A Level Programming Project Report

Junrong Chen

November 20, 2021

Contents

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 save 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 compared 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 recognization' 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.

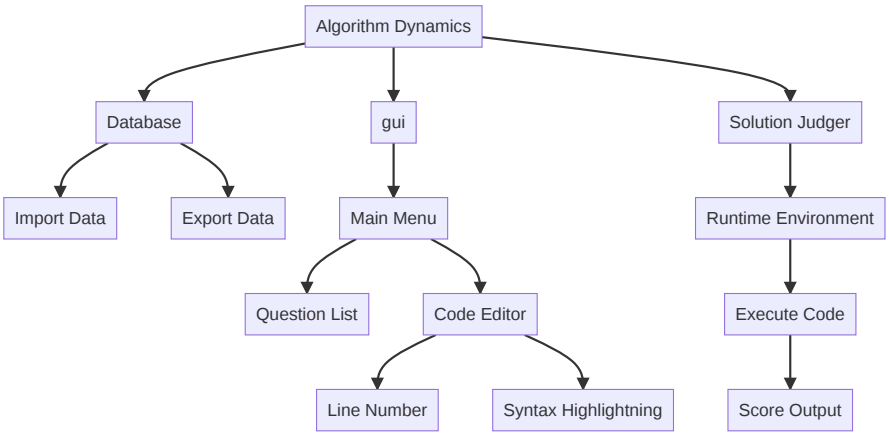
In reality, the students' answer is sent to a teacher to mark it against the mark scheme. The teacher needs to read the code line by line and check whether it is correct. This process is abstracted into a judger that marks the code against pre-generated test cases, which transforms a problem that originally 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 judgers 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 learn and practise?
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 learn and practise?

   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. Adding 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 learn and practise 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 learn and practise?

   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.

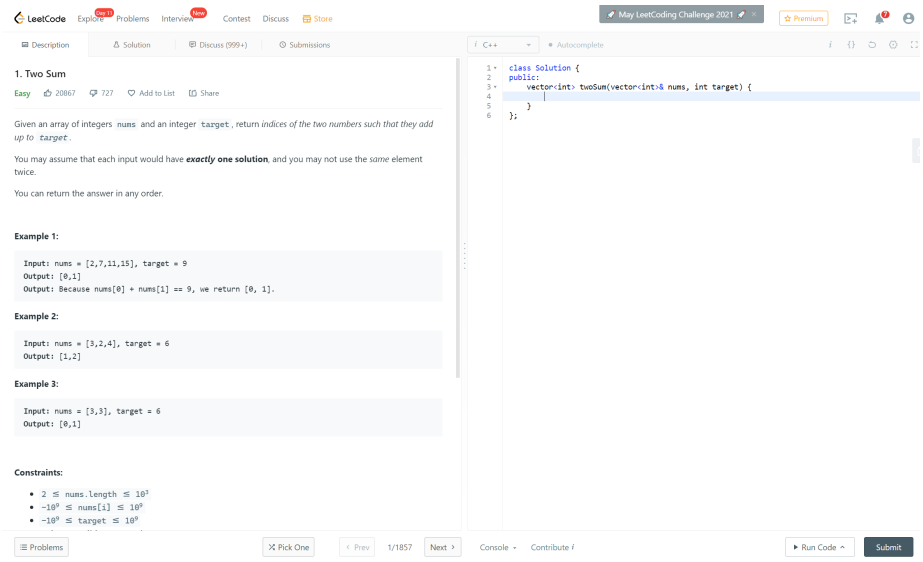
- LeetCode
- Codeforces

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.

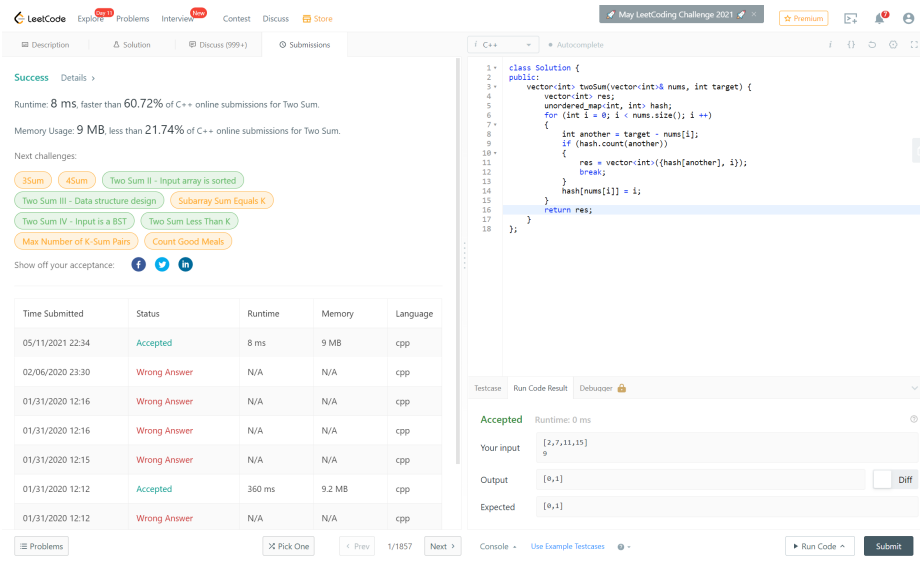
LeetCode provides a database containing more than 1000 coding questions.

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.

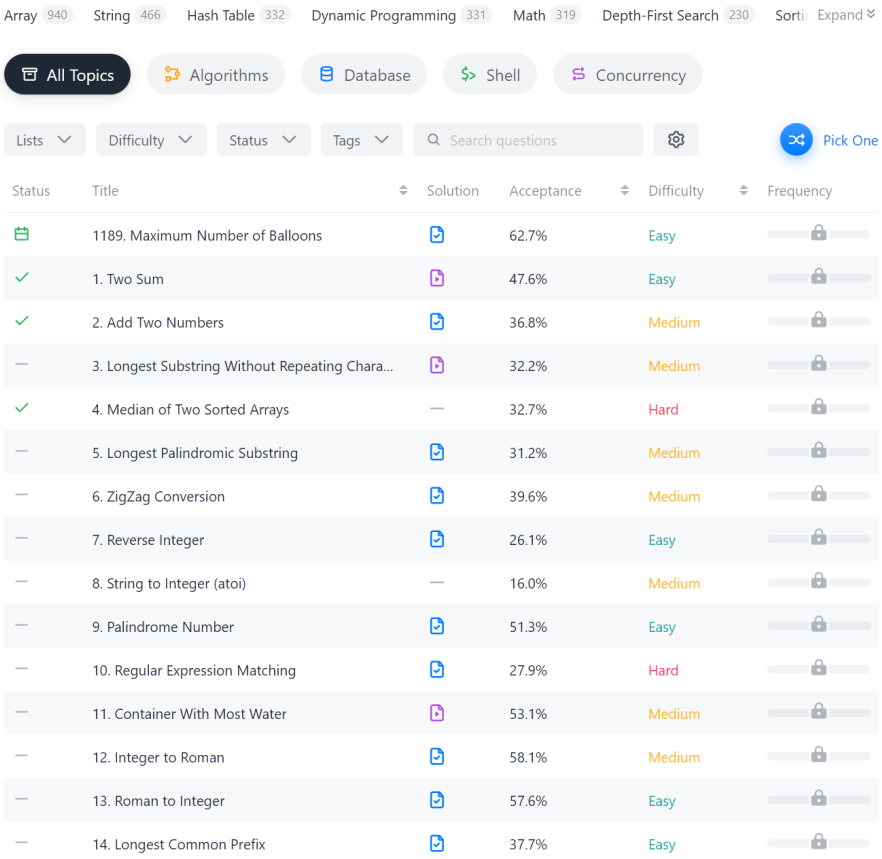


The code editor provides line number and syntax highlighting functions. User can change their programming language with a drop-down box. LeetCode supports all mainstream programming languages. My solution should be able to support multiple programming languages as well, which allows students with different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

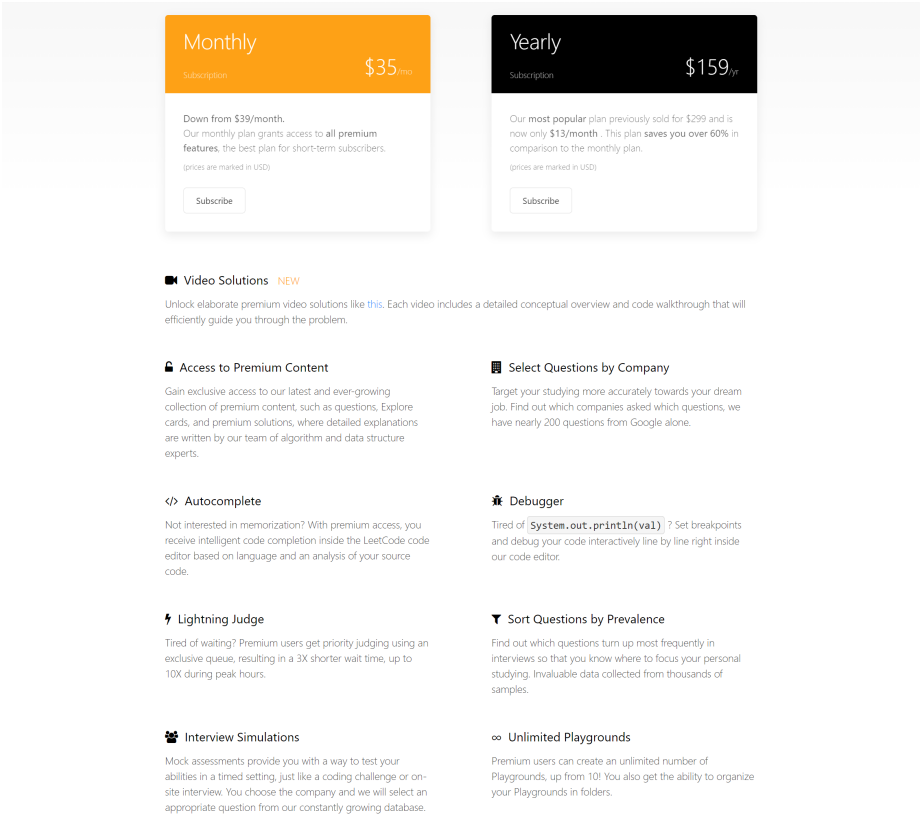
The split view design is clean and handy. The user can see the question and write their solution on the same page without switching between different windows. The design of examples and the 'Run Code' button is useful as well. I can refer to LeetCode's coding layout when designing my solution's interface.

Question database



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 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.

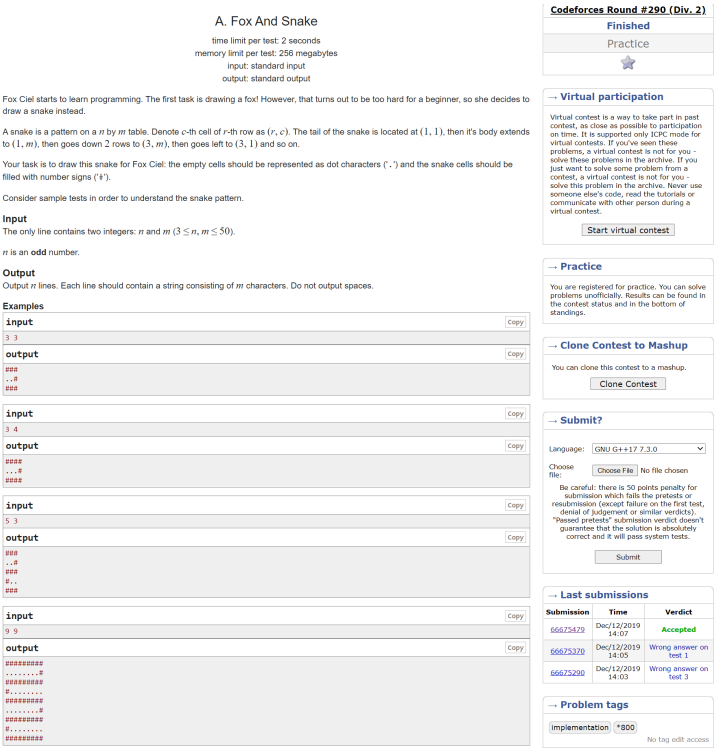
LeetCode does not support custom questions or any functions for educators. It is mainly designed for self-learners. My solution is designed for school use, so

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 hold coding competitions.

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 ('.') 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 ($3 \le n, m \le 50$).

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

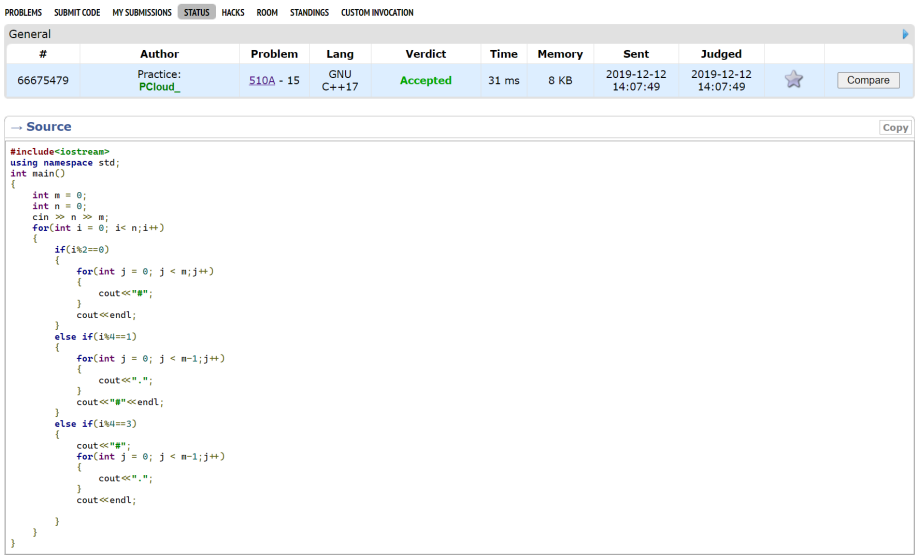
→ Last submissions

Submission	Time	Verdict
66675479	Dec/12/2019 14:07	Accepted
66675370	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

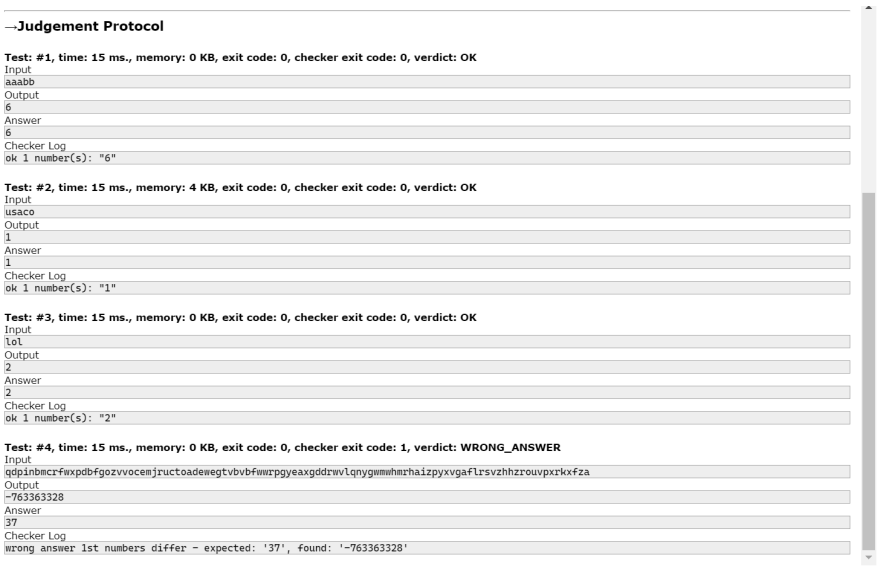
#	Author	Problem	Lang	Verdict	Time	Memory	Sent	Judged		
66675479	Practice: PCloud_	510A - 15	GNU C++17	Accepted	31 ms	8 KB	2019-12-12 14:07:49	2019-12-12 14:07:49		Compare

→ 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
usaco
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
lol
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
qdpinbmcrfwxpdbfgozvvocemjructoadewegtvbvbfwwrpgyeaxgddrwvlqnygwmwhmrhaizpyxvgaflrsvzhhzrouvpxrkxfza
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 shared 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, distributing assignments, collecting assignments, and marking as-

signments. 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 switching 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 extra software is inconvenient and may violate the IT security policy of some schools.

The judger can only accept 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.

Distributing the questions and assignments is still 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 judger 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 judger. 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	Allow 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 allows 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.
Use 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.

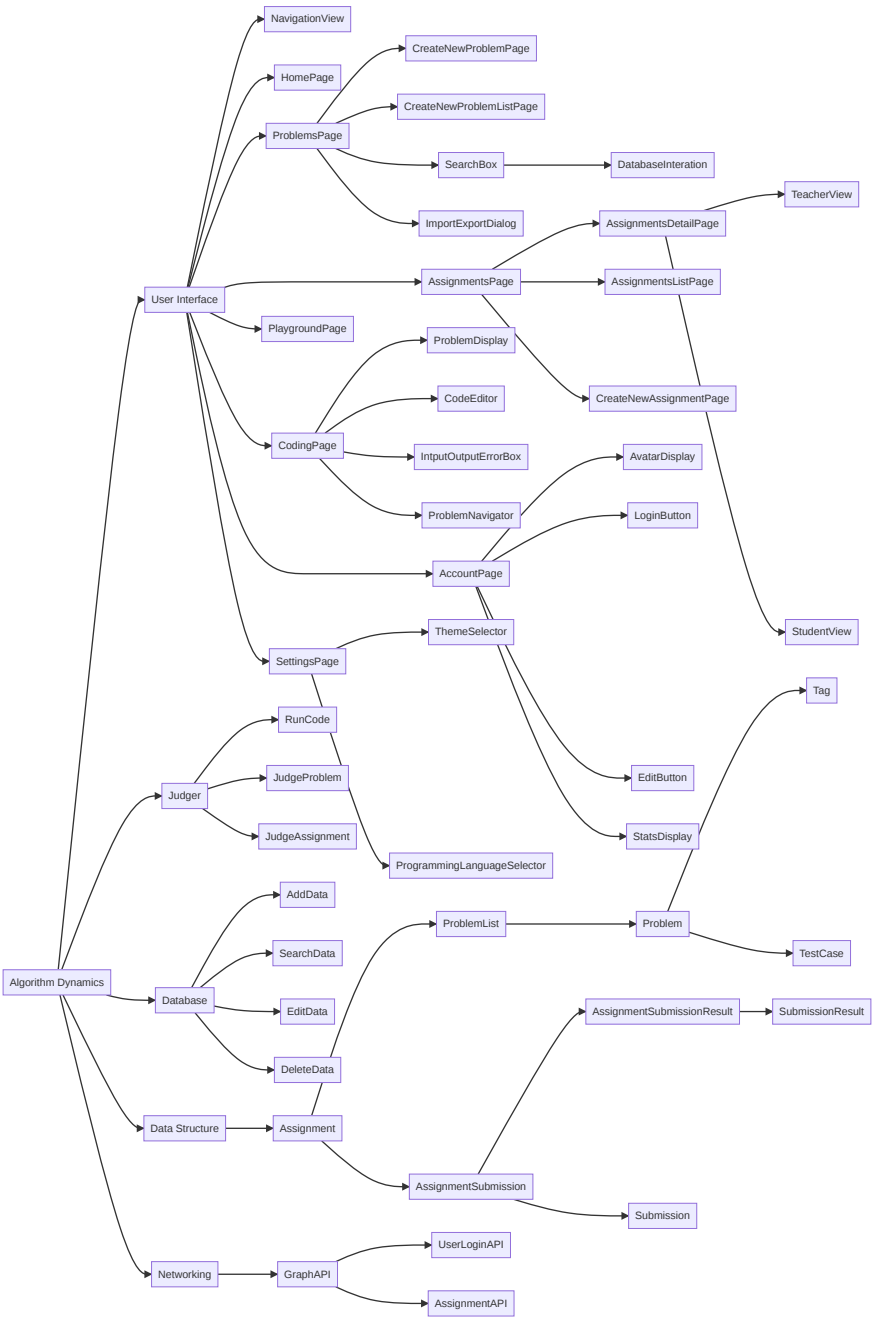
All the success criteria will first be tested through the unit tests created during the development. Then they will be tested and improved by me during the Alpha Testing stage. Finally, they will be tested by the stakeholders in the Beta Testing stage, and the evaluation will be based on their user experience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

Chapter 2

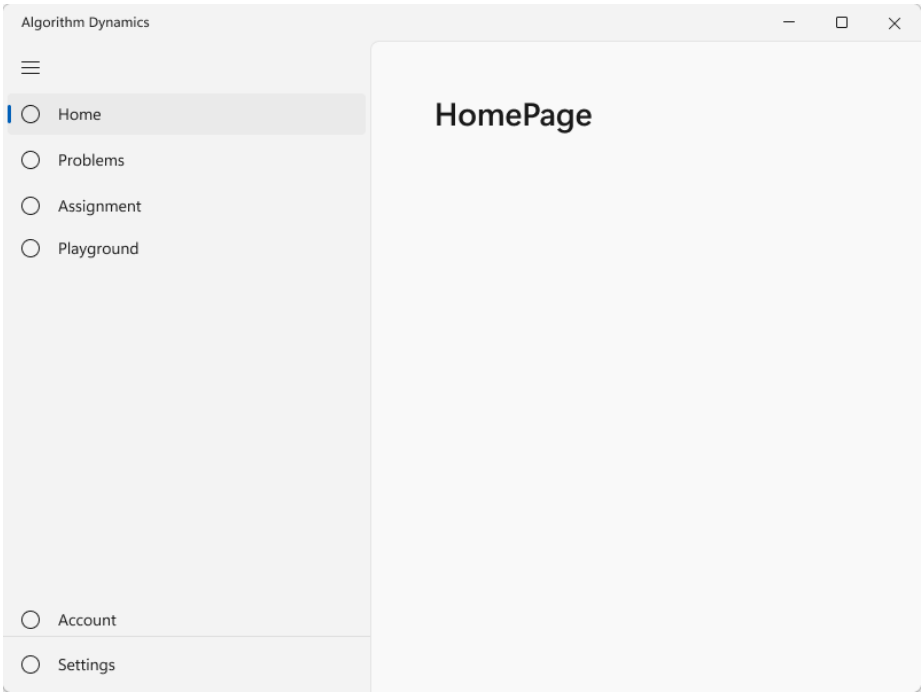
Design

2.1 Decomposition

As shown in the decomposition diagram on the next page, the entire problem will be decomposed into 5 main sub-problems: user interface, judger module, database module, data structure design and API module. Each of which will be further decomposed as explained below.


Algorithm Dynamics
User Interface
NavigationView
HomePage
ProblemsPage
CreateNewProblemPage
CreateNewProblemListPage
SearchBox
DatabaseInteration
ImportExportDialog
PlaygroundPage
CodingPage
ProblemDisplay
CodeEditor
IntputOutputErrorBox
ProblemNavigator
AccountPage
AvatarDisplay
LoginButton
SettingsPage
ThemeSelector
AssignmentsPage
AssignmentsDetailPage
TeacherView
AssignmentsListPage
CreateNewAssignmentPage
StudentView
Judger
RunCode
JudgeProblem
JudgeAssignment
ProgrammingLanguageSelector
EditButton
StatsDisplay
Tag
Database
AddData
SearchData
EditData
DeleteData
ProblemList
Problem
TestCase
Data Structure
Assignment
AssignmentSubmission
AssignmentSubmissionResult
SubmissionResult
Submission
Networking
GraphAPI
UserLoginAPI
AssignmentAPI

2.1.1 NavigationView



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

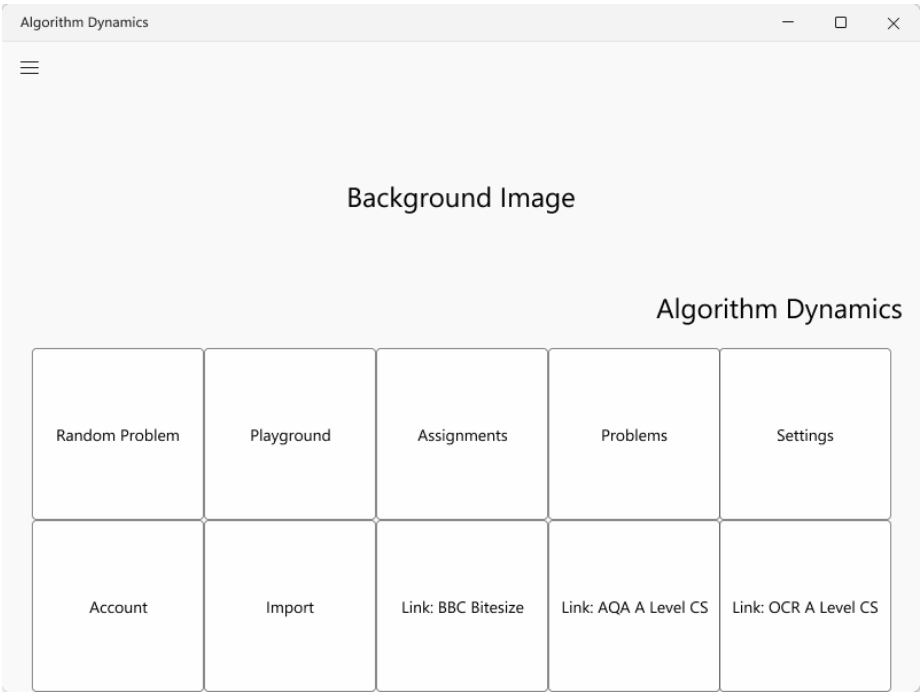
Usability feature

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 with 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



The HomePage is the first page that gets displayed to the user. On the top of the HomePage, there will be a beautiful background image under the 'Algorithm Dynamics' title. On the bottom, there are 10 useful buttons link to different functions.

Usability feature

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 directed to these websites, which makes it easy for the user to look up specifications and revise content. All the useful functions of the software are grouped on the HomePage, which makes them easily accessible and makes the software easy to use.

Validation

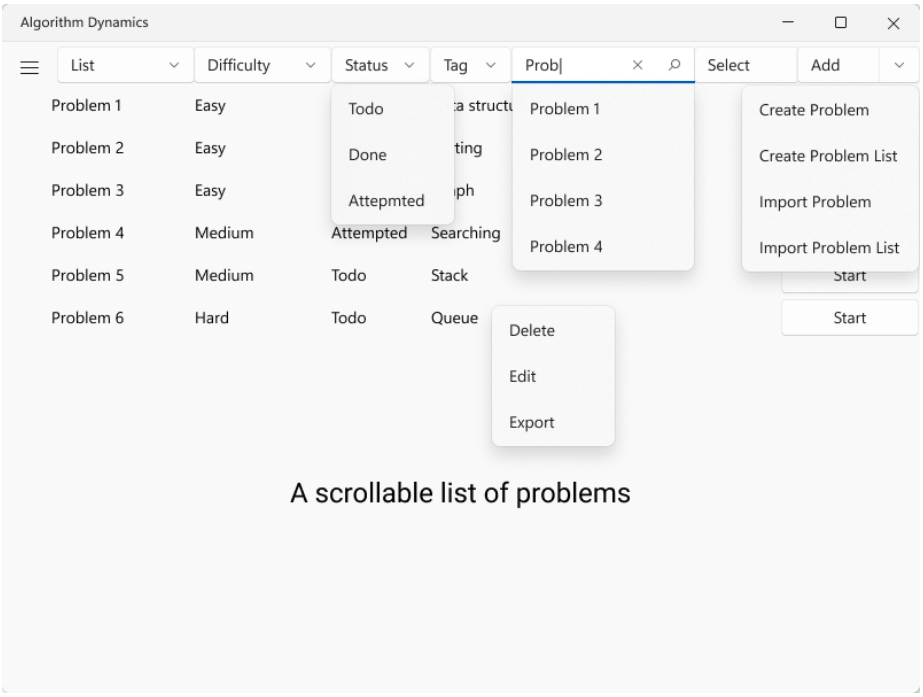
There are only buttons in the HomePage for the user to click, so only valid actions can be taken, no further data validation is required.

Stakeholder feedback

Timofei thinks putting the links on the same page makes it messy. He thinks those websites and resources can be easily find on the Internet, so there is no need to put extra links on the HomePage.

PCloud points out that the icon grid needs to be responsive, so when he uses the software on a small screen, all buttons should still be displayed properly.

2.1.3 ProblemsPage



The ProblemsPage displays all 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

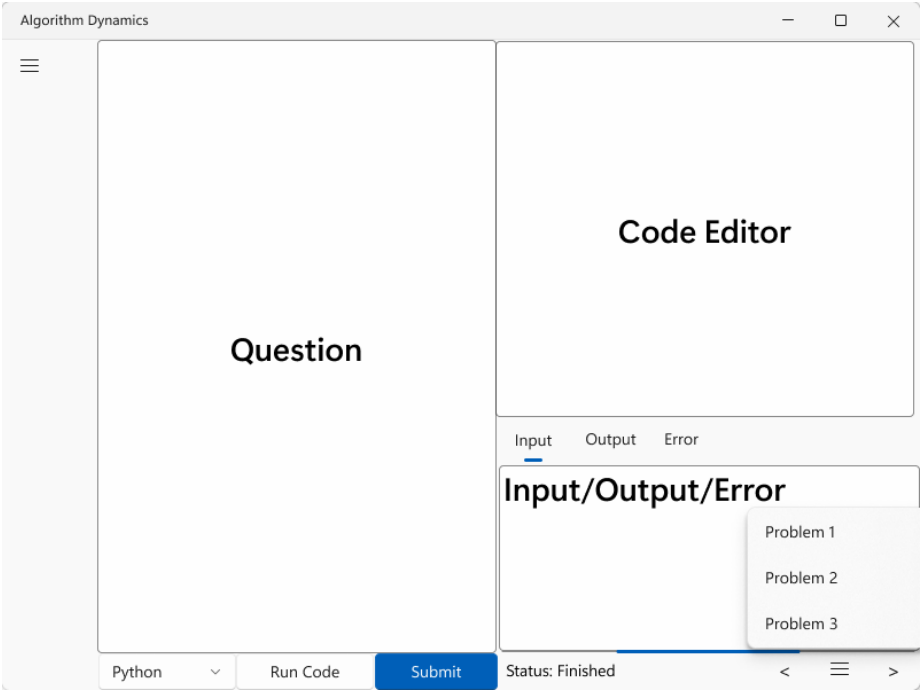
The user can apply the selection condition by either selecting different fields in the dropdown boxes or directly typing into the search box. A scrollable list of problems that match the conditions will be shown below with detailed information. The user can start working on a problem by clicking the 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 apply 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. I will implement an advanced searching algorithm to improve the quality of the search results, and I will explain the algorithm in the algorithm design section.

Validation

Most components on this page are still buttons, users can only click them and no invalid data can be input. The search box is where the user can input some text only. A flyout will be displayed to promote the user to click the button instead of inputting data themselves. Also, a length check will be applied. The user can only enter a maximum of 32 characters so they cannot crash the search box or 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 prevents SQL injection and provide a better searching experience.

2.1.4 CodingPage



The CodingPage is where the user works on a programming problem. It contains a question panel, a code editor and an IO panel.

Usability feature

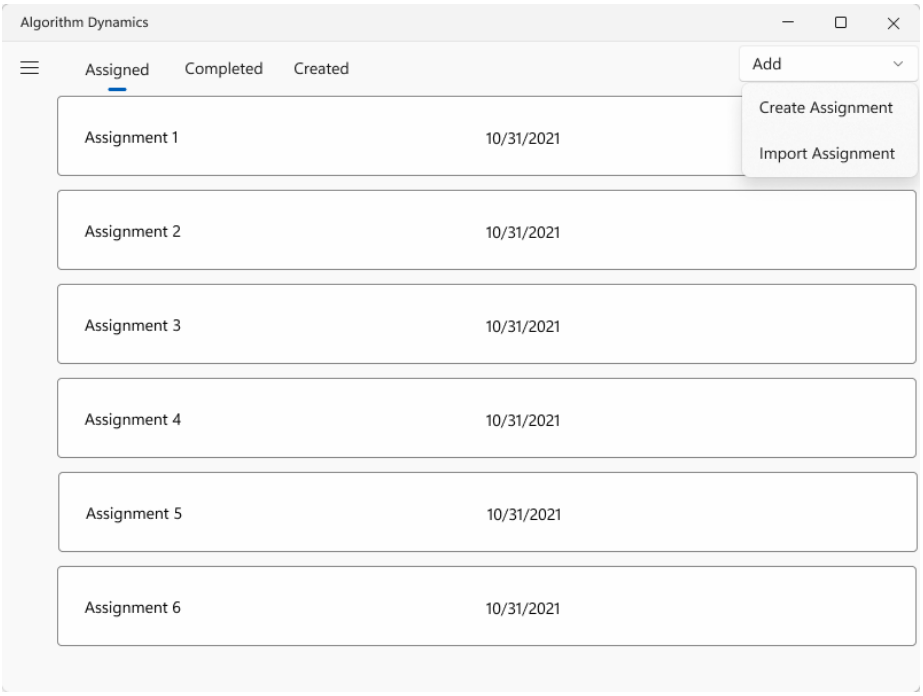
The question is displayed on the left and a code editor will be displayed on the top right. The input, output and error messages will be displayed on the button right, the user can switch between them by clicking the corresponding tab. On the bottom, the user can select programming language using a dropdown 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 judger and 3 navigation buttons on the button right to make it easy to navigate between different problems. There is a progress bar between the IO panel and the navigation buttons, it displays the judging progress.

The code editor will support line numbers, basic syntax highlighting and keyboard shortcuts to make it easy to use.

Validaton

Again, most of the components on the page are either read-only (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 validated by the compiler or the interpreter of the selected programming language. However, the IO panel requires further validation. To prevent the user from printing out a huge amount of data which might result in poor performance, the IO panel will perform a length check and only display the first 2048 characters of the output.

2.1.5 AssignmentsPage



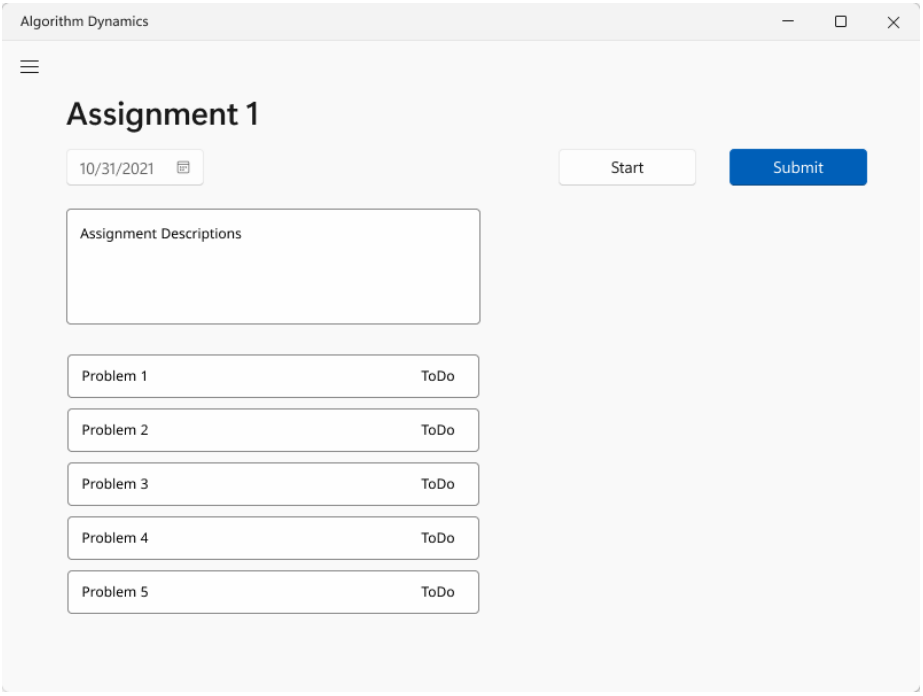
The AssignmentPage is where the user interacts with the assignments. For students, they can work on their assignments; for teachers, they can create and mark assignments.

Usability feature

The user can switch between different tabs by clicking the navigation bar at the top. They can enter the detailed view of an assignment by clicking one assignment. A dropdown button is placed on the top right for the user to create or import a new assignment. All assignments will be displayed in a list with their due date.

Validation

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



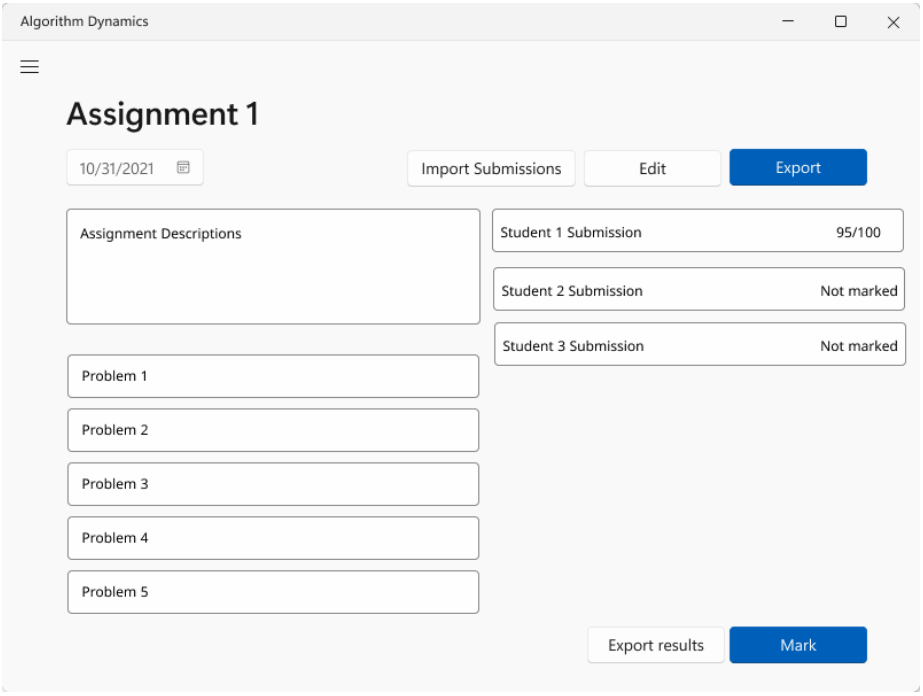
This is the assignment details page for students, which gets displayed when a student clicks on an assignment.

Usability feature

The descriptions and due date of the assignment are displayed on the top left. A list of problems is displayed below where the user can see their status and start working on one by clicking it. When the user clicks the start button, he will be navigated to the CodingPage and the problems will be loaded. When he finishes all problems, he can click the submit button to either submit it through API or export it to a file to send to the teacher for marking.

Validation

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



This is the assignment details page for teachers, which gets displayed when a teacher clicks on an assignment.

Usability feature

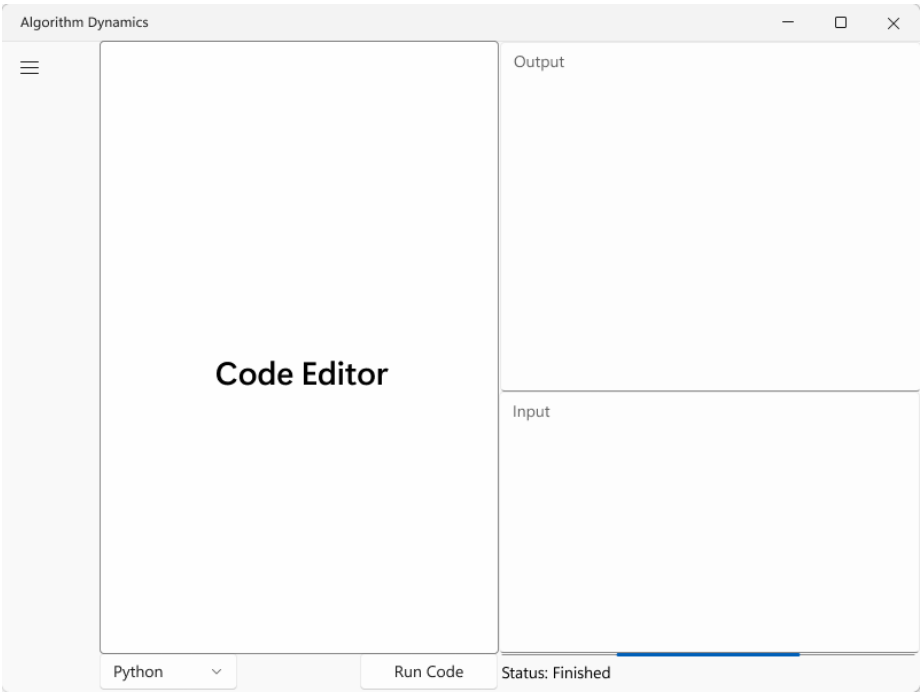
The teacher can click the Import Submissions button to import students' submissions or click the edit button to edit the assignment, or the export button to distribute the assignment. All student submissions and their status will be listed on the right, the teacher can click the mark button to mark all of them automatically and click the student submission to see the code details in the CodeingPage.

After marking, the teacher can click the export results button to export all student's marks into a CSV file for further data analysis.

Validation

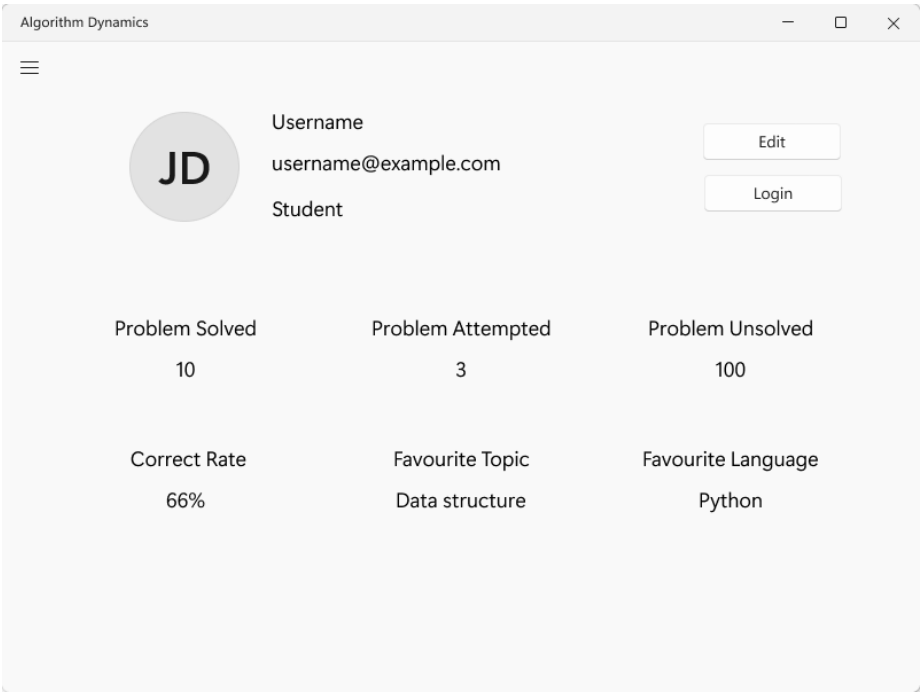
There are only buttons for the user to click, so only valid actions can be taken, no further data validation is required.

2.1.6 PlaygroundPage



The PlaygroundPage is a free space for the user to run any code quickly. It is essentially the same page as the CodingPage but without a coding problem. The interface is slightly changed with a larger code editor and a separate IO panel for a better user experience.

2.1.7 AccountPage



The AccountPage is used to manage the current user and display user statistics.

Usability feature

If the user has login into his Microsoft account, the avatar, username, email and role will be managed by his Microsoft Account, the edit button will navigate to the Microsoft account website, and the login button will be replaced with a logout button. If the user has a login locally, he can edit his information by clicking the edit button and log in to Microsoft account by clicking the login button.

Some interesting statistics will be displayed on the button.

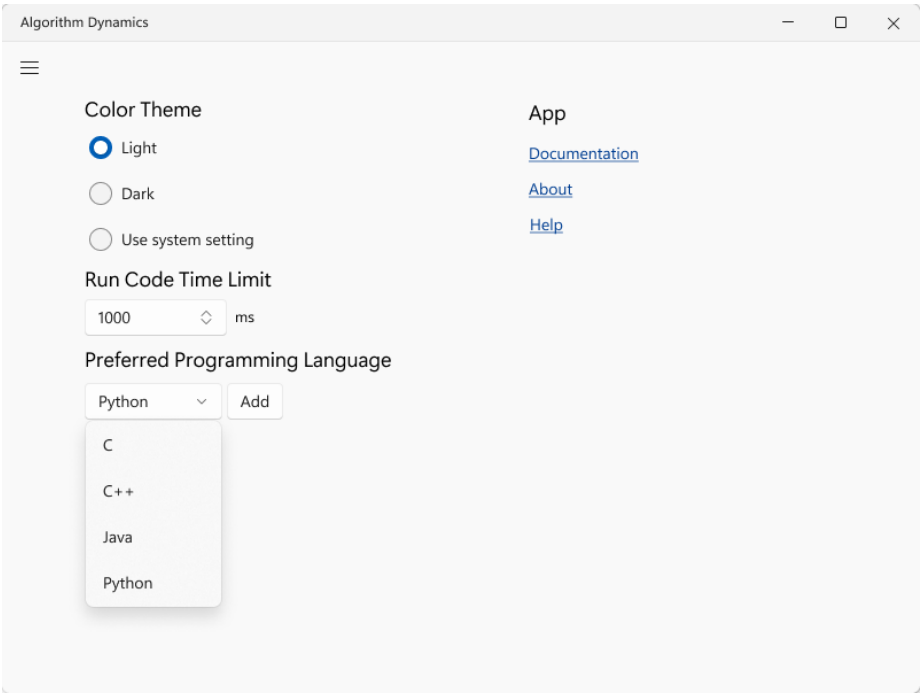
Validation

When the user edits his local identity, the user name will not be allowed to exceed 32 characters, and a format check will be applied to the email address.

Stakeholder feedback

Timofei thinks there can be more interesting statistics, such as favourite variable names, and the total time spend in the software.

2.1.8 SettingsPage



The SettingsPage is where the user adjusts software settings.

Usability feature

The user can adjust the colour theme of the software to use a light theme, dark theme or system default theme. The user can also change the run code time limit and programming language profiles in the SettingsPage.

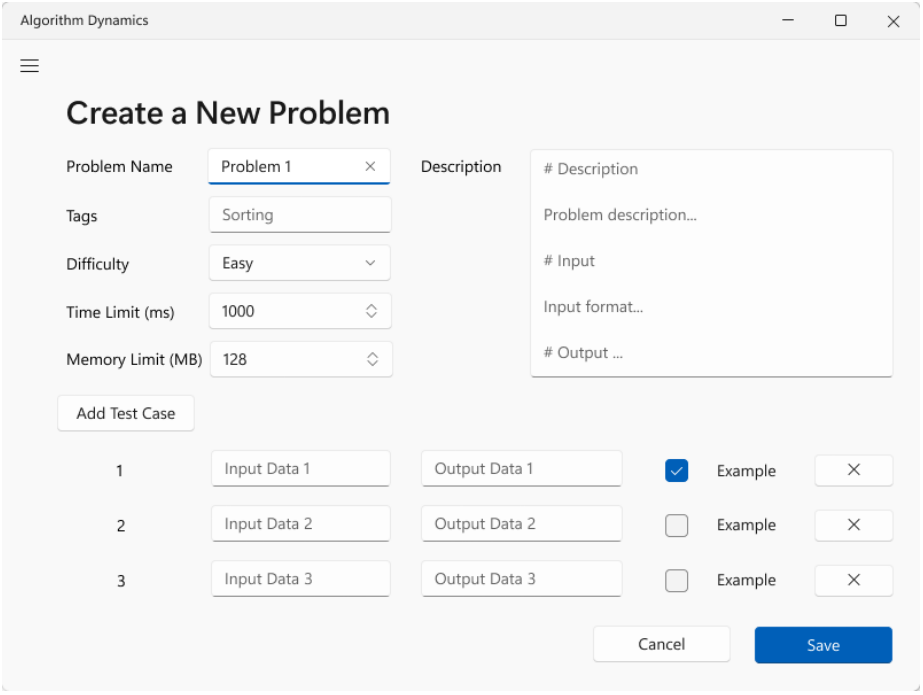
Validation

Radio buttons are used for colour theme selection so it is impossible to enter any invalid data. A number box is used for the time limit settings so only positive integers can be input. A dropdown list is used for the programming language setting so only valid input is accepted.

Stakeholder feedback

Timofei likes the idea of being able to switch between a light theme and a dark theme. He thinks it will be better if users are allowed to create their custom theme. He also suggests that allowing the user to change the font size manually may help a lot.

2.1.9 CreateNewProblemPage



The CreateNewProblemPage is used to create new problems. It contains multiple text boxes for the user to input problem information.

This page will be reused for editing existing problems. The title will be altered correspondingly.

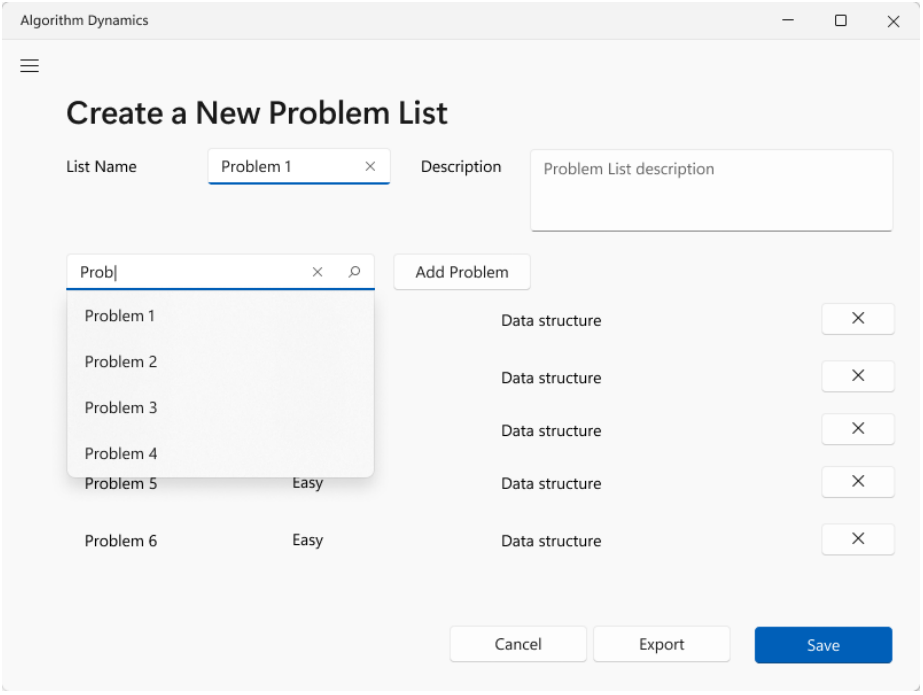
Usability feature

The Tags field will be a tokenizing text box with an auto suggestion dropdown menu to make input faster. The Difficulty field will have a dropdown list for easy selection. The Time Limit and Memory Limit fields will be number boxes. The user can add a new test case by clicking the add button and removing existing ones by clicking the delete button. If the user clicks the cancel button, a flyout will be displayed to confirm cancellation.

Validation

There are many input boxes on this page, they will all be validated. For the Problem Name field, the user can enter 32 characters at maximum. In the Tags field, the user can enter 5 tags at maximum. In the Time Limit and Memory Limit fields, the user can only enter numbers.

2.1.10 CreateNewProblemListPage



The CreateNewProblemListPage is used to create a new problem list. It contains multiple text boxes for the user to input problem list information.

The page will be reused for editing existing problem lists. The title will be altered correspondingly.

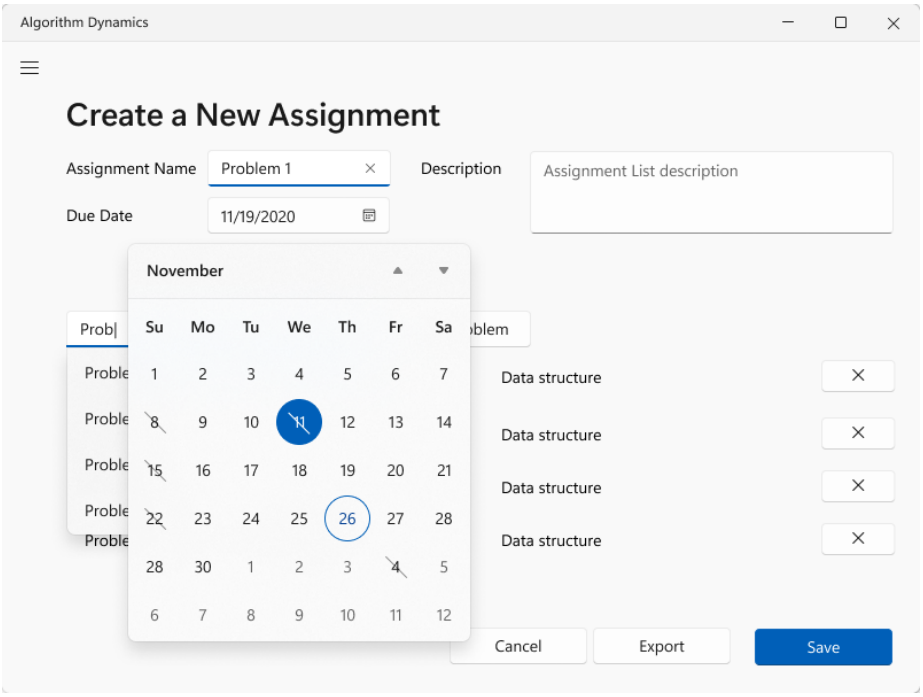
Usability feature

There will be an autosuggest box for the user to search for problems and add them to the list. There is an export and a save button at the button for the user to export or save the problem list. If the user clicks the cancel button, a flyout will be displayed to confirm cancellation. The test cases will be a scrollable list to display multiple test cases.

Validation

A similar validation is performed just like the CreateNewProblempage. For the List Name field, the user can enter 32 characters at maximum.

2.1.11 CreateNewAssignmentPage



The CreateNewAssignmentPage is used to create new assignments. It contains multiple text boxes for the user to input assignment data.

The page will be reused for editing existing assignments as well. The title will be altered correspondingly.

Usability feature

The due date will be a DateTime picker, a visual calendar will be displayed for the user to pick a due date. An autosuggest search box will be placed for the user to search for problems easily.

Validation

The DateTime picker will prevent invalid dates from being input, the name field will have a maximum of 32 characters limit.

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, judges whether it gives the correct output.

The Judger will take a few seconds to judge each `Submission`, while it is judging, the other parts of the software need to keep working on their tasks. So I use concurrent processing to let the Judger run parallel with the main program.

When judging a single `TestCase`, the Judger will first start the compiler in a new process to compile the user's code. The Judger redirects the output of the compiler to an internal string variable to store all outputs. When the compiler finishes compiling, an interrupt is triggered and the Judger will be notified that the compilation has been done. It will check the compiler's exit code and its output, if there are any compilation errors, they will be reported back. Otherwise, the Judger will run the compiled executable, and feed the input to its `stdin` and redirects all `stdout` and `stderr` to internal string variables. On top of that, the Judger will start a countdown timer in a different process to track the time. When the countdown reaches zero, an interrupt will be sent to the Judger and it will terminate the user's code and report a time limit exceed error, or if the user's code finishes first, then its output will be compared with the expected output, the result and all statistics will then be reported.

A problem may contain multiple test cases, so all test cases will be first pushed into a judging queue, and the judger will judge them one by one.

Similarly, an assignment may contain multiple problems, all problems will be first pushed into a problem queue, and each problem will be processed one by one.

The detailed judging algorithm will be explained later in the algorithm design section.

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.

The SQLite database has some limitations. It only supports five data types. `TEXT`, `INTEGER`, `NULL`, `REAL` and `BLOB`. So other data types such as `DateTime` and `Guid` will need to be converted before storing.

I choose to use Microsoft.Data.SQLite for the database interface, which allows me to send SQL requests to the database, handle errors and provide safety features to prevent SQL injection. Details about the design of the database and queries will be described in the Database design section.

2.1.14 API module

The API module will allow the user to log in to their Microsoft Education account, and interact with the Education Assignment function. Users will be able to create and manage assignments using education assignments APIs provided by Microsoft.

When the user clicks the login button in the SettingsPage, the API module will handle the login requests. When a logged-in user opens the AssignmentPage, the API module will fetch all assignments. The API module will also allow the teachers to distribute assignments directly through API calls and the students to hand in submissions without any kind of exporting.

The API module will be implemented at the end after all local functions are working correctly. The corresponding API calls and endpoints will be discussed later.

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 items 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.

```
function search(sourceList, keyword)
    // Create an empty list to store the results
    resultList = new List<string>()
    // Split the keyword into pieces by space
    splitKeyword = keyword.ToLower().Split(' ')
    // Compare each piece of keyword with the sourceList
    // Add the matching result into resultList
    for i=0 to sourceList.Length - 1
        for j=0 to splitKeyword.Length - 1
        sourceKey = sourceList[i].ToLower()
            if sourceKey.Contains(splitKeyword[j]) then
                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 a linear search on each keyword. I choose to use the 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-world 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 in `A Long Problem List`, while normal linear search will not be able to do this.

In 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 base on this algorithm.

Before anything can run, the Judger will need to first write the user's code to a source code file. And before it can do that, it needs to know where it should write to. I will create three private variables `_SourceCodeFilePath`, `_SourceCodeFolderPath` and `_ExecutableFilePath` to store the file paths for the source code file and the executable file. I use the idea of encapsulation here so these variables can only be set by the public method `SetSourceCodeFilePath`, so they will not be modified unexpectedly.

```
1   partial class Judger
2       private _SourceCodeFilePath
3       private _SourceCodeFolderPath
4       private _ExecutableFilePath
5
6       public procedure SetSourceCodeFilePath(FolderPath, FileName)
```

```
7           _SourceCodeFolderPath = FolderPath
8           _SourceCodeFilePath = FolderPath + FileName + ".txt"
9           _ExecutableFilePath = FolderPath + FilePath + ".exe"
10      endprocedure
11  endclass
```

For compiling programming languages such as C, C++ and Java, the source code needs to be compiled before running. So I need to add a `Compile` function to support these programming languages. The compile task needs to be run concurrently in a different process so the main UI can stay responsive. So instead of monitoring the compiler all the time, I will bind two event handlers to the compile process, to process its output data and error data. Those data will then be stored in two private string variables for later use.

```
1   partial class Judger
2       private _CompilationOutput
3       private _CompilationError
4
5       private procedure CompileProcess_ErrorDataReceived(sender, e)
6           // Validate the data before storing
7           if string.IsNullOrEmpty(e.Data) == False then
8               _CompilationError += e.Data + '\n'
9           endif
10      endprocedure
11
12      private procedure CompileProcess_OutputDataReceived(sender,
        ↪  e)
13          // Validate the data before storing
14          if string.IsNullOrEmpty(e.Data) == False then
15              _CompilationOutput += e.Data + '\n
16          endif
17      endprocedure
18  endclass
```

Now, it is ready to run the compiler. Before it can start compiling, the old outputs need to be cleaned. I design two magic variables in the language configurations allowing more flexible arguments, these magic variables need to be pre-processed before compiling. A process start info needs to be created to let the Judger know how to start the new process. After binding the event handlers, the compiler can start running.

```
1   partial class Judger
2       private function Compile()
3           // Clear the old data
4           _CompilationOutput = ""
5           _CompilationError = ""
6
```

```
7           // Pre-process language compile command and arguments
8           string fileName =
            ↪  language.CompileCommand.Replace("{SourceCodeFilePath}",
            ↪  _SourceCodeFilePath).Replace("{ExecutableFilePath}",
            ↪  _ExecutableFilePath)
9           string arguments =
            ↪  language.CompileArguments.Replace("{SourceCodeFilePath}",
            ↪  _SourceCodeFilePath).Replace("{ExecutableFilePath}",
            ↪  _ExecutableFilePath)
10
11          // Create a new process start info
12          ProcessStartInfo StartInfo = new ProcessStartInfo
13          (
14              FileName: fileName,
15              Arguments: arguments,
16              UseShellExecute: false,
17              CreateNoWindow: true,
18              RedirectStandardOutput: true,
19              RedirectStandardError: true,
20          )
21
22          // Create a new compile process
23          Process CompileProcess = new Process(StartInfo)
24
25          // Bind event handlers to the compile process
26          CompileProcess.OutputDataReceived += new
            ↪  DataReceivedEventHandler(CompileProcess_OutputDataReceived)
27          CompileProcess.ErrorDataReceived += new
            ↪  DataReceivedEventHandler(CompileProcess_ErrorDataReceived)
28
29          // Start the compile process
30          CompileProcess.Start()
31          CompileProcess.BeginOutputReadLine()
32          CompileProcess.BeginErrorReadLine()
33          CompileProcess.WaitForExitAsync()
34
35          // Return the exit code
36          return CompileProcess.ExitCode
37      endfunction
38  endclass
```

In the end, an integer exit code will be returned. If the exit code is zero, the Judger will know that the compilation is successful. If it is something else, then the Judger will need to report a compile error with detailed data from `_CompilationError`.

If the user's code is compiled successfully or it does not need to be compiled, the `Execute` function is called to run the code or the executable, pass the input to the process and get its output. The `Execute` function will also need to kill

the running process if it runs too long. So I will use a custom enumeration type `StatusCode` to manage the process' status.

```
enum StatusCode
    PENDING,
    RUNNING,
    FINISHED,
    TIME_LIMIT_EXCEEDED,
    MEMORY_LIMIT_EXCEED
endenum

partial class Judger
    private _StatusCode
endclass
```

`StatusCode` has five possible values, `PENDING` is set when the code is waiting to be run. When it starts executing, the status code will be set to `RUNNING`. If it finishes in time, its status will be set to `FINISHED`. If it exceeds the time limit, the process will be killed by the Judger and a `TLE` will be recorded. If it exceeds the memory limit, the process will be killed by the Judger and a `MLE` will be recorded.

Similarly, I will use event handlers to record all the outputs from the user's code. I add a third handler to process the exit event, which will determine the `_StatusCode` mentioned above.

```
partial class Judger
    private _StandardOutput
    private _StandardError

    private procedure ExecuteProcess_Exited(sender, e)
        // Only successfully finished when the status code is not
        ↪ TLE or MLE
        if _StatusCode != StatusCode.TIME_LIMIT_EXCEEDED and
        ↪ _StatusCode != StatusCode.MEMORY_LIMIT_EXCEED then
            _StatusCode = StatusCode.FINISHED
        endif
    endprocedure

    private procedure ExecuteProcess_OutputDataReceived(sender,
    ↪ e)
        // Validate the data before storing
        if string.IsNullOrEmpty(e.Data) == False then
            _StandardOutput += e.Data + '\n'
        endif
    endprocedure

    private procedure ExecuteProcess_ErrorDataReceived(sender, e)
```

```
20          // Validate the data before storing
21          if string.IsNullOrEmpty(e.Data) == False then
22              _StandardError += e.Data + '\n'
23          endif
24      endprocedure
25  endclass
```

The `Execute` function has a similar design with the `Compile` function. It configures all the running information and event handlers as usual. However, this time I add two extra processes running concurrently with the execution process. The first one is a timer, which countdown the time limit. When the timer counts to zero, it checks whether the execute process is still running, if it does, then the execution process will be killed and a `TLE` will be reported. The second one is a memory monitor, which checks the peak memory usage of the execution process every 10 milliseconds. If the memory usage is larger than the memory limit, then the execution process will be killed and a `MLE` will be reported.

```
1   partial class Judger
2       private _WorkingSet64;
3       private function Execute(Input, Language, TimeLimit,
        ↪  MemoryLimit)
4           // Clear the old data
5           _StandardOutput = ""
6           _StandardInput = ""
7           _WorkingSet64 = 0
8
9           // Initialize status code
10          _StatusCode = StatusCode.PENDING
11
12          // Pre-process language run command and arguments
13          string fileName =
            ↪  Language.RunCommand.Replace("{SourceCodeFilePath}",
            ↪  _SourceCodeFilePath).Replace("{ExecutableFilePath}",
            ↪  _ExecutableFilePath)
14          string arguments =
            ↪  Language.RunArguments.Replace("{SourceCodeFilePath}",
            ↪  _SourceCodeFilePath).Replace("{ExecutableFilePath}",
            ↪  _ExecutableFilePath)
15
16          // Create a new process start info
17          ProcessStartInfo StartInfo = new ProcessStartInfo
18          (
19              FileName: fileName,
20              Arguments: arguments,
21              UseShellExecute: false,
22              CreateNoWindow: true,
23              RedirectStandardInput: true,
24              RedirectStandardOutput: true,
25              RedirectStandardError: true,
```

```
26          )
27
28          // Create a new execute process
29          Process ExecuteProcess = new ExecuteProcess(StartInfo)
30
31          // Bind event handlers to the compile process
32          ExecuteProcess.OutputDataReceived += new
            ↪  DataReceivedEventHandler(ExecuteProcess_OutputDataReceived)
33          ExecuteProcess.ErrorDataReceived += new
            ↪  DataReceivedEventHandler(ExecuteProcess_ErrorDataReceived)
34          ExecuteProcess.Exited += new
            ↪  EventHandler(ExecuteProcess_Exited)
35
36          // Start the execute process
37          ExecuteProcess.Start()
38          ExecuteProcess.BeginOutputReadLine()
39          ExecuteProcess.BeginErrorReadLine()
40          ExecuteProcess.StandardInput.WriteLine(Input);
41
42          // Set the status code to Running
43          _StatusCode = StatusCode.RUNNING
44
45         // Start the timer
46          Timer timer = new Timer(() =>
47          {
48              // If the user's code is still running
49              // when the timer has finished
50              // Kill the user's code and record a TLE
51              if ExecuteProcess.HasExited == false then
52                  ExecuteProcess.Kill()
53                  _StatusCode = StatusCode.TIME_LIMIT_EXCEEDED
54              endif
55          }, null, TimeLimit, Timeout.Infinite)
56
57          // Start the memory monitor
58          Thread MemoryMonitor = new Thread(() =>
59          {
60              while ExecuteProcess.HasExited == false
61                  ExecuteProcess.Refresh()
62                  _WorkingSet64 = ExecuteProcess.PeakWorkingSet64
63                  if _WorkingSet64 > MemoryLimit then
64                      ExecuteProcess.Kill()
65                      _StatusCode =
                        ↪  StatusCode.MEMORY_LIMIT_EXCEEDED
66                  endif
67              endwhile
68          });
69          MemoryMonitor.Start()
70
71          // Wait the process to finish
```

```
72          ExecuteProcess.WaitForExitAsync()
73
74          // Return the exit code
75          return ExecuteProcess.ExitCode
76      endfunction
77  endclass
```

Now I have everything I need to build the `RunCode` function. It takes in the user's code, input data, language configuration, time limit and memory limit, outputs a `RunCodeResult` containing all the running information. I also need to set up a stopwatch to measure how long the code has been executed, unlike the timer before, this stopwatch only provides data for statistics records. In the end, the `RunCode` function will handle some basic errors, it will judge whether the user's code run, but it will not judge whether the user's code provides the correct output. The `RunCode` algorithm only runs the code and further judging will be handed over to other algorithms.

```
1   partial class Judger
2       public function RunCode(UserCode, Input, Language, TimeLimit,
        ↪   MemoryLimit)
3           // Create a new TestCaseResult
4           RunCodeResult result = new RunCodeResult()
5
6           // Write the source code to file
7           sourceCodeFile = openWrite(_SourceCodeFilePath)
8           sourceCodeFile.write(UserCode)
9           sourceCodeFile.close()
10
11          // Compile if needed
12          if Language.NeedCompile then
13              // Compile and record the exit code
14              exitCode = Compile(Language)
15
16              // If compile failed then return a COMPILE_ERROR
17              if exitCode != 0 then
18                  result.StandardOutput = _CompilationOutput
19                  result.StandardError = _CompilationError
20                  result.ResultCode = ResultCode.COMPILE_ERROR
21                  return result
22              endif
23          endif
24
25          // Set up a Stopwatch to record the running time
26          Stopwatch watch = new Stopwatch()
27          watch.Start()
28
29          // Execute the code
30          exitCode = Execute(Input, Language, TimeLimit)
31
```

```
32          watch.Stop()
33
34          // Store the stats to the result
35          result.StandardOutput = _StandardOutput
36          result.StandardError = _StandardError
37          result.CPUTime = watch.ElapsedMilliseconds
38
39          // If time limit exceed, return TLE
40          if _StatusCode == StatusCode.TIME_LIMIT_EXCEEDED then
41              result.ResultCode = ResultCode.TIME_LIMIT_EXCEEDED
42              return result
43          endif
44
45          // If memory limit exceed, return MLE
46          if _StatusCode == StatusCode.MEMORY_LIMIT_EXCEEDED then
47              result.ResultCode = ResultCode.MEMORY_LIMIT_EXCEEDED
48              return result
49          endif
50
51          // If receive a error or exit code is not zero
52          // Return a runtime error
53          if string.IsNullOrEmpty(result.StandardError) == False or
            ↪  exitCode != 0 then
54              result.ResultCode = ResultCode.RUNTIME_ERROR
55              return result
56          endif
57
58          // Otherwise, return success
59          result.ResultCode = ResultCode.SUCCESS
60          return result
61      endfunction
62  endclass
```

2.2.3 Judger Judge TestCase algorithm

This algorithm is designed to allow the Judger to judge a test case. On top of the `RunCode` function, this algorithm compare the user's output with the expected output, so it can judge whether the user's code works correctly. The `JudgeTestCase` function returns a `TestCaseResult` containing more information.

```
1   partial class Judger
2       public function JudgeTestCase(UserCode, TestCase, Language,
        ↪  TimeLimit)
3           // Run the code with RunCode and get the result
4           TestCaseResult result = (TestCaseResult)RunCode(UserCode,
            ↪  TestCase.Input, Language, TimeLimit)
5           result.TestCase = TestCase
```

```
6
7           // If the code is executed successfully
8           // Judge whether its output matches the expected output
9           if result.ResultCode == ResultCode.SUCCESS then
10
11              // Trim the trailing spaces before comparing
12              userOutput = result.StandardOutput.Trim()
13              expectedOutput = TestCase.Output.Trim()
14
15              // If they are not matched
16              // Report wrong answer
17              if userOutput != expectedOutput then
18                  result.ResultCode = ResultCode.WRONG_ANSWER
19              endif
20          endif
21          return result
22      endfunction
23  endclass
```

2.2.4 Judger Judge Problem algorithm

This algorithm is designed to judge a submission. I set up a judging queue and push all test cases into the queue. Then I take them out and judge them one by one. A queue is suitable for this use case because I want the first test case entered to be judged first. A submission result will be returned at the end.

```
1   partial class Judger
2       public function JudgeProblem(Submission)
3           // Create the submission result
4           SubmissionResult result = new SubmissionResult()
5           result.Submission = Submission
6
7           // Create the judging queue
8           Queue JudgeQueue = new
            ↪  Queue(Submission.Problem.TestCases)
9
10          // Keep judging until the queue is empty
11          while len(JudgeQueue) > 0
12              Code = Submission.Code
13              TestCase = JudgeQueue.Dequeue()
14              Language = Submission.Language
15              TimeLimit = Submission.Problem.TimeLimit
16              MemoryLimit = Submission.Problem.MemoryLimit
17              TestCaseResult = JudgeTestCase(Code, TestCase,
                ↪  Language, TimeLimit, MemoryLimit)
18              result.Add(TestCaseResult)
19          endwhile
20          return result
```

```
21      endfunction
22  endclass
```

2.2.5 Judger Judge Assignment algorithm

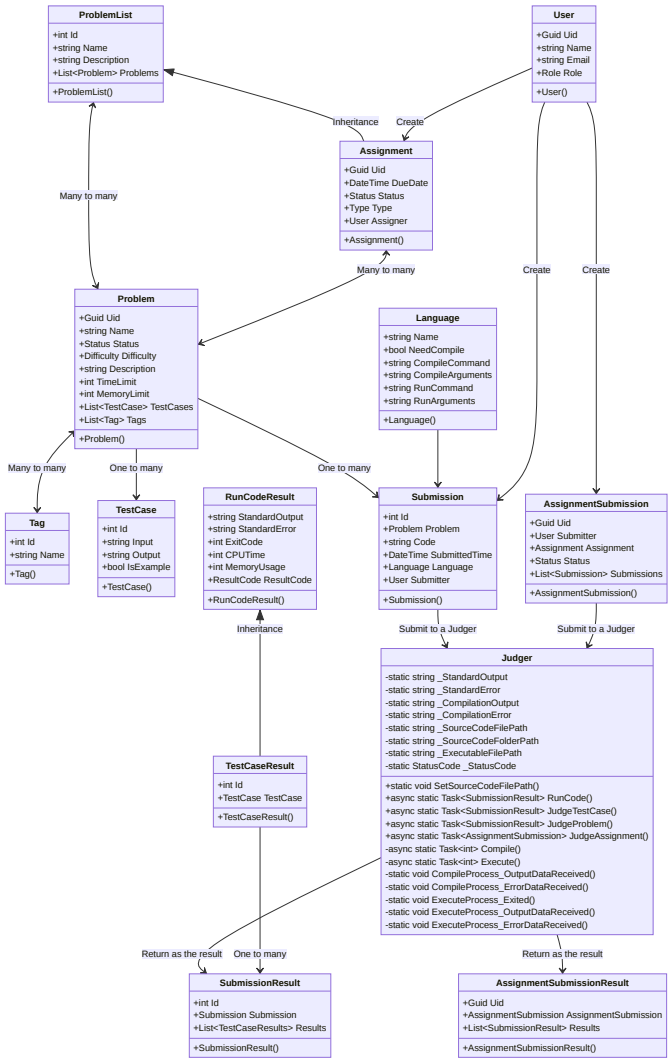
This algorithm is designed to judge an assignment submission. I set up a judging queue and push all problems into the queue. Then I take them out and judge them one by one. A queue is suitable for this use case because I want the first problem entered to be judged first. An assignment submission result will be returned at the end.

```
1   partial class Judger
2       public function JudgeAssignment(AssignmentSubmission)
3           // Create the assignment submission result
4           AssignmentSubmissionResult result = new
            ↪  AssignmentSubmissionResult()
5           result.AssignmentSubmission = AssignmentSubmission
6
7           // Create the judging queue
8           Queue JudgeQueue = new
            ↪  Queue(AssignmentSubmission.Submissions)
9
10          // Keep judging until the queue is empty
11          while len(JudgeQueue) > 0
12              Submission = JudgeQueue.Dequeue()
13              SubmissionResult = JudgeProblem(Submission)
14              result.Add(SubmissionResult)
15          endwhile
16          return result
17      endfunction
18  endclass
```

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.


ProblemList
+int Id
+string Name
+string Description
+List<Problem> Problems
+ProblemList()
User
+Guid Uid
+string Name
+string Email
+Role Role
+User()
Inheritance
Create
Assignment
+Guid Uid
+DateTime DueDate
+Status Status
+Type Type
+User Assigner
+Assignment()
Many to many
Many to many
Create
Create
Problem
+Guid Uid
+string Name
+Status Status
+Difficulty Difficulty
+string Description
+int TimeLimit
+int MemoryLimit
+List<TestCase> TestCases
+List<Tag> Tags
+Problem()
Language
+string Name
+bool NeedCompile
+string CompileCommand
+string CompileArguments
+string RunCommand
+string RunArguments
+Language()
Many to many
One to many
One to many
Tag
+int Id
+string Name
+Tag()
TestCase
+int Id
+string Input
+string Output
+bool IsExample
+TestCase()
RunCodeResult
+string StandardOutput
+string StandardError
+int ExitCode
+int CPUTime
+int MemoryUsage
+ResultCode ResultCode
+RunCodeResult()
Submission
+int Id
+Problem Problem
+string Code
+DateTime SubmittedTime
+Language Language
+User Submitter
+Submission()
AssignmentSubmission
+Guid Uid
+User Submitter
+Assignment Assignment
+Status Status
+List<Submission> Submissions
+AssignmentSubmission()
Inheritance
Submit to a Judger
Submit to a Judger
Judger
-static string _StandardOutput
-static string _StandardError
-static string _CompilationOutput
-static string _CompilationError
-static string _SourceCodeFilePath
-static string _SourceCodeFolderPath
-static string _ExecutableFilePath
-static StatusCode _StatusCode
+static void SetSourceCodeFilePath()
+async static Task<SubmissionResult> RunCode()
+async static Task<SubmissionResult> JudgeTestCase()
+async static Task<SubmissionResult> JudgeProblem()
+async static Task<AssignmentSubmission> JudgeAssignment()
-async static Task<int> Compile()
-async static Task<int> Execute()
-static void CompileProcess_OutputDataReceived()
-static void CompileProcess_ErrorDataReceived()
-static void ExecuteProcess_Exited()
-static void ExecuteProcess_OutputDataReceived()
-static void ExecuteProcess_ErrorDataReceived()
TestCaseResult
+int Id
+TestCase TestCase
+TestCaseResult()
Return as the result
One to many
Return as the result
SubmissionResult
+int Id
+Submission Submission
+List<TestCaseResults> Results
+SubmissionResult()
AssignmentSubmissionResult
+Guid Uid
+AssignmentSubmission AssignmentSubmission
+List<SubmissionResult> Results
+AssignmentSubmissionResult()

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 this information when they first open the software, an instance of User will be created and stored in 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 the 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 programming question.

Variable	Data type	Justification
`Uid`	`Guid`	The `Problem` uses a `Guid` value for its `Uid` instead of a normal `int` value to ensure the `Uid` is unique globally. So when students import an `Assignment` with multiple `Problems` into their database, the `Uid` will not conflict with any existing values, and it will not be changed.
`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 provdes 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 intrepreter.
`RunArguments`	`string`	This `Judger` pass this arguments to the executable or the intrepreter.

`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 formating 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` contains a `List<Problem>` variable, which 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`.
`Problems`	`List<Problem>`	`Problems` is a list of `Problem`, which supports basic functions to manipulate a list of data.

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

`Judger`

The `Judger` is a static class, only eposes 5 functions to run and judge user's code. The `Judger` takes a `Submission` as an input, and outputs a `SubmissionResult`. No variable is exposed. The private attributes and methods are discussed in the algorithm design section.

`RunCodeResult`

`RunCodeResult` stores the judging result for a piece of code. It does not store the information about whether the output is correct, it only stores the runtime statistics and whether the code has been compiled and executed correctly. This result will not be stored into the database, so there is no `Id` attribute.

Variable	Data type	Justification
`StandardOutput`	`string`	`StandardOutput` stores the content output by the user's code.
`StandardError`	`string`	`StandardError` stores the error messages received (if any) when executing the user's code.
`ExitCode`	`int`	`ExitCode` stores the exit code of the user's code.
`CPUTime`	`int`	`CPUTime` stores the time it takes for the user's code to execute.
`MemoryUsage`	`int`	`MemoryUsage` stores the memory usage record for the user's code.
`ResultCode`	`enum ResultCode`	`ResultCode` is a custom enumeration type storing the judging result. It has 7 possible values, `WRONG_ANSWER`, `SUCCESS`, `COMPILE_ERROR`, `TIME_LIMIT_EXCEEDED`, `MEMORY_LIMIT_EXCEEDED`, `RUNTIME_ERROR` or `SYSTEM_ERROR`.

`TestCaseResult`

`TestCaseResult` stores the judging result for each individual `TestCase`. It is inheriated from the `RunCodeResult` to inheriate all the statistics.

Variable	Data type	Justification
`Id`	`int`	The `Id` is the local unique identifier for a `TestCaseResult`. It is used to uniquely determine a `TestCaseResult` in the database.
`TestCase`	`TestCase`	`TestCase` stores the corresponding `TestCase` that produces this result. This field will be normalized into `TestCase.Id` before storing into the database.

`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 `AssignmentSubmissionResult` 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.
`SubmisionResults`	`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.

```
{
    "Theme": "Light",
    "RunCodeTimeLimit": 1000,
    "SelectedLanguage": "Python",
    "LanguageConfiguration":
    [
        {
            "Name": "C++",
            "NeedCompile": true,
            "CompileCommand": "g++",
            "CompileArguments": "-x c++ {SourceCodeFilePath} -o
            ↪  {ExecutableFilePath}",
            "RunCommand": "{ExecutableFilePath}",
            "RunArguments": ""
        },
        {
            "Name": "Python",
            "NeedCompile": false,
            "CompileCommand": "",
            "CompileArguments": "",
            "RunCommand": "python",
            "RunArguments": "{SourceCodeFilePath}"
        }
    ],
    "CurrentUser": "6ee2ebef-3f50-43b7-adf4-78c460339fd0"
}
```

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 applyed and write back to this field.

The `RunCodeTimeLimit` field stores the max time allowed for the run code function. The data will be validated before applying, if it is not a positive integer, then a default 1000 will be applied and written 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 written back to this field.

The `LanguageConfiguration` is a list of dictionaries 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 log in 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 template 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. The 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 `Uid` 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.

Uid	Name	Description	Status
<GUID>	“name”	“Problem description”	0

Difficulty	TimeLimit	MemoryLimit
0	1000	1000

```
CREATE TABLE Problem
(
    Uid TEXT PRIMARY KEY NOT NULL,
    Name TEXT NOT NULL,
    Description TEXT NOT NULL,
    Status INTEGER NOT NULL,
    Difficulty INTEGER NOT NULL,
    TimeLimit INTEGER NOT NULL,
    MemoryLimit INTEGER NOT NULL
)
```

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”

```
CREATE TABLE Tag
(
    Id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    Name TEXT NOT NULL UNIQUE
)
```

TagRecord 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	ProblemUid	TagId
1	<GUID>	1
2	<GUID>	2

```
CREATE TABLE TagRecord
(
    PRIMARY KEY(Id) INTEGER NOT NULL AUTO_INCREMENT,
    FOREIGN KEY (ProblemUid) REFERENCES Problem(Id),
    FOREIGN KEY (TagId) REFERENCES Tag(Id),
)
```

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 `ProblemUid` is enabled for the `TestCase` to store its corresponding `Problem`.

Id	Input	Output	IsExample	ProblemUid
1	“Input1”	“Output1”	1	<GUID>

```
CREATE TABLE TestCase
(
    PRIMARY KEY (Id) INTEGER NOT NULL AUTO_INCREMENT,
    Input TEXT NOT NULL,
    Output TEXT NOT NULL,
    IsExample INTEGER NOT NULL,
    FOREIGN KEY (ProblemUid) REFERENCES Problem(Uid)
)
```

Submission table

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

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	ProblemUid	Code	Time
1	<GUID>	“source code”	“11/11/2021 14:20:04“

Language	UserUid
“Python”	<GUID>

```
CREATE TABLE Submission
```

```
(
    PRIMARY KEY (Id) INTEGER NOT NULL AUTO_INCREMENT,
    FOREIGN KEY (ProblemUid) REFERENCES Problem(Uid),
    Code TEXT,
    Time TEXT NOT NULL,
    Language TEXT NOT NULL,
    FOREIGN KEY (UserUid) REFERENCES User(Uid),
)
```

ProblemList table

The ProblemList table will have an auto-increment integer `Id` as its primary key. The description is allowed to be empty for easy sharing.

Id	Name	Description
1	ProblemList 1	Description for ProblemList 1

```
CREATE TABLE Submission
(
    PRIMARY KEY (Id) INTEGER NOT NULL AUTO_INCREMENT,
    Name TEXT NOT NULL,
    Description TEXT
)
```

ProblemListRecord table

The `ProblemList` and `Problem` has a many-to-many relationship, so a ProblemListRecord table is used to normalize the data.

Id	ProblemUid	ProblemListId
1	`<Guid>`	1

Assignment table

AssignmentSubmission table

TestCaseResult table

SubmissionResult table

AssignmentSubmissionResult table

2.3.4 Import/Export design

All data will be exported into JSON files.

```
{
    "FileType": "Algorithm Dynamics Exported Data",
    "DataType": "Problem",
    "Data": ...
}
```

Apart from the raw data, two extra fields are added to help validate the JSON file. The `FileType` will always be `"Algorithm Dynamics Exported Data"`. This field will be validated first to make sure the correct file is imported. The `DataType` determines the type of the `Data` and helps the module to deserialize the data.

Import/Export a Problem

```
{
    "FileType": "Algorithm Dynamics Exported Data",
    "DataType": "Problem",
    "Data": {
        "Name": "Problem 1",
        "Difficulty": 2,
        "Description": "Problem description 1",
        "TimeLimit": 1000,
        "MemoryLimit": 1000,
        "TestCases": [
            {
                "Input": "3 4",
                "Output": "5",
                "IsExample": true
            },
            {
                "Input": "echo",
```

```
                "Output": "echo",
                "IsExample": false
            }
        ],
        "Tags": [
            {
                "Name": "Tag1"
            },
            {
                "Name": "Tag2"
            }
        ]
    }
}
```

Import/Export a ProblemList

```
{
    "FileType": "Algorithm Dynamics Exported Data",
    "DataType": "ProblemList",
    "Data": {
        "Name": "Problem List 1",
        "Description": "Problem list description 1",
        "Problems": [
            {
                "Name": "Problem 1",
                "Difficulty": 2,
                "Description": "Problem description 1",
                "TimeLimit": 1000,
                "MemoryLimit": 1000,
                "TestCases": [
                    {
                        "Input": "3 4",
                        "Output": "5",
                        "IsExample": true
                    },
                    {
                        "Input": "echo",
                        "Output": "echo",
                        "IsExample": false
                    }
                ],
                "Tags": [
                    {
                        "Name": "Tag1"
                    },
                    {
                        "Name": "Tag2"
```

```
                    }
                ]
            },
            {
                "Name": "Problem 2",
                "Difficulty": 0,
                "Description": "Problem description 2",
                "TimeLimit": 2000,
                "MemoryLimit": 2000,
                "TestCases": [
                    {
                        "Input": "",
                        "Output": "Hello world",
                        "IsExample": true
                    }
                ],
                "Tags": []
            }
        ],
        "Count": 2
    }
}
```

Import/Export an Assignment

```
{
    "FileType": "Algorithm Dynamics Exported Data",
    "DataType": "Assignment",
    "Data": {
        "Uid": "b527fe24-e872-4ef6-b1a7-7b78c9ebe110",
        "DueDate": "2021-11-19T00:00:00+00:00",
        "Status": 0,
        "Type": 0,
        "Assigner": {
            "Uid": "359a2d2b-d1c1-4169-a255-9289fcfff4e6",
            "Name": "User1",
            "Email": "user1@example.com",
            "Role": 1
        },
        "Name": "Assignment 1",
        "Description": "Assignmet description 1",
        "Problems": [
            {
                "Name": "Problem 1",
                "Difficulty": 2,
                "Description": "Problem description 1",
                "TimeLimit": 1000,
                "MemoryLimit": 1000,
```

```
                "TestCases": [
                    {
                        "Input": "3 4",
                        "Output": "5",
                        "IsExample": true
                    }
                ],
                "Tags": [
                    {
                        "Name": "Tag1"
                    },
                    {
                        "Name": "Tag2"
                    }
                ]
            },
            {
                "Name": "Problem 2",
                "Difficulty": 0,
                "Description": "Problem description 2",
                "TimeLimit": 2000,
                "MemoryLimit": 2000,
                "TestCases": [
                    {
                        "Input": "",
                        "Output": "Hello world",
                        "IsExample": true
                    }
                ],
                "Tags": []
            }
        ]
    }
}
```

Import/Export an AssignmentSubmission

Import/Export an AssignmentSubmissionResult

2.4 Development testing

2.4.1 Milestones

There are seven major milestones for the development of the solution.

1. Create the UI

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

2.4.2 Milestone 1: Create the UI

TODO: testing plan

The user interface is created first. At this stage, all buttons and layouts are created, but there is no code behind them. So I will do dry run tests to see whether the rendering, navigation and input box validation works correctly.

Test input	Expect output	Justification
Click different buttons on the NavigationView.	The main frame switches to the correct page.	A normal test to make sure the NavigationView is working correctly.
Click different buttons on the MainPage.	The main frame switches to different functions correctly.	A normal test to make sure the HomePage is working correctly.
Input a long string to the search box.	Only the first 32 characters are registered.	An erroneous test to make sure the length limit is working. Since at this stage, the searching algorithm is not implemented yet, the search result is not tested.
Select different items in the difficulty, list, tag dropdown boxes.	Items can be selected correctly.	A normal test to make sure the dropdown boxes are working.
Click the start button on the problem.	Navigate to the CodingPage correctly.	A normal test to make sure the start navigation is working correctly.
Input the Markdown test data below to the Problem section.	The Markdown text should be rendered and formatted correctly.	A normal test to see whether the Markdown text can be rendered correctly.
Input the Code test data below to the code editor.	The line number shows up correctly. The syntax highlighting works correctly. The shortcuts work correctly.	A normal test to make sure the code editor is working correctly.

Markdown test data

```
# Markdown Tests

## Headers

### heading 3
```

```
6
7   Under heading 3
8
9   #### heading 4
10
11  Under heading 4
12
13  ##### heading 5
14
15  Under heading 5
16
17  ###### heading 6
18
19  Under heading 6
20
21  ## Comment
22
23  <!-- You cannot see this comment -->
24
25  ## Horizontal Rules
26
27  ---
28
29  ___
30
31  ***
32
33  ## Emphasis
34
35  *This text will be italic*
36
37  _This will also be italic_
38
39  **This text will be bold**
40
41  __This will also be bold__
42
43  ~~Strike through this text.~~
44
45  _You **can** combine them_
46
47  ***bold and italics***
48
49  ~~**strikethrough and bold**~~
50
51  ~~*strikethrough and italics*~~
52
53  ~~***bold, italics and strikethrough***~~
54
55  ## Blockquotes
```

```
> Donec massa lacus, ultricies a ullamcorper in, fermentum sed augue.
Nunc augue augue, aliquam non hendrerit ac, commodo vel nisi.
>> Sed adipiscing elit vitae augue consectetur a gravida nunc vehicula. Donec auctor
odio non est accumsan facilisis. Aliquam id turpis in dolor tincidunt mollis ac eu diam.


## Lists

* Item 1
* Item 2
  * Item 2a
  * Item 2b

1. Item 1
1. Item 2
1. Item 3
   1. Item 3a
   1. Item 3b

- [x] Task 1
- [ ] Task 2
- [ ] Task 3

## Code

`Inline Code`

```cpp
#include <iostream>
int main()
{
    std::cout << "Hello world!" << endl;
    return 0;
}
```

Tables

Option	Description
data	path to data files to supply the data that will be passed into templates.
engine	engine to be used for processing templates. Handlebars is the default.
ext	extension to be used for dest files.

```

```
Links

<https://assemble.io>

<contact@revolunet.com>

[Assemble](https://assemble.io)

[Upstage](https://github.com/upstage/ "Visit Upstage!")

* [Chapter 1](#headers)
* [Chapter 2](#comment)
* [Chapter 3](#horizontal-rules)

Footnotes

This is a digital footnote[^1].
This is a footnote with "label"[^label]

[^1]: This is a digital footnote
[^label]: This is a footnote with "label"

Images

![Minion](https://octodex.github.com/images/minion.png)

![Alt text](https://octodex.github.com/images/stormtroopocat.jpg
↪ "The Stormtroopocat")
```

Code test data

```
#include <iostream>
using namespace std;
int main()
{
    int a, b;
    cin >> a >> b;
    cout << a + b << endl;
    return 0;
}
```

2.4.3 Milestone 2: Implement data structures

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: Create database

TODO: testing plan

Test input	Expect output	Justification

2.4.6 Milestone 5: Handle data import/export

TODO: testing plan

Test input	Expect output	Justification

2.4.7 Milestone 6: Handle settings

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 on 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

During the alpha testing, I will test the code myself. I will create a unit test project and write unit test cases to test the data structures, database, judger and data import and export automatically. The unit test will execute as a black-box test, creating some test input and testing whether the output matches. By automating the testing process, it will save me a lot of time hand inputting all

the test data and comparing the output, which improves the development speed and the code quality.

I will use the code coverage to measure whether the code is well tested. The code coverage calculates how many codes are executed during the unit test. A higher code coverage indicates the code is better tested.

Any failed test cases will be investigated and at the end of the alpha testing, all unit tests must be passed with a code coverage higher than 90%.

2.5.2 Beta testing

During the beta testing stage. I will invite my stakeholders to use the software. First, the deployment of the software is tested. I will test how difficult it is for my stakeholders to download and install the software on their computer, identify and fix any compatibility issues that occur during this process.

I will create some test coding problems for the students to import and practice on. During this process, the import function, the coding page, the judger and the user interface is tested. After that, I will invite the students to play with the software themselves and identify and fix any issues they meet.

I will guide the teacher to create test problems, problem lists and assignments to test the user interface and the database. Then export all the problems and assignments created to test the export function. I will provide some test assignment submissions for the teacher to test the assignment management function. After that, I will invite the teacher to play with the software himself and identify and fix any issues he meets.

In the end, I will conduct an integration test by inviting the teacher and the students to work on a real programming assignment using this software. All functions of the software are tested during this stage and any issues identified during this stage will be fixed.

Chapter 3

Development

Chapter 4

Evaluation

(TODO): Add evaluation chapter