Variable	Data type	Justification
Id	int	The Id is the local unique identifier for a Problem . It is used to uniquely determine a Problem in the database. When exporting a Problem , the Id will not be exported. Instead, a new Id is given when the other user imports the Problem .
Name	string	The Name is a string contains the name of a Problem .

Variable	Data type	Justification
Description	string	The Description is a string storing the detailed description of a Problem . Markdown syntax is supported for a better user experience.
Status	enum Status	Status is an enumeration type containing 3 possible status: Todo , Solved and Attempted . This is used to collect user statistics data. I choose to use a custom enumeration type instead of some magic numbers to make the code more readable and easier to maintain.
Difficulty	enum Difficulty	Difficulty is an enumeration type containing 3 possible difficulties: Easy , Medium and Hard . This provides a way for the user to search and select problems by their difficulties.
TimeLimit	int	TimeLimit sets the max time allowed for the user code to run in millisecond. When the running time exceed the TimeLimit , the running code will be killed and a Time Limit Exceed error will be recorded. This prevents infinite loop from using up all computing resources and rejects inefficient algorithms.
MemoryLimit	int	MemoryLimit sets the max memory allowed for the user's code to consume in bytes. When the memory usage exceed the memory limit, the running code will be killed and a Memory Limit Exceed Error will be recorded. This prevents incorrect code from using up all memory space and rejects inefficient algorithms.
TestCases	List<TestCase>	TestCases is a list containing all TestCase for the problem. A list is more appropriate than an array because it allows new TestCase to be added or remove existing ones during runtime.
Tags	List<Tags>	Tags is a list containing all Tags for a Problem . Similarly, I use a list for Tags so it can be added or removed during runtime.

Language

Language defines the compiling and running configurations for different programming languages. Language configurations are not stored in the database, instead, they are stored in the settings file, so users can add their custom configurations easily.

Variable	Data type	Justification
Name	string	Name stores the name of each programming language. The Name will be displayed in the drop down menu for the user to select their preferred programming language.
NeedCompile	bool	Some programming languages need to be compiled before running, such as C, C++ and Java. This attribute is used to tell the Judger to compile the code before executing.
CompileCommand	string	If a programming language requires compilation, this command is executed to call the compiler.
CompileArguments	string	If a programming language requires compilation, this arguments is passed to the compiler to specify file path and related compile arguments.
RunCommand	string	The Judger uses this command to run the executable or call the interpreter.
RunArguments	string	This Judger pass this arguments to the executable or the interpreter.

Submission

A **Submission** is created when the user submits a code solution to the **Judger**. The **Submission** will contain all the information including the time, the source code, the programming language selected and the corresponding **Problem** for the **Judger** to judge.

Variable	Data type	Justification
Id	int	The Id is the local unique identifier for a Submission . It is used to uniquely determine a Submission in the database.
Problem	Problem	The Problem contains the corresponding Problem of this Submission , which also contains the TestCases for the Judger to judge the Submission . Before storing a Submission into the database, this value needs to be normalized to the Id of that Problem .
Code	string	Code stores the submitted code, which will be executed and judged by the Judger .
SubmittedTime	DateTime	SubmittedTime stores the time the Submission is created. Instead of using a string or an int value, I decide to use the native data type DateTime provided by C#, which makes it easier to process date time, and prevent any formatting issues.
Language	Language	Language stores the programming language selected by the user, so the Judger knows how to run the code.
Submitter	User	Submitter stores the User who submits this Submission . This field will be normalized to the User.Uid before storing into the database.

ProblemList

A **ProblemList** is a list of **Problem**, which allows the user to share multiple **Problem** easily. It is also the parent of **Assignment** and provides basic functions for it. The **ProblemList** is inheriate from the **List<Problem>** class, which is provided by the .NET 5 library. **List<Problem>** provides all basic functions for a list, such as add, remove, sort and find, so I don't need to reinvent the wheel. I choose a list instead of an array because the number of **Problem** inside the list will be changed during runtime, so a list is more appropriate for my use case.

Variable	Data type	Justification
Id	int	The Id is the local unique identifier for a ProblemList . It is used to uniquely determine a ProblemList in the database.
Name	string	Name stores the name of the Problem List .
Description	string	Description stores the description of the ProblemList .

Assignment

An **Assignment** is a **ProblemList** which gets distributed to students. When an **Assignment** is distributed, a copy of that **Assignment** is created. All **TestCases** with **IsExample** set to **false** will be removed to prevent students from cheating. The **Type** of the **Assignment** will be set to **Copy** to indicate it is a distributed copy. The **Judger** will reject to judge a distributed **Assignment** and the **AssignmentsPage** will show the **Assignment** under the Assigned tab instead of the Created tab.

Assignment is inheriate from the **ProblemList**, so it can reuse the **Name** and **Description** attributes and all methods to manage a list of **Problem**. Upon that, new attributes and methods are added to make it functional.

Variable	Data type	Justification
Uid	Guid	The Assignment uses a Guid value for its Uid instead of a normal int value to ensure the Uid is unique globally. So when the user import an Assignment into their database, the Uid will not conflict with any existing values, and it will not be changed (Unlike a ProblemList , for which will be assigned a new Id when importing). The Uid will be referenced by the AssignmentSubmission so the Judger will be able to know which Assignment it is judging.
DueDate	DateTime	The DueDate stores the time for the due date of the Assignment . Instead of using a string or an int value, I decide to use the native data type DateTime provided by C# , which makes it easier to process date time, and prevent all formatting issues.
Status	enum Status	Status is an enumeration type containing 4 possible status for a source Assignment : Draft , Scheduled , Published and Assigned for the teacher to manage the lifecycle of an Assignment . For a distributed Assignment , there are 4 possible Status , NotStarted , InProgress , Completed and OverDue , which helps the student to manage the lifecycle of an Assignment . I choose to use a custom enumeration type instead of some magic numbers to make the code more readable and easier to maintain.
Type	enum Type	Type is an enumeration type containing 2 possible types, Source or Copy . It is used to manage the distribution of an Assignment as described above.
Assigner	User	Assigner stores the User who assigns the Assignment . This field will be normalized into the User.Uid before storing into the database.

AssignmentSubmission

When a student finishes an **Assignment**, an **AssignmentSubmission** is created for the **Judger** to judge. The **AssignmentSubmission** can either be exported to file and sent to the teacher, or it can be uploaded using API. The teacher imports the **AssignmentSubmission** or uses API to load it, and the **Judger** will be able to mark it and return the result.

Variable	Data type	Justification
Uid	Guid	The <code>AssignmentSubmission</code> uses a <code>Guid</code> value for its <code>Uid</code> instead of a normal <code>int</code> value to ensure the <code>Uid</code> is unique globally. So when teachers import an <code>AssignmentSubmission</code> into their database, the <code>Uid</code> will not conflict with any existing values, and it will not be changed.
Submitter	User	The user information is collected when exporting an <code>AssignmentSubmission</code> , so the teacher will be able to know who is the <code>Submitter</code> . This field needs to be normalized into <code>User.Uid</code> before storing into the database.
Assignment	Assignment	The <code>Assignment</code> corresponding to this <code>AssignmentSubmission</code> is recorded, so the <code>Judger</code> knows how to judge it. This field needs to be normalized into <code>Assignment.Uid</code> before storing into the database.
Submissions	List<Submission>	<code>Submissions</code> stores a list of <code>Submission</code> , which contains the user's <code>Submission</code> for each <code>Problem</code> assigned.
Status	enum Status	<code>Status</code> is an enumeration type containing 3 possible types, <code>NotJudged</code> , <code>Judged</code> and <code>Returned</code> for the teacher to keep track of each submission.

Judger

The `Judger` is a static class, only exposes 4 functions to run and judge user's code. The `Judger` takes a `Submission` as an input, and outputs a `SubmissionResult`. No variable is exposed and no data will be saved to the database. The private variables are discussed in the Algorithm design section.

TestCaseResult

`TestCaseResult` stores the judging result for each individual `TestCase`.

Variable	Data type	Justification
Id	int	The Id is the local unique identifier for a <code>TestCaseResult</code> . It is used to uniquely determine a <code>TestCaseResult</code> in the database.
TestCase	TestCase	<code>TestCase</code> stores the corresponding <code>TestCase</code> that produces this result. This field will be normalized into <code>TestCase.Id</code> before storing into the database.
StandardOutput	string	<code>StandardOutput</code> stores the content output by the user's code.
StandardError	string	<code>StandardError</code> stores the error messages received (if any) when executing the user's code.
ExitCode	int	<code>ExitCode</code> stores the exit code of the user's code.
CPUTime	int	<code>CPUTime</code> stores the time it takes for the user's code to execute.
MemoryUsage	int	<code>MemoryUsage</code> stores the memory usage record for the user's code.
ResultCode	enum ResultCode	<code>ResultCode</code> is a custom enumeration type storing the judging result. It has 7 possible values, <code>WRONG_ANSWER</code> , <code>SUCCESS</code> , <code>COMPILE_ERROR</code> , <code>TIME_LIMIT_EXCEEDED</code> , <code>MEMORY_LIMIT_EXCEEDED</code> , <code>RUNTIME_ERROR</code> or <code>SYSTEM_ERROR</code> .

SubmissionResult

When the `Judger` finishes judging a `Submission`, a `SubmissionResult` is created and stored. It includes the result and statistics about the `Submission`.

Variable	Data type	Justification
Id	int	The Id is the local unique identifier for a SubmissionResult . It is used to uniquely determine a Submission in the database.
Submission	Submission	Submission stores the corresponding Submission of this result. It needs to be normalized into the Submission.Id before storing into the database.
TestCaseResults	List<TestCaseResult>	TestCaseResults is a list of TestCaseResult , storing result of each individual TestCase .

AssignmentSubmissionResult

When the **Judger** finishes judging a **AssignmentSubmission**, a **SubmissionResult** is created and stored. It includes the result and statistics about the **AssignmentSubmission**.

Variable	Data type	Justification
Uid	Guid	It uses a Guid value for its Uid instead of a normal int value to ensure the Uid is unique globally.
AssignmentSubmission	AssignmentSubmission	It stores the corresponding submission of this result. It needs to be normalized into the Id before storing into the database.
SubmissionResults	List<SubmissionResult>	It is a list of SubmissionResult , storing result of each individual Submission .

2.3.2 Settings design

The settings will be stored in a JSON file.

```

1  {
2      "Theme": "Light",
3      "SelectedLanguage": "Python",
4      "LanguageConfiguration":
5      [
6          {
7              "Name": "C++",
8              "NeedCompile": true,
9              "CompileCommand": "g++",
10             "CompileArguments": "-x c++ {SourceCodeFilePath} -o
11             ↪ {ExecutableFilePath}",
12             "RunCommand": "{ExecutableFilePath}",
13             "RunArguments": ""
14         },
15         {
16             "Name": "Python",
17             "NeedCompile": false,
18             "CompileCommand": "",
19             "CompileArguments": "",
20             "RunCommand": "python",
21             "RunArguments": "{SourceCodeFilePath}"
22         }
23     ],
24     "CurrentUser": "6ee2ebef-3f50-43b7-adf4-78c460339fd0"
25 }

```

The **Theme** field stores the current color theme. There are three possible values, **Light**, **Dark** or **Default**. The data will be validated before applying, if it is empty or invalid, the **Default** theme will be applied and write back to this field.

The **SelectedLanguage** field stores the default programming language selected by the user. The value will be validated before applying, if it is empty or invalid, the internal default Python programming language configuration will be applied and write back to this field.

The **LanguageConfiguration** is a list of dictionarys allowing the user to configure custom programming languages. The data will be validated before applying, if a configuration is invalid, it will be ignored.

The **CurrentUser** field stores the **Uid** of the **User**. The data will be validated. If the **Uid** is not found in the database or the field is empty, the user will be asked to login and a new value will be generated and written back to this field.

The entire JSON file will be validated before any data is fetched. If the file does not exist, a default template will be created. If the format or the encoding is wrong, a default templete will override the incorrect file.

2.3.3 Database design

User table

The User table will have `Uid` as its primary key. Since Sqlite does not support `Guid` data type natively, it will be converted into a string for storage. Same reason why the `Role` is stored in integer, and because the enumeration type is implemented in integer under the hood, there is no data conversion issue.

All fields will be validated and no empty field is allowed.

The GUID will look like this: `149f3e41-dcdb-43d5-8964-75249489f6cf`. It is a long random unique string of characters. Because it is too long, in this section, a placeholder `<GUID>` will be used to represent a string of GUID.

Uid	Name	Email	Role
<GUID>	"UserName"	"UserName@test.com"	0

```
1 CREATE TABLE User
2 (
3     Uid TEXT PRIMARY KEY NOT NULL,
4     Name TEXT NOT NULL,
5     Email TEXT NOT NULL,
6     Role INTEGER NOT NULL
7 )
```

Problem table

The Problem table will have an auto increment integer `Id` as its primary key. Both `Status` and `Difficulty` will be stored in integer for the same reason before.

All fields will be validated and no empty field is allowed.

Id	Name	Description	Status	Difficulty	TimeLimit	MemoryLimit
1	"name"	".."	0	0	1000	1000

```
1 CREATE TABLE Problem
2 (
3     Id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
4     Name TEXT NOT NULL,
5     Description TEXT NOT NULL,
6     Status INTEGER NOT NULL,
7     Difficulty INTEGER NOT NULL,
8     TimeLimit INTEGER NOT NULL,
9     MemoryLimit INTEGER NOT NULL
10 )
```

Tag table

The Tag table will have an auto increment integer Id as its primary key.

All fields will be validated and no empty field is allowed.

Id	Name
1	"Tag1"

```
1 CREATE TABLE Tag
2 (
3     Id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
4     Name TEXT NOT NULL UNIQUE
5 )
```

TestCase table

The TestCase table will have an auto increment integer Id as its primary key. Since Sqlite does not support boolean data type, an integer will be used to store the data of IsExample, 0 will represent false and 1 will represent true.

All fields will be validated and no empty field is allowed.

The TestCase has a many-to-one relation to the Problem, one Problem can have multiple TestCase while one TestCase can only belong to one Problem. So, a foreign key ProblemId is enabled for the TestCase to store its corresponding Problem.

Id	Input	Output	IsExample	ProblemId
1	"Input1"	"Output1"	1	1

```
1 CREATE TABLE TestCase
2 (
3     PRIMARY KEY (Id) INTEGER NOT NULL AUTO_INCREMENT,
4     Input TEXT NOT NULL,
5     Output TEXT NOT NULL,
6     IsExample INTEGER NOT NULL,
7     FOREIGN KEY (ProblemId) REFERENCES Problem(Id)
8 )
```

Submission table

The Submission table will have an auto increment integer Id as its primary key. Since Sqlite does not support DateTime data type, the time will be converted to a string before storing. Similarly conversion will be done for Language and SubmitterId.

All fields will be validated and no empty field is allowed, except for the Code, empty Submission is allowed.

A Submission has a one-to-many relationship with a Problem, a Problem can have multiple Submission while a Submission can only be submitted to one Problem. It also has a many-to-one relation with a User. A User can create multiple Submission while a Submission can only be created by one User. So it will have two foreign keys to establish such relationship.

Id	ProblemId	Code	SubmittedTime	Language	SubmitterId
1	1	source code	11/11/2021 14:20:04	Python	<GUID>

```

1 CREATE TABLE Submission
2 (
3     PRIMARY KEY (Id) INTEGER NOT NULL AUTO_INCREMENT,
4     FOREIGN KEY (ProblemId) REFERENCES Problem(Id),
5     Code TEXT,
6     SubmittedTime TEXT NOT NULL,
7     Language TEXT NOT NULL,
8     FOREIGN KEY (SubmitterId) REFERENCES User(Uid),
9 )

```

ProblemList table

Id	Name	Description
1	ProblemList 1	Description for ProblemList 1

```

1 CREATE TABLE Submission
2 (
3     PRIMARY KEY (Id) INTEGER NOT NULL AUTO_INCREMENT,
4     Name TEXT NOT NULL,
5     Description TEXT
6 )

```

Assignment table

AssignmentSubmission table

TestCaseResult table

SubmissionResult table

AssignmentSubmissionResult table

ProblemTag table

The **Tag** has a many-to-many relation with the **Problem**. A **Problem** can have multiple **Tags** and a **Tag** can be assigned to multiple **Problems**. So a link table is required to normalize the data.

Id	ProblemId	TagId
1	1	1
2	1	2

```
1 CREATE TABLE ProblemTag
2 (
3     PRIMARY KEY(Id) INTEGER NOT NULL AUTO_INCREMENT,
4     FOREIGN KEY (ProblemId) REFERENCES Problem(Id),
5     FOREIGN KEY (TagId) REFERENCES Tag(Id),
6 )
```


2.3.4 Import/Export design

Import/Export a Problem

Import/Export a ProblemList

Import/Export an Assignment

Import/Export an AssignmentSubmission

Import/Export an AssignmentSubmissionResult

2.4 Development testing

2.4.1 Milestones

1. Create the main interface
2. Handle settings
3. Implement the Judger
4. Implement data structures
5. Create the database
6. Handle data import/export
7. Handle API calls

2.4.2 Milestone 1: Create the main interface

TODO: testing plan

Test input	Expect output	Justification
------------	---------------	---------------

2.4.3 Milestone 2: Handle settings

TODO: testing plan

Test input	Expect output	Justification
------------	---------------	---------------

2.4.4 Milestone 3: Implement the Judger

TODO: testing plan

Test input	Expect output	Justification
------------	---------------	---------------

2.4.5 Milestone 4: Implement data structures

TODO: testing plan

Test input	Expect output	Justification
------------	---------------	---------------

2.4.6 Milestone 5: Create the database

TODO: testing plan

Test input	Expect output	Justification
------------	---------------	---------------

2.4.7 Milestone 6: Handle data import/export

TODO: testing plan

Test input	Expect output	Justification
------------	---------------	---------------

2.4.8 Milestone 7: Handle API calls

Test input	Expect output	Justification
Click the login button in the Account page, and input the correct email and password.	Login successfully, the information on the Account page is updated correctly.	A normal test to make sure the software can process the login API correctly.

2.5 Post-development testing

2.5.1 Alpha testing

2.5.2 Beta testing

2.5.3 Integration testing

Chapter 3

Development

Chapter 4

Evaluation

(TODO): Add evaluation chapter