A Level Programming Project Report

Junrong Chen

January 1, 2022

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 recognition' is needed which makes the design and programming much easier. The code editor will also provide a better user experience. Features such as syntax highlighting cannot exist on paper but are possible in a code editor.

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

1.4.2 Thinking ahead

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

1.4.3 Thinking procedurally and decomposition

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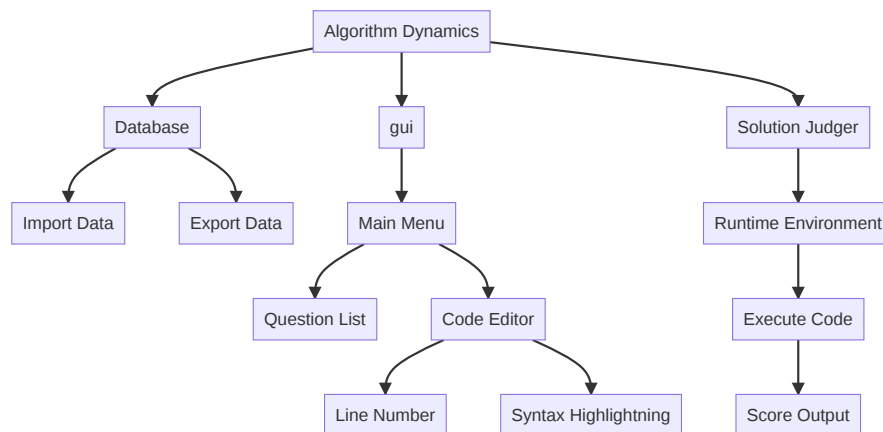
1.4.4 Thinking concurrently

When judging the students' solution, many test cases can be executed at the same time to reduce the judging time. The number of parallel 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
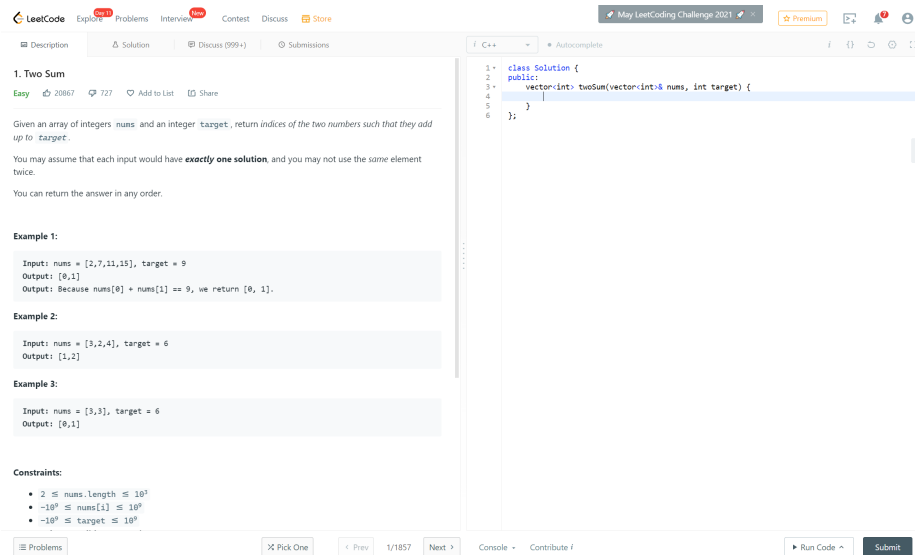
This is LeetCode's main coding area. The user's screen is split into two parts - the question section and the code editor for inputting answers. Users can drag the splitter in the middle to adjust the size of each section.

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

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

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

Pricing

The 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

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

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.

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.

```mermaid
graph LR
    AlgorithmDynamics["Algorithm Dynamics"] --> UserInterface["User Interface"]
    AlgorithmDynamics --> Judger["Judger"]
    AlgorithmDynamics --> Database["Database"]
    AlgorithmDynamics --> DataStructure["Data Structure"]
    AlgorithmDynamics --> Networking["Networking"]

    UserInterface --> NavigationView["NavigationView"]
    UserInterface --> HomePage["HomePage"]
    UserInterface --> ProblemsPage["ProblemsPage"]
    UserInterface --> AssignmentsPage["AssignmentsPage"]
    UserInterface --> PlaygroundPage["PlaygroundPage"]
    UserInterface --> CodingPage["CodingPage"]
    UserInterface --> AccountPage["AccountPage"]
    UserInterface --> SettingsPage["SettingsPage"]

    ProblemsPage --> CreateNewProblemPage["CreateNewProblemPage"]
    ProblemsPage --> CreateNewProblemListPage["CreateNewProblemListPage"]
    ProblemsPage --> SearchBox["SearchBox"]
    ProblemsPage --> ImportExportDialog["ImportExportDialog"]

    SearchBox --> DatabaseInteration["DatabaseInteration"]

    AssignmentsPage --> AssignmentsDetailPage["AssignmentsDetailPage"]
    AssignmentsPage --> AssignmentsListPage["AssignmentsListPage"]
    AssignmentsPage --> CreateNewAssignmentPage["CreateNewAssignmentPage"]

    AssignmentsDetailPage --> TeacherView["TeacherView"]
    AssignmentsDetailPage --> StudentView["StudentView"]

    CodingPage --> ProblemDisplay["ProblemDisplay"]
    CodingPage --> CodeEditor["CodeEditor"]
    CodingPage --> IntputOutputErrorBox["IntputOutputErrorBox"]
    CodingPage --> ProblemNavigator["ProblemNavigator"]

    AccountPage --> AvatarDisplay["AvatarDisplay"]
    AccountPage --> LoginButton["LoginButton"]
    AccountPage --> EditButton["EditButton"]
    AccountPage --> StatsDisplay["StatsDisplay"]

    SettingsPage --> ThemeSelector["ThemeSelector"]

    Judger --> RunCode["RunCode"]
    Judger --> JudgeProblem["JudgeProblem"]
    Judger --> JudgeAssignment["JudgeAssignment"]
    Judger --> ProgrammingLanguageSelector["ProgrammingLanguageSelector"]

    Database --> AddData["AddData"]
    Database --> SearchData["SearchData"]
    Database --> EditData["EditData"]
    Database --> DeleteData["DeleteData"]

    DeleteData --> ProblemList["ProblemList"]
    ProblemList --> Problem["Problem"]

    Problem --> Tag["Tag"]
    Problem --> TestCase["TestCase"]

    DataStructure --> Assignment["Assignment"]
    Assignment --> AssignmentSubmission["AssignmentSubmission"]

    AssignmentSubmission --> AssignmentSubmissionResult["AssignmentSubmissionResult"]
    AssignmentSubmission --> Submission["Submission"]

    AssignmentSubmissionResult --> SubmissionResult["SubmissionResult"]

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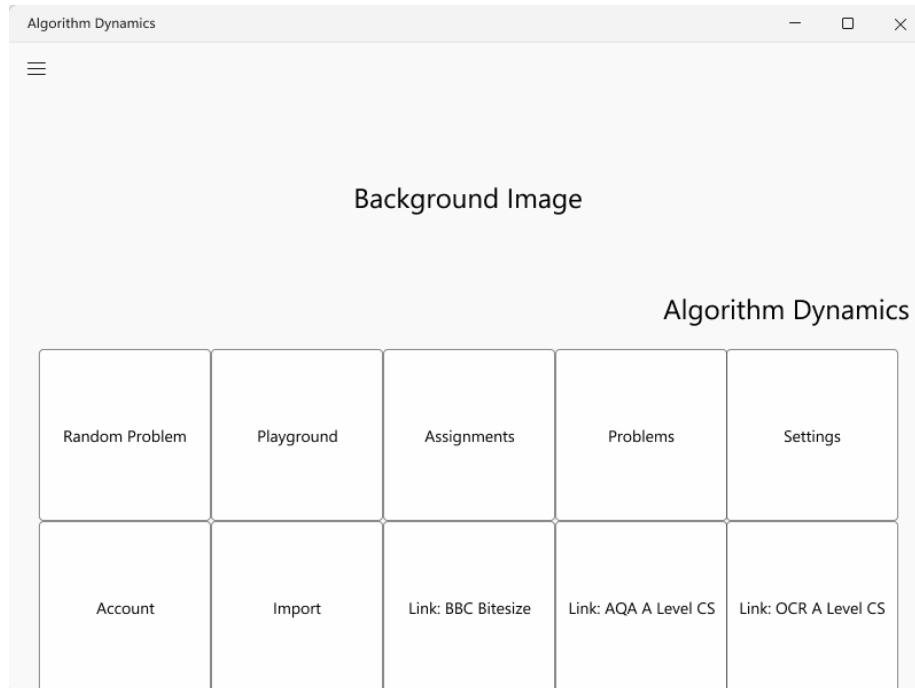
The HomePage is the first page that gets displayed to the user. On the top of the HomePage, there will be a beautiful background image under the 'Algorithm Dynamics' title. On the bottom, there are 10 useful buttons link to different functions.

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 found 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.

Mr Grimwood says it will be great if some greeting messages with recommendations can be displayed, such as "You have 6 assignments to do this week." or "You did pretty bad on data structures problems last time, maybe try this problem list to help you revise better."

Response

Timofei has a good point here, and I decide to remove these links. The UI will be responsive to different window sizes, and this will be tested.

The greeting message and recommendations is a great idea. I will add an additional row below the image to display the greeting and recommendations.
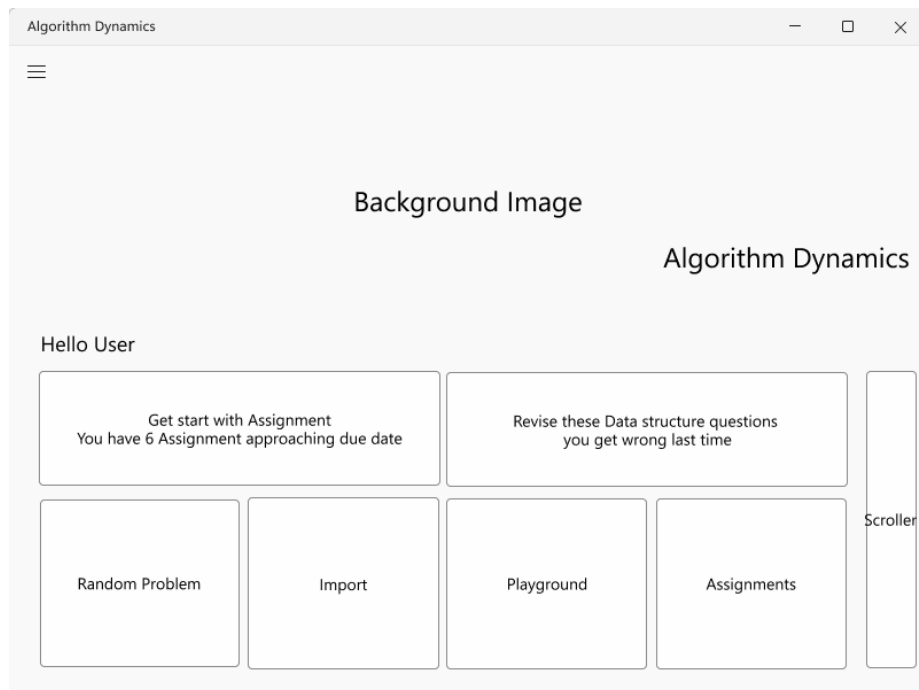
This is the redesigned HomePage. It adds greeting messages and recommendations. It has a responsive grid and a scroller is displayed when the item does not fit in one page. The useless links are removed.

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
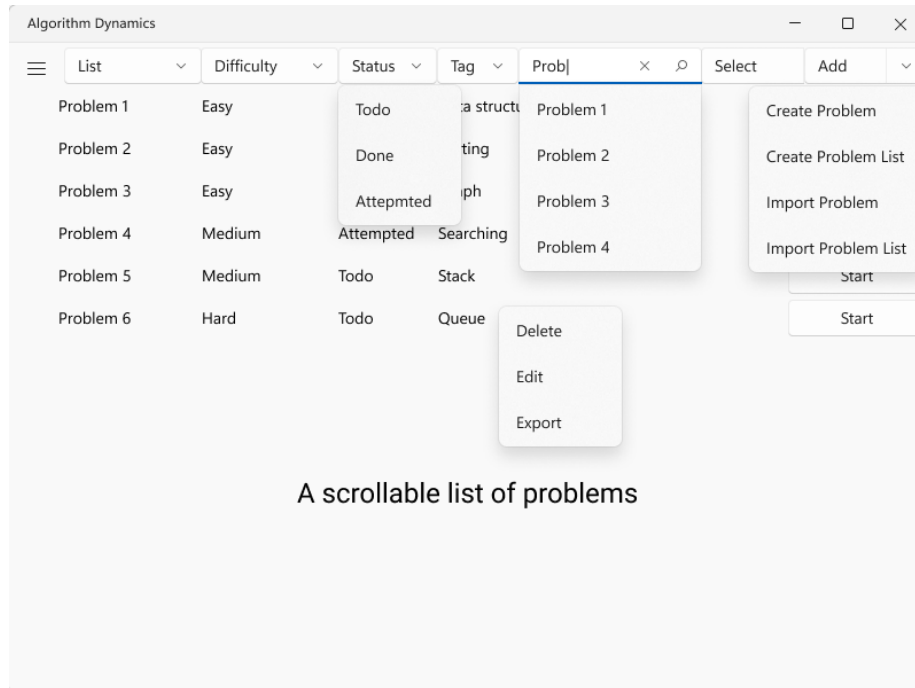
The user can apply the selection condition by either selecting different fields in the dropdown boxes or directly typing into the search box. A scrollable list of problems that match the conditions will be shown below with detailed information. The user can start working on a problem by clicking the 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
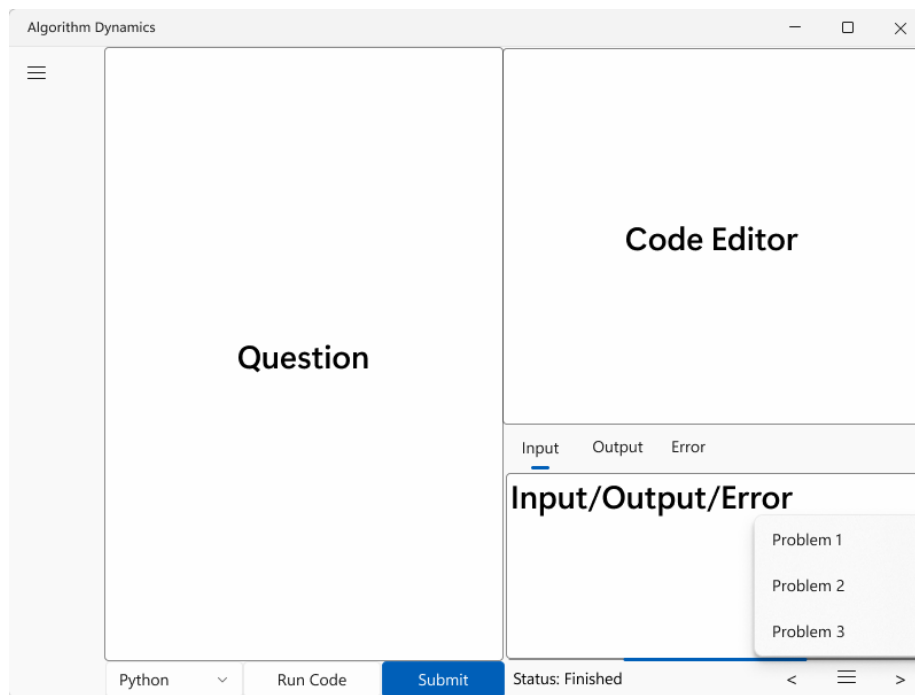
The CodingPage is where the user works on a programming problem. It contains a question panel, a code editor and an IO panel.

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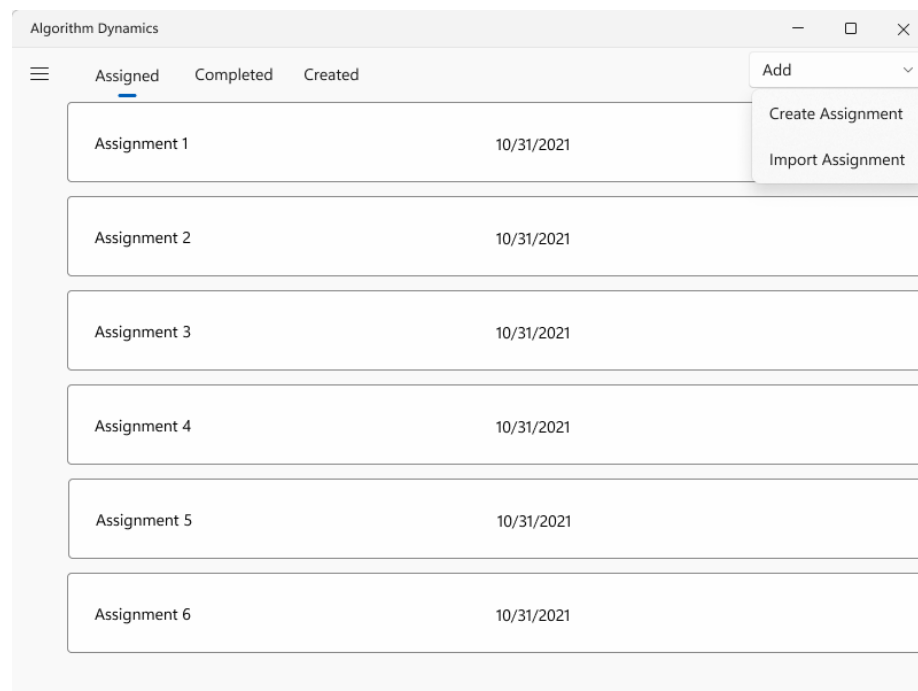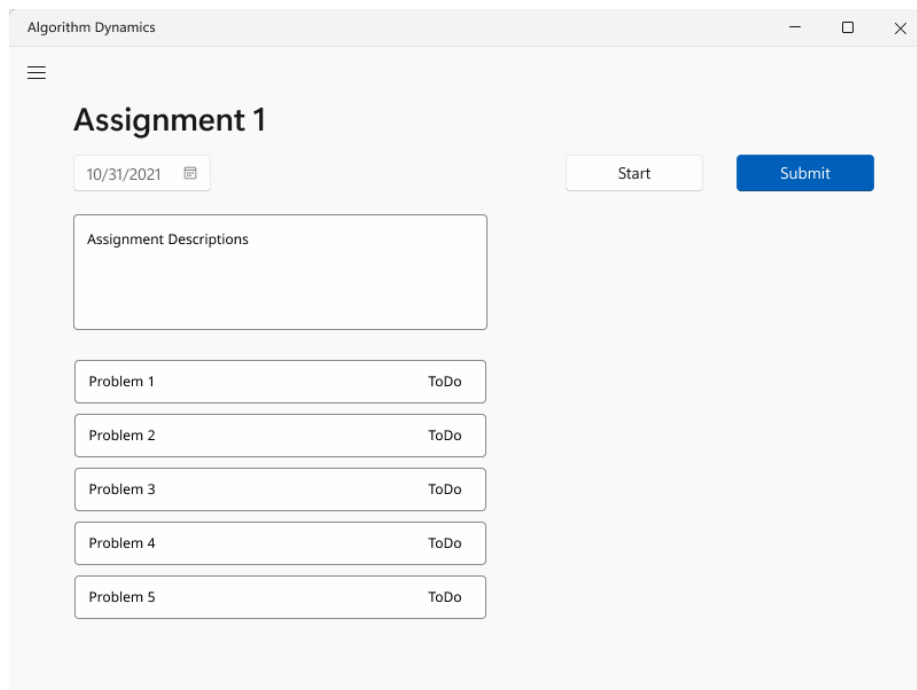
The AssignmentPage is where the user interacts with the assignments. For students, they can work on their assignments; for teachers, they can create and mark assignments.

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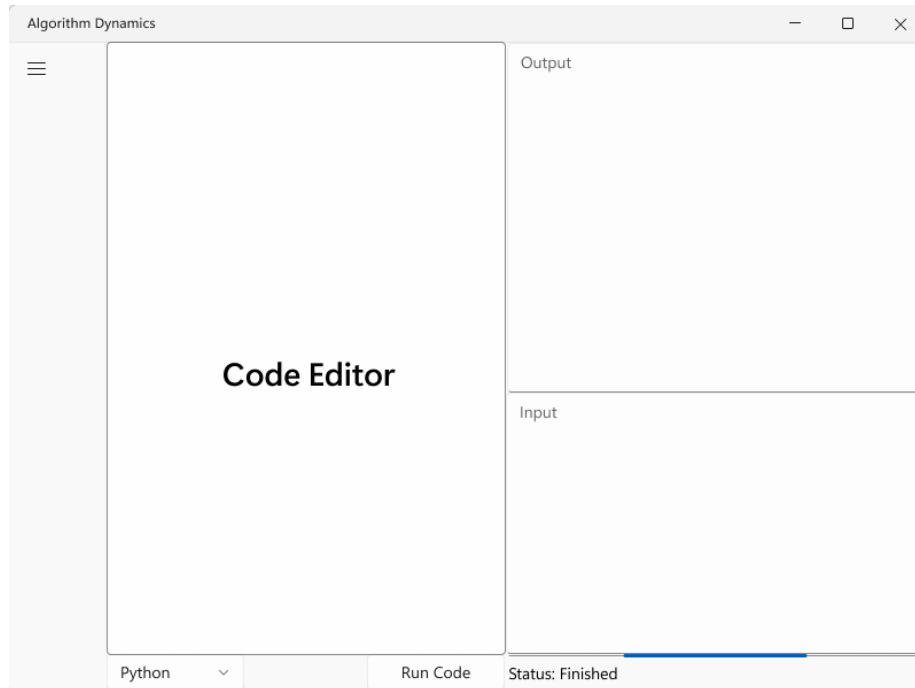
The PlaygroundPage is a free space for the user to run any code quickly. It is essentially the same page as the CodingPage but without a coding problem. The interface is slightly changed with a larger code editor and a separate IO panel for a better user experience.

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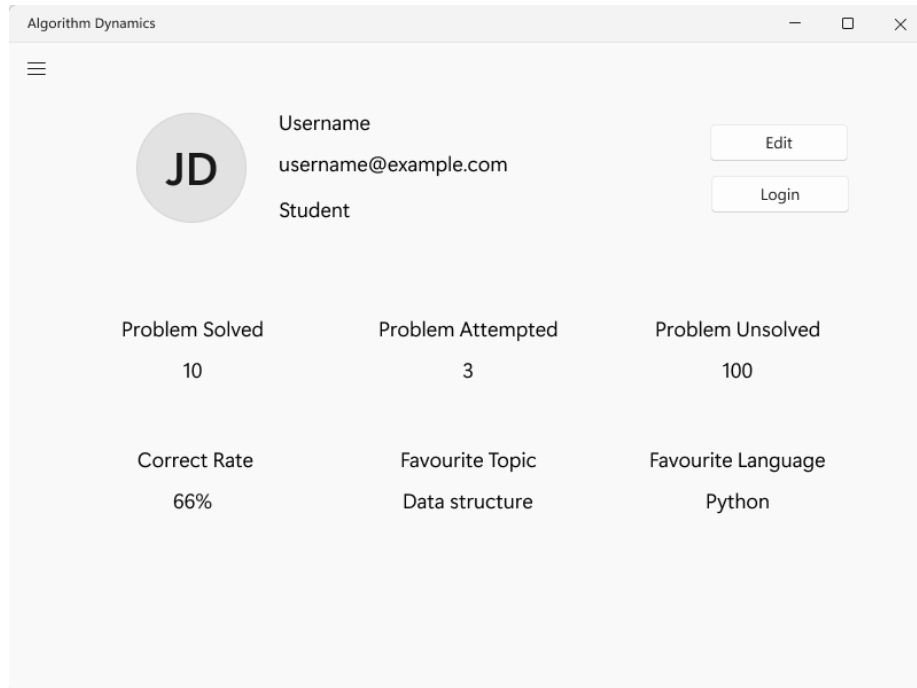
The AccountPage is used to manage the current user and display user statistics.

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 spends in the software.

Mr Grimwood suggests that "topic needs more revision" can be added to the statistics.

Response

It may be too difficult to implement such statistics but this is an interesting idea. This feature is put in the backlog and will not be implemented for now.

"Topic needs more revision" will be put as a recommendation, on the HomePage instead, which will make it more useful.

2.1.8 SettingsPage
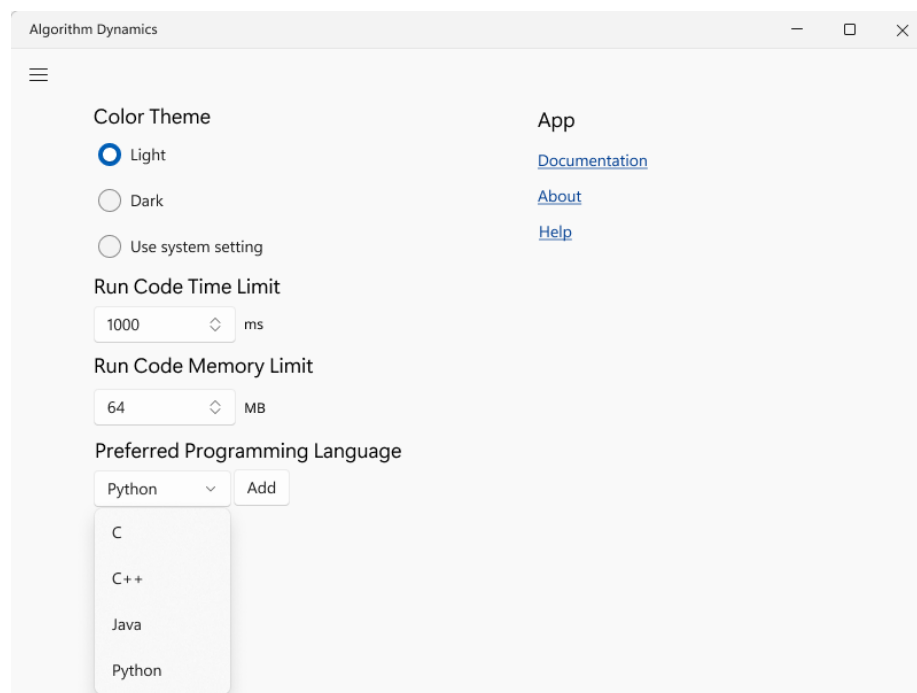
The SettingsPage is where the user adjusts software settings.

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, memory 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 and memory 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 themes. He also suggests that allowing the user to change the font size manually may help a lot.

Response

Extra themes take too much time to design, but it is an interesting idea. The font size is managed by the system and it is not very easy to change. Both ideas are great, they are put into the backlog and will not be implemented right away.

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.

2.1.15 Stakeholder feedback

I conduct an interview with each stakeholder after designing all the user interface elements. The statkeholders are generally very satisfying with the design. Suggestions are mainly about the HomePage design and the AccountPage design, I have recorded their suggestions and my responses in those sections.

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.

```
1  function search(sourceList, keyword)
2      // Create an empty list to store the results
3      resultList = new List<string>()
4      // Split the keyword into pieces by space
5      splitKeyword = keyword.ToLower().Split(' ')
6      // Compare each piece of keyword with the sourceList
7      // Add the matching result into resultList
8      for i=0 to sourceList.Length - 1
9          for j=0 to splitKeyword.Length - 1
10             sourceKey = sourceList[i].ToLower()
```

```
11              if sourceKey.Contains(splitKeyword[j]) then
12                  resultList.Add(sourceList[i])
13              endif
14          next j
15      next i
16
17      // If no result is found,
18      // add an "not found" notice to the resultList
19      if resultList.Length == 0 then
20          resultList.Add("No results found")
21      endif
22  endfunction
```

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

Next, I use two nested loops to perform 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
```

44

```
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.

```
1   enum StatusCode
2       PENDING,
3       RUNNING,
4       FINISHED,
5       TIME_LIMIT_EXCEEDED,
6       MEMORY_LIMIT_EXCEED
7   endenum
8
9   partial class Judger
10      private _StatusCode
11  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.

```
1   partial class Judger
2       private _StandardOutput
3       private _StandardError
4
5       private procedure ExecuteProcess_Exited(sender, e)
6           // Only successfully finished when the status code is not
            ↪ TLE or MLE
7           if _StatusCode != StatusCode.TIME_LIMIT_EXCEEDED and
            ↪ _StatusCode != StatusCode.MEMORY_LIMIT_EXCEED then
8               _StatusCode = StatusCode.FINISHED
9           endif
10      endprocedure
11
12      private procedure ExecuteProcess_OutputDataReceived(sender,
        ↪ e)
13          // Validate the data before storing
14          if string.IsNullOrEmpty(e.Data) == False then
15              _StandardOutput += e.Data + '\n'
16          endif
17      endprocedure
```

```
18
19    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 ex-
pected output, so it can judge whether the user's code works correctly. The
`JudgeTestCase` function returns a `TestCaseResult` containing more informa-
tion.

```
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

Problem

+int Id
+string Name
+Status Status
+Difficulty Difficulty
+string Description
+int TimeLimit
+int MemoryLimit
+List<TestCase> TestCases
+List<Tag> Tags
+string SourceHash

+Problem()

Language

+string Name
+bool NeedCompile
+string CompileCommand
+string CompileArguments
+string RunCommand
+string RunArguments

+Language()

Create   Create

Many 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()

One to many

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

TestCaseResult

+int Id
+TestCase TestCase

+TestCaseResult()

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()

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
Id	int	The `Id` is the local unique identifier for a `Problem`. It is used to uniquely determine a `Problem` in the database. When a `Problem` is imported in to the database, a new `Id` is given.
Name	string	The `Name` is a string contains the name of a `Problem`.
SourceHash	string	This field is `null` for all normal `Problem`. It is only set for `Problem` which comes with an `Assignment`, and it is set to the hash value of the original `Problem`. This value will be recorded for `Submission`.

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 in 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.

```
1  {
2      "Theme": "Light",
3      "RunCodeTimeLimit": 1000,
4      "RunCodeMemoryLimit": 67108864,
5      "SelectedLanguage": "Python",
6      "LanguageConfiguration":
7      [
8          {
9              "Name": "C++",
10             "NeedCompile": true,
11             "CompileCommand": "g++",
12             "CompileArguments": "-x c++ {SourceCodeFilePath} -o
   ↪    {ExecutableFilePath}",
13             "RunCommand": "{ExecutableFilePath}",
14             "RunArguments": ""
15         },
16         {
17             "Name": "Python",
18             "NeedCompile": false,
19             "CompileCommand": "",
20             "CompileArguments": "",
21             "RunCommand": "python",
22             "RunArguments": "{SourceCodeFilePath}"
23         }
24     ],
25     "CurrentUser": "6ee2ebef-3f50-43b7-adf4-78c460339fd0"
26 }
```

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 `RunCodeMemoryLimit` field stores the max memory allowed for the run code function. The data will be validated before applying, if it is not a positive integer, then a default 67108864, which is 64MB 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 `Id` as its primary key. Both `Status` and `Difficulty` will be stored in integer for the same reason before.

All fields will be validated and only the `SourceHash` is allowed to be null as explained in the data structure design section.

Uid	Name	Description	Status
`<GUID>`	"name"	"Problem description"	0

Difficulty	TimeLimit	MemoryLimit	SourceHash
0	1000	1000	null

```sql
1  CREATE TABLE Problem
2  (
3      Uid TEXT PRIMARY KEY NOT NULL,
4      Name TEXT NOT NULL,
5      Description TEXT NOT NULL,
6      Status INTEGER NOT NULL,
7      Difficulty INTEGER NOT NULL,
8      TimeLimit INTEGER NOT NULL,
9      MemoryLimit INTEGER NOT NULL,
10     SourceHash TEXT
11 )
```

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"

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

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	ProblemId	TagId
1	1	1
2	1	2

```sql
1  CREATE TABLE TagRecord
2  (
```

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

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.

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

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

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

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

Submission table

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

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 relationship with a `User`. A `User` can create multiple `Submission` while a `Submission` can only be created by one `User`. It also has a many-to-one relationship with a `AssignmentSubmission`. So it will have three foreign keys to establish such relationships.

All fields will be validated and no empty field is allowed, except for the Code, empty `Submission` is allowed and for the AssignmentSubmissionResultUid, if the `Submission` does not belong to an `Assignment`, a null value is allowed.

Id	ProblemId	Code	Time
1	1	"source code"	"11/11/2021 14:20:04"

Language	UserUid	AssignmentSubmissionUid
"Python"	`<GUID>`	`<GUID>`

```
1  CREATE TABLE Submission
2  (
3      PRIMARY KEY (Id) INTEGER NOT NULL AUTO_INCREMENT,
4      FOREIGN KEY (ProblemId) REFERENCES Problem(Id),
5      Code TEXT,
6      Time TEXT NOT NULL,
7      Language TEXT NOT NULL,
8      FOREIGN KEY (UserUid) REFERENCES User(Uid),
9      FOREIGN KEY (AssignmentSubmissionResultUid) REFERENCES
       ↪  AssignmentSubmission(Uid),
10 )
```

ProblemList table

The ProblemList table will have an auto-increment `Id` as its primary key.

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

Id	Name	Description
1	"ProblemList 1"	"Description for ProblemList 1"

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

ProblemListRecord table

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

Id	ProblemId	ProblemListId
1	1	1

```
1  CREATE TABLE ProblemListRecord
2  (
3      PRIMARY KEY (Id) INTEGER NOT NULL AUTO_INCREMENT,
4      FOREIGN KEY (ProblemId) REFERENCES Problem(Id),
```

```
5        FOREIGN KEY (ProblemListId) REFERENCES ProblemList(Id)
6    )
```

Assignment table

The Assignment table will have `Uid` as its primary key. It has a one-to-many relationship with a User, so a `UserUid` field is used to record this relationship.

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

Uid	DueDate	Status	Type	UserUid
<GUID>	"11/11/2021 14:20:04"	1	0	<GUID>

```
1    CREATE TABLE Assignment
2    (
3        PRIMARY KEY (Uid) TEXT NOT NULL,
4        DueDate TEXT NOT NULL,
5        Status INTEGER NOT NULL,
6        Type INTEGER NOT NULL,
7        FOREIGN KEY (UserUid) REFERENCES User(Uid)
8    )
```

AssignmentRecord table

`Assignment` has a many-to-many relationship with the `Problem` so an AssignmentRecord table is used to normalize the data.

Id	AssignmentUid	ProblemId	SourceHash
1	<GUID>	1	<Hash>

```
1    CREATE TABLE AssignmentRecord
2    (
3        PRIMARY KEY (Id) INTEGER NOT NULL AUTO_INCREMENT,
4        FOREIGN KEY (AssignmentUid) REFERENCES Assignment(Uid),
5        FOREIGN KEY (ProblemId) REFERENCES Problem(Id),
6        SourceHash TEXT NOT NULL
7    )
```

AssignmentSubmission table

The AssignmentSubmission table will have `Uid` as its primary key. It has a many-to-one relationship with an Assignment, so a `AssignmentUid` field is used to record this relationship. It has a many-to-one relationship with a User, so a `UserUid` field is used to record this relationship.

Uid	AssignmentUid	UserUid	Status
\<GUID\>	\<GUID\>	\<GUID\>	0

```
1  CREATE TABLE AssignmentSubmission
2  (
3      Uid TEXT PRIMARY KEY NOT NULL,
4      FOREIGN KEY (AssignmentUid) REFERENCES Assignment(Uid),
5      FOREIGN KEY (UserUid) REFERENCES User(Uid),
6      Status INTEGER NOT NULL
7  )
```

TestCaseResult table

The TestCaseResult table will have an auto-increment `Id` as its primary key.

It has a many-to-one relationship with a TestCase, so a `TestCaseId` field is used to record this relationship. It has a many-to-one relationship with a SubmissionResult, so a `SubmissionResultId` field is used to record this relationship.

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

Id	TestCaseId	StandardOutput	StandardError	ExitCode
1	1	"Standard output"	"Standard error"	0

CPUTime	MemoryUsage	ResultCode	SubmissionResultId
1000	1000	0	1

```
1   CREATE TABLE TestCaseResult
2   (
3       PRIMARY KEY (Id) INTEGER NOT NULL AUTO_INCREMENT,
4       FOREIGN KEY (TestCaseId) REFERENCES TestCase(Id),
5       StandardOutput TEXT NOT NULL,
6       StandardError TEXT NOT NULL,
7       ExitCode INTEGER NOT NULL,
8       CPUTime INTEGER NOT NULL,
9       MemoryUsage INTEGER NOT NULL,
10      ResultCode INTEGER NOT NULL,
11      FOREIGN KEY (SubmissionResultId) REFERENCES
        ↪   SubmissionResult(Id),
12  )
```

SubmissionResult table

The SubmissionResult table will have an auto-increment `Id` as its primary key.

It has a one-to-one relationship with a Submission, so a `SubmissionId` field is used to record this relationship. It may have a many-to-one relationship with

an AssignmentSubmissionResult, so an AssignmentSubmissionResultUid field is used to record this relationship.

Id	SubmissionId	AssignmentSubmissionResultUid

```
1  CREATE TABLE SubmissionResult
2  (
3      PRIMARY KEY (Id) INTEGER NOT NULL AUTO_INCREMENT,
4      FOREIGN KEY (SubmissionId) REFERENCES Submission(Id),
5      FOREIGN KEY (AssignmentSubmissionResultUid) REFERENCES
       ↪  AssignmentSubmissionResult(Uid)
6  )
```

AssignmentSubmissionResult table

The AssignmentSubmission table will have `Uid` as its primary key. It has a one-to-one relationship with a AssignmentSubmission, so a `AssignmentSubmissionId` field is used to record this relationship.

Uid	AssignmentSubmissionUid
<GUID>	<GUID>

```
1  CREATE TABLE AssignmentSubmissionResult
2  (
3      Uid TEXT PRIMARY KEY NOT NULL,
4      FOREIGN KEY (AssignmentSubmissionUid) REFERENCES
       ↪  AssignmentSubmission(Uid)
5  )
```

2.3.4 Import/Export design

All data will be exported into JSON files.

```
1  {
2      "FileType": "Algorithm Dynamics Exported Data",
3      "DataType": "Problem",
4      "Data": ...
5  }
```

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

When the user exports a problem, the problem instance will be serialized. The test cases and tags associated with the problem will be converted into a list, and their Ids will be removed.

This is a sample exported file.

```
1  {
2      "FileType": "Algorithm Dynamics Exported Data",
3      "DataType": "Problem",
4      "Data": {
5          "Name": "Problem 1",
6          "SourceHash": null,
7          "Difficulty": 2,
8          "Description": "Problem description 1",
9          "TimeLimit": 1000,
10         "MemoryLimit": 1000,
11         "TestCases": [
12             {
13                 "Input": "3 4",
14                 "Output": "5",
15                 "IsExample": true
16             },
17             {
18                 "Input": "echo",
19                 "Output": "echo",
20                 "IsExample": false
21             }
22         ],
23         "Tags": [
24             {
25                 "Name": "Tag1"
26             },
27             {
28                 "Name": "Tag2"
29             }
30         ]
31     }
32  }
```

When the problem is imported, a new Id will be created for it.

The problem and test cases are imported first.

```
1  INSERT OR REPLACE INTO Problem VALUES (Name, Difficulty,
   ↪  Description, TimeLimit, MemoryLimit, SourceHash)
2  INSERT OR REPLACE INTO TestCase VALUES (Input, Output, IsExample,
   ↪  last_insert_rowid)
```

After that, the tags are imported and the corresponding TagRecord table is
updated.

```sql
INSERT OR REPLACE INTO Tag VALUES (Name)
INSERT INTO TagRecord VALUES (Problem.last_insert_rowid,
    Tag.last_insert_rowid)
```

Import/Export a ProblemList

When the user exports a problem list, several problems are wrapped into a list
and additional information is added. The Id of the original problem list is not
included in the exported file, a new Id will be assigned when the list is imported.

This is a sample exported file.

```json
{
    "FileType": "Algorithm Dynamics Exported Data",
    "DataType": "ProblemList",
    "Data": {
        "Name": "Problem List 1",
        "Description": "Problem list description 1",
        "Problems": [
            {
                "Name": "Problem 1",
                "SourceHash": null,
                "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"
```

```
33                   }
34                 ]
35             },
36             {
37                 "Name": "Problem 2",
38                 "SourceHash": null,
39                 "Difficulty": 0,
40                 "Description": "Problem description 2",
41                 "TimeLimit": 2000,
42                 "MemoryLimit": 2000,
43                 "TestCases": [
44                     {
45                         "Input": "",
46                         "Output": "Hello world",
47                         "IsExample": true
48                     }
49                 ],
50                 "Tags": []
51             }
52         ],
53         "Count": 2
54     }
55 }
```

When a problem list is imported, all problems are imported as individual problems. Then the information about the problem list itself is imported and the ProblemListRecord table is updated.

```
1 INSERT OR REPLACE INTO ProblemList VALUES (Name, Description)
2 INSERT OR REPLACE INTO ProblemListRecord VALUES
  ↪ (Problem.last_insert_rowid, ProblemList.last_insert_rowid)
```

Import/Export an Assignment

When an assignment is exported, a SourceHash value is computed and attached to each problem, all test cases that are not examples are removed. Data about the assigner, due date, status and type are also attached to the exported file.

This is a sample exported file.

```
1 {
2     "FileType": "Algorithm Dynamics Exported Data",
3     "DataType": "Assignment",
4     "Data": {
5         "Uid": "b527fe24-e872-4ef6-b1a7-7b78c9ebe110",
6         "DueDate": "2021-11-19T00:00:00+00:00",
7         "Status": 0,
```

```
 8          "Type": 0,
 9          "Assigner": {
10              "Uid": "359a2d2b-d1c1-4169-a255-9289fcfff4e6",
11              "Name": "User1",
12              "Email": "user1@example.com",
13              "Role": 1
14          },
15          "Name": "Assignment 1",
16          "Description": "Assignmet description 1",
17          "Problems": [
18              {
19                  "Name": "Problem 1",
20                  "SourceHash": "DFEFB848F705D65C7A861D36BD358F3F",
21                  "Difficulty": 2,
22                  "Description": "Problem description 1",
23                  "TimeLimit": 1000,
24                  "MemoryLimit": 1000,
25                  "TestCases": [
26                      {
27                          "Input": "3 4",
28                          "Output": "5",
29                          "IsExample": true
30                      }
31                  ],
32                  "Tags": [
33                      {
34                          "Name": "Tag1"
35                      },
36                      {
37                          "Name": "Tag2"
38                      }
39                  ]
40              },
41              {
42                  "Name": "Problem 2",
43                  "SourceHash": "F9B9B7578BFA6C949A146821C8F78DD3",
44                  "Difficulty": 0,
45                  "Description": "Problem description 2",
46                  "TimeLimit": 2000,
47                  "MemoryLimit": 2000,
48                  "TestCases": [
49                      {
50                          "Input": "",
51                          "Output": "Hello world",
52                          "IsExample": true
53                      }
54                  ],
55                  "Tags": []
56              }
57          ]
```

```
58        }
59    }
```

When an assignment gets imported, the assigner is first added to the User table.

```
1  INSERT OR REPLACE INTO User VALUES (Uid, Email, Name, Role)
```

Next, the record of the assignment is inserted into the Assignment table.

```
1  INSERT OR REPLACE INTO Assignment VALUES (Uid, DueDate, Status,
↪    Type, UserUid)
```

Finally, all the problems are imported as individual problems and the Problem-ListRecord table is updated.

```
1  INSERT OR REPLACE INTO ProblemListRecord VALUES (Uid,
↪    Problem.last_insert_rowid, SourceHash)
```

Import/Export an AssignmentSubmission

When an AssignmentSubmission is exported, the Assignment field will be normalized and only the Uid field will be kept. The Problem field will be normalized and only the SourceHash will be kept. The Language field will be normalized and only the language name will be kept. So the exported file will not contain any redundant information and at have an optimal size.

This is a sample exported file.

```
1  {
2      "FileType": "Algorithm Dynamics Exported Data",
3      "DataType": "AssignmentSubmission",
4      "Data": {
5          "Uid": "5872e4f1-731d-4e49-ab28-37971827fabe",
6          "Submitter": {
7              "Uid": "f84b5ec9-6156-4278-a9b1-d097e24409c1",
8              "Name": "student1",
9              "Email": "student1@example.com",
10             "Role": 0
11         },
12         "Assignment": {
13             "Uid": "977255d2-523b-4538-a472-872109bd1fc2",
14         },
15         "Status": 0,
16         "Submissions": [
```

```
17              {
18                  "Problem": {
19                      "SourceHash":
                       ↪   "DFEFB848F705D65C7A861D36BD358F3F"
20                  },
21                  "Code": "print(7)",
22                  "SubmittedTime":
                   ↪   "2021-11-20T22:21:07.0812049+00:00",
23                  "Language": {
24                      "Name": "Python",
25                  }
26              }
27          ]
28      }
29  }
```

When the AssignmentSubmission is imported into the database, the Submitter will be imported first.

```
1  INSERT OR REPLACE INTO User VALUES (Uid, Email, Name, Role)
```

Next, the problem Id is determined using the SourceHash, and the list of submissions will be imported to the Submission table.

```
1  INSERT INTO Submission VALUES (ProblemId, Code, Time, Language,
   ↪   UserUid, AssignmentUid)
```

In the end, the record in the AssignmentSubmission table will be updated.

```
1  INSERT INTO AssignmentSubmission VALUES (Uid, AssignmentUid,
   ↪   UserUid, Status)
```

Import/Export an AssignmentSubmissionResult

When an AssignmentSubmissionResult is exported, the AssignmentSubmission field is normalized so only the Uid is left. For each SubmissionResult, the SourceHash of the Problem is kept to determine the Submission.

This is a sample exported file.

```
1  {
2      "FileType": "Algorithm Dynamics Exported Data",
3      "DataType": "AssignmentSubmissionResult",
4      "Data": {
```

```
 5          "Uid": "8593d2cd-1b0e-4b26-aed0-37d4dc54a5a2",
 6          "AssignmentSubmission": {
 7              "Uid": "a2a9062c-18eb-48ad-bee8-ccb37f208bf5"
 8          },
 9          "Results": [
10              {
11                  "Submission": {
12                      "Problem": {
13                          "SourceHash":
                              ↪   "DFEFB848F705D65C7A861D36BD358F3F"
14                      }
15                  },
16                  "Results": [
17                      {
18                          "StandardOutput": "7",
19                          "StandardError": "",
20                          "ExitCode": 0,
21                          "CPUTime": 1000,
22                          "MemoryUsage": 1000,
23                          "ResultCode": 1
24                      }
25                  ]
26              }
27          ]
28      }
29 }
```

When the AssignmentSubmissionResult is imported into the database, the AssignmentSubmissionResult table is updated first.

```
1 INSERT INTO AssignmentSubmissionResult VALUES (Uid,
  ↪  AssignmentSubmissionResultUid)
```

Next, each SubmissionResult is imported into the SubmissionResult table. The SubmissionId can be determined by the SourceHash.

```
1 INSERT INTO SubmissionResult VALUES (SubmissionId,
  ↪  AssignmentSubmissionResultUid)
```

In the end, the TestCaseResult table is updated. The TestCaseId will be left null.

```
1 INSERT INTO TestCaseResult VALUES (null, StandardOutput,
  ↪  StandardError, ExitCode, CPUTime, MemoryUsage, ResultCode,
  ↪  SubmissionId)
```

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

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.

The SettingsPage will be tested after the settings module is implemented. The Create pages will be tested after the Database is implemented. The Account-Page will be tested after the API handler is implemented.

Test input	Expect output	Justification
Click different buttons on the NavigationView.	The mainframe switches to the correct page.	A normal test to make sure the NavigationView is working correctly.
Click different buttons on the MainPage.	The mainframe 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

```
1   # Markdown Tests
2
3   ## Headers
4
5   ### 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.
```

```
101  ## Links
102
103  <https://assemble.io>
104
105  <contact@revolunet.com>
106
107  [Assemble](https://assemble.io)
108
109  [Upstage](https://github.com/upstage/ "Visit Upstage!")
110
111  * [Chapter 1](#headers)
112  * [Chapter 2](#comment)
113  * [Chapter 3](#horizontal-rules)
114
115  ## Footnotes
116
117  This is a digital footnote[^1].
118  This is a footnote with "label"[^label]
119
120  [^1]: This is a digital footnote
121  [^label]: This is a footnote with "label"
122
123  ## Images
124
125  ![Minion](https://octodex.github.com/images/minion.png)
126
127  ![Alt text](https://octodex.github.com/images/stormtroopocat.jpg
     ↪   "The Stormtroopocat")
128
```

Code test data

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a, b;
6      cin >> a >> b;
7      cout << a + b << endl;
8      return 0;
9  }
```

2.4.3   Milestone 2: Implement data structures

At this milestone, the data structure is implemented. Testing the data structure itself is meaningless because it is just some structures used to store data, but whether the design of the data structure is appropriate will be tested throughout

further development.

2.4.4 Milestone 3: Implement the Judger

At this milestone, the Judger is implemented. A series of tests are designed to test whether the Judger works correctly.

Test input	Expect output	Justification
The Normal RunCode test data below.	The number 7 gets output. No error gets reported.	A normal test to make sure the Judger can compile and execute the user's code correctly.
The CE RunCode test data below.	A compile error is reported.	An erroneous test to make sure if the user's code does not compile, the correct error is reported.
The TLE RunCode test data below.	A time limit exceed error is reported. The process is terminated.	An erroneous test to make sure if the user's code exceeds the time limit, it gets terminated and the error is reported correctly.
The MLE RunCode test data below.	A memory limit exceed error is reported. The process is terminated.	An erroneous test to make sure if the user's code exceeds the memory limit, it gets terminated and the error is reported correctly.
The RE RunCode test data below.	A runtime error is reported.	An erroneous test to make sure if the user's code crashes itself, the error can be identified and reported correctly.
The Normal Submission test data below.	The user's code is executed correctly, and success is returned.	A normal test to make sure the Judger can judge a Submission.
The WA Submission test data below.	The user's code is executed correctly, and a wrong answer is returned.	A erroneous test to make sure if the user's output does not match the expected one, an error is reported correctly.

Normal RunCode test data

```
1  Input:
2  3 4
```

```
1  #include <iostream>
```

```
2   using namespace std;
3   int main()
4   {
5       int a, b;
6       cin >> a >> b;
7       cout << a + b << endl;
8   }
```

CE RunCode test data

```
1   int main()
2   {
3       a
4   }
```

TLE RunCode test data

```
1   int main()
2   {
3       while (true) {}
4   }
```

MLE RunCode test data

```
1   int a[100000000];
2   int main()
3   {
4       for (int i = 0; i < 100000000; i ++)
5           a[i] = i;
6   }
```

RE RunCode test data

```
1   int main()
2   {
3       return -1;
4   }
```

Normal Submission test data

```
1   Input:
2   3 4
```

```
3   Expected Output:
4   7
```

```
1   #include <iostream>
2   int main()
3   {
4       std::cout << 7;
5   }
```

WA Submission test data

```
1   Input:
2   3 4
3   Expected Output:
4   7
```

```
1   #include <iostream>
2   int main()
3   {
4       std::cout << 8;
5   }
```

2.4.5   Milestone 4: Create database

At this milestone, the database is implemented. A series of tests are designed
to query, insert, update and delete data from the database to see whether it
works correctly.

Test input	Expect output	Justification
Search an existing problem in the ProblemsPage. Set the List, Tag, Status and Difficulty to their values.	The matching problems show up correctly.	A normal test to make sure database query is working correctly.
Create/Edit a Problem in the CreateNewProblemPage	The Problem is created successfully.	A normal test to make sure the database insert/update is working correctly.
Create/Edit a Problem with invalid data	The change is rejected.	A erroneous test to make sure the data validation is working correctly.
Create/Edit a ProblemList in the CreateNewProblemListPage	The ProblemList is created successfully.	A normal test to make sure the database insert/update is working correctly.
Create/Edit a ProblemList with invalid data	The change is rejected.	A erroneous test to make sure the data validation is working correctly.
Create/Edit an Assignment in the CreateNewAssignmentPage	The Assignment is created successfully.	A normal test to make sure the database insert/update is working correctly.
Create/Edit an Assignment with invalid data	The change is rejected.	A erroneous test to make sure the data validation is working correctly.
Delete a Problem/ProblemList/Assignment	The data is deleted from the database successfully.	A normal test to make sure the database delete is working correctly.

2.4.6 Milestone 5: Handle data import/export

At this milestone, the data import/export module is implemented. A series of tests designed to test whether the software can export and import data correctly.

Test input	Expect output	Justification
Export/Import a Problem.	The Problem gets exported to a JSON file correctly/The Problem is imported correctly.	A normal test to make sure the export/import function is working correctly.
Export/Import a ProblemList.	The ProblemList gets exported to a JSON file correctly/The ProblemList is imported correctly.	A normal test to make sure the export/import function is working correctly.
Export/Import a Assignment.	The Assignment gets exported to a JSON file correctly/The Assignment is imported correctly.	A normal test to make sure the export/import function is working correctly.
Export/Import an AssignmentSubmission.	The AssignmentSubmission gets exported to a JSON file correctly/The AssignmentSubmission is imported correctly.	A normal test to make sure the export/import function is working correctly.
Export/Import a AssignmentSubmissionResult.	The AssignmentSubmissionResult gets exported to a JSON file correctly/The AssignmentSubmissionResult is imported correctly.	A normal test to make sure the export/import function is working correctly.
Import a blank file.	Refuse to import.	A erroneous test to make sure any invalid data are rejected.
Import a non-JSON file.	Refuse to import.	A erroneous test to make sure any invalid data are rejected.

2.4.7 Milestone 6: Handle settings

At this milestone, the settings in the SettingsPage are implemented. A series of settings are designed to test whether the software can process the settings correctly.

Test input	Expect output	Justification
A normal settings file.	The settings can be loaded and processed correctly.	A normal test to make sure the settings can be loaded correctly.
Adjust the settings in the Set- tingsPage.	The settings file has been updated correctly.	A normal test to make sure the settings can be saved correctly.
No settings file.	A default settings file is generated.	An extreme test simulates the situation when the software is first installed and there is no existing settings file. A correct default settings file must be created correctly.
Settings file with incorrect JSON format.	A default settings file overrides the incorrect file.	An erroneous test to make sure the settings module can process files with an incorrect format.
A settings file with incorrect or missing values.	A default value is written back to all incorrect or missing fields. The correct fields are not affected.	An erroneous test to make sure the settings module can deal with incorrect or missing values.

Normal settings file

```
1  {
2      "Theme": "Light",
3      "RunCodeTimeLimit": 1000,
4      "RunCodeMemoryLimit": 67108864,
5      "SelectedLanguage": "Python",
6      "LanguageConfiguration":
7      [
8          {
9              "Name": "C++",
10             "NeedCompile": true,
11             "CompileCommand": "g++",
12             "CompileArguments": "-x c++ {SourceCodeFilePath} -o
    ↪    {ExecutableFilePath}",
13             "RunCommand": "{ExecutableFilePath}",
14             "RunArguments": ""
15         },
16         {
17             "Name": "Python",
18             "NeedCompile": false,
19             "CompileCommand": "",
20             "CompileArguments": "",
```

```
21          "RunCommand": "python",
22          "RunArguments": "{SourceCodeFilePath}"
23        }
24     ],
25     "CurrentUser": "6ee2ebef-3f50-43b7-adf4-78c460339fd0"
26 }
```

A settings file with incorrect format

```
1 {
2     "Theme": "Light,
```

A settings file with incorrect and missing fields

```
1 {
2     "Theme": "RandomTheme",
3     "RunCodeTimeLimit": 1000,
4     "RunCodeMemoryLimit": 67108864,
5     "SelectedLanguage": "Python"
6 }
```

2.4.8   Milestone 7: Handle API calls

At this milestone, the API calls are implemented. A series of test data is designed to test whether the software can handle API calls correctly.

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.
Click the login button on the Account page, and input the wrong email and password.	Login failed with correct error message displayed.	A erroneous test to make sure the software handles login failure correctly.
Login with a teacher account, navigate to the assignment page.	All local assignments and remote assignments created by the teacher show up correctly.	A normal test to make sure the assignments can be fetched correctly.
Login with a teacher account, publish a new assignment.	The assignment gets published correctly.	A normal test to make sure the assignments can be distributed through API correctly.
Login with a teacher account, enter a published assignment, click the mark button to mark all submissions.	All submissions are marked automatically.	A normal test to make sure all submissions can be processed correctly.
Login with a student account, navigate to the assignment page.	All local assignments and remote assignments show up correctly.	A normal test to make sure the assignments can be fetched correctly.
Login with a student account, submit a submission to an assignment.	The submission gets submitted successfully.	A normal test to make sure the submission API works correctly.
Click the logout button on the Account page.	The user gets logged out correctly.	A normal test to make sure the user can log out 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 by creating a set of test problems, test problem lists and test assignments. 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.

To test the APIs, I will need to create a Microsoft Education Organization and several students and teachers account for testing. These testing accounts will be connected to the application to simulate API usages from real users.

Any failed tests 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

3.1 Milestone 1: Create the UI

3.1.1 Objective

During this milestone, a basic prototype with user interface will be created. The prototype needs to compile and run and the user should be able to navigate around the app to see different UI elements.

3.1.2 Create a minimal running prototype

I will create a prototype with a minimal user interface to test run the development environment and building tools.

I create the `App.xaml` file, which will include all application resources. Right now, the only resources imported is the `XamlControlsResource`, which provides the resources for the all controls. Later on, I will create and import new resources in this file.

```
1  <Application
2      x:Class="Algorithm_Dynamics.App"
3
    ↪   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5      xmlns:local="using:Algorithm_Dynamics">
6      <Application.Resources>
7          <ResourceDictionary>
8              <ResourceDictionary.MergedDictionaries>
9                  <XamlControlsResources
                    ↪   xmlns="using:Microsoft.UI.Xaml.Controls" />
```

```
10                <!-- Other merged dictionaries here -->
11            </ResourceDictionary.MergedDictionaries>
12            <!-- Other app resources here -->
13        </ResourceDictionary>
14    </Application.Resources>
15 </Application>
```

I then create the `App.xaml.cs` file, which will include the main entry point of the application. It inheriates from the `Application` class, and create a `Window` object to show the main window. The `App` class will activate the main window when it is launched.

```
1  using Microsoft.UI.Xaml;
2
3  namespace Algorithm_Dynamics
4  {
5      /// <summary>
6      /// Provides application-specific behavior to supplement the
         ↪  default Application class.
7      /// </summary>
8      public partial class App : Application
9      {
10         /// <summary>
11         /// Initializes the singleton application object.  This
            ↪  is the first line of authored code
12         /// executed, and as such is the logical equivalent of
            ↪  main() or WinMain().
13         /// </summary>
14         public App()
15         {
16             InitializeComponent();
17         }
18
19         /// <summary>
20         /// Invoked when the application is launched normally by
            ↪  the end user.  Other entry points
21         /// will be used such as when the application is launched
            ↪  to open a specific file.
22         /// </summary>
23         /// <param name="args">Details about the launch request
            ↪  and process.</param>
24         protected override void
            ↪  OnLaunched(LaunchActivatedEventArgs args)
25         {
26             m_window = new MainWindow();
27             m_window.Activate();
28         }
29
30         private Window m_window;
```

95

```
31        }
32    }
```

Next, I create the `MainWindow.xaml` file, which includes the layout of the main window. Right now, it only has a stack panel and a button inside to test everything is working.

```
1    <Window
2        x:Class="Algorithm_Dynamics.MainWindow"
3
     →   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5        xmlns:local="using:Algorithm_Dynamics"
6        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
7
     →   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
8        mc:Ignorable="d">
9
10       <StackPanel
11           Orientation="Horizontal"
12           HorizontalAlignment="Center"
13           VerticalAlignment="Center">
14           <Button
15               x:Name="myButton"
16               Click="myButton_Click"
17               Content="Click Me"/>
18       </StackPanel>
19   </Window>
```
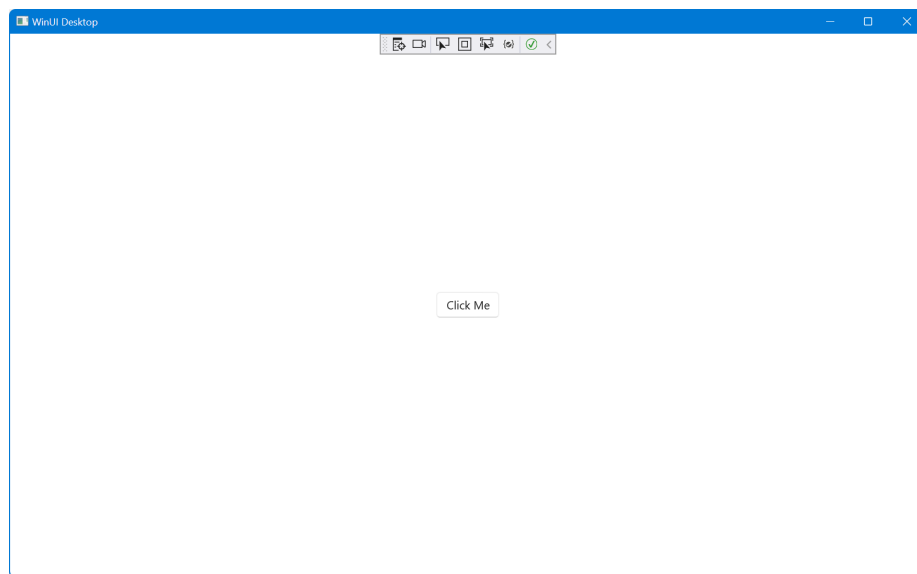
Finally, I create the `MainWindow.xaml.cs` file. It contains all the UI logic. In this prototype, when the button in the UI is clicked, its text will be changed to "Clicked".
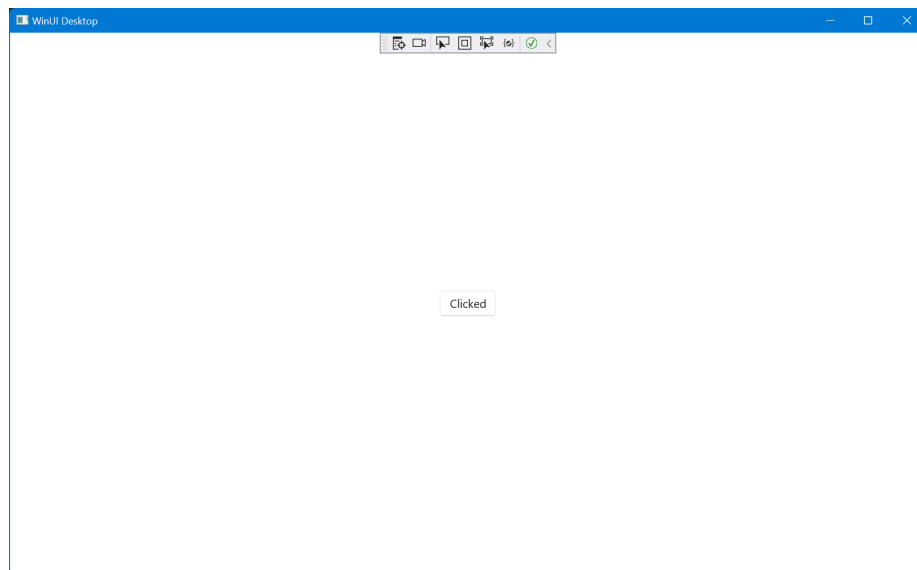
```
1    using Microsoft.UI.Xaml;
2
3    namespace Algorithm_Dynamics
4    {
5        /// <summary>
6        /// The main window object that is used to display all other
         →   elements
7        /// </summary>
8        public sealed partial class MainWindow : Window
9        {
10           public MainWindow()
11           {
12               InitializeComponent();
13           }
```

```
14
15          private void myButton_Click(object sender,
         ↪   RoutedEventArgs e)
16          {
17              myButton.Content = "Clicked";
18          }
19      }
20  }
```


As expected, a window with a button shows up correctly. After clicking the
button, the content in the button is changed to "Clicked" correctly.

This means the development environment and the building tools are set up correct. And I can start developing a more complex user interface.

3.1.3 Create the NavigationView

First, I will write the NavigationView. In the `MainWindow.xaml` file, I add the NavigationView control.

```
1  <NavigationView
2      x:Name="MainNavView"
3      x:FieldModifier="public"
4      IsBackButtonVisible="Collapsed"
5      SelectionChanged="MainNavView_SelectionChanged">
6  </NavigationView>
```

I assign a name `MainNavView` to the control so I can call it later in the code behind. I decide to hide the back button, which prevent the user to navigate the page in an unintended way. I also assign a `SelectionChanged` event handler to the control, which will be used to handle the navigation.

I then need to add different options to the navigation menu.

```
1   <NavigationView.MenuItems>
2       <NavigationViewItem
3           x:Name="HomePageNavViewItem"
4           Icon="Home"
5           Content="Home"
6           Tag="HomePage" />
7       <NavigationViewItem
8           x:Name="ProblemsPageNavViewItem"
9           Icon="List"
10          Content="Problems"
11          Tag="ProblemsPage" />
12      <NavigationViewItem
13          x:Name="AssignmentsPageNavViewItem"
14          Icon="Library"
15          Content="Assignments"
16          Tag="AssignmentsPage"/>
17      <NavigationViewItem
18          x:Name="PlaygroundPageNavViewItem"
19          Icon="Edit"
20          Content="Playground"
21          Tag="PlaygroundPage"/>
22  </NavigationView.MenuItems>
```

There are four options in the menu as designed. Each of which is assigned a name, icon, content and tag. The content and icon will be displayed to the user.

The tag is used to identify the page that will be displayed when the option is clicked.

I add the account option to the footer.

```
1   <NavigationView.FooterMenuItems>
2       <NavigationViewItem
3           x:Name="AccountNavItem"
4           Icon="Contact"
5           Content="Account"
6           Tag="AccountPage"/>
7   </NavigationView.FooterMenuItems>
```

I add the content frame used to contain different pages.

```
1   <Frame
2       x:Name="ContentFrame"
3       Padding="0"
4       Visibility="Visible"/>
```

Now the layout of the NavigationView is finished. I need to add the code to initialize the control and handle the navigation.

When the main window is first initialized, the HomePage should be selected, and the title of the window should be set correctly. So I update the constructor of the main window to set the values.

```
1   public MainWindow()
2   {
3       InitializeComponent();
4       Title = "Algorithm Dynamics";
5       // Select HomePage when first loaded
6       MainNavView.SelectedItem = MainNavView.MenuItems[0];
7   }
```

Next, I need to handle the `SelectionChanged` event, this event gets triggered when the user clicks on a navigation item, and it will update the content frame to display the corresponding page.

```
1   /// <summary>
2   /// Handle the SelectionChanged event of the MainNavView
3   /// Navigate to the corresponding page when the selection is
4   ↪   changed
5   /// </summary>
6   /// <param name="sender"></param>
7   /// <param name="args"></param>
```

```
 7   private void MainNavView_SelectionChanged(NavigationView sender,
     ↪   NavigationViewSelectionChangedEventArgs args)
 8   {
 9       if (args.IsSettingsSelected)
10       {
11           // If the settings is selected, navigate to the settings
             ↪   page
12           ContentFrame.Navigate(typeof(Pages.SettingsPage));
13       }
14       else
15       {
16           // Otherwise, get the selected item. If it is not null,
             ↪   get its Tag
17           // Navigate to Algorithm_Dynamics.Pages.<Tag>
18           NavigationViewItem selectedItem = args.SelectedItem as
             ↪   NavigationViewItem;
19           if (selectedItem != null)
20           {
21               string tag = selectedItem.Tag as string;
22               string pageName = "Algorithm_Dynamics.Pages." + tag;
23               Type pageType = Type.GetType(pageName);
24               ContentFrame.Navigate(pageType);
25           }
26       }
27   }
```

Now, if I build and run the application, I should see the `NavigationView` works correctly. However, I get a runtime exception.

```
 1   System.NullReferenceException
 2     HResult=0x80004003
 3     Message=Object reference not set to an instance of an object.
 4     Source=WinRT.Runtime
 5     StackTrace:
 6      at ABI.System.Type.ToAbi(Type value)
 7      at ABI.System.Type.CreateMarshaler(Type value)
 8      at
        ↪   ABI.Microsoft.UI.Xaml.Controls.INavigate.global::Microsoft.UI.Xaml.Controls.INaviga
        ↪   sourcePageType)
 9      at Microsoft.UI.Xaml.Controls.Frame.Navigate(Type
        ↪   sourcePageType)
10      at
        ↪   Algorithm_Dynamics.MainWindow.MainNavView_SelectionChanged(NavigationView
        ↪   sender, NavigationViewSelectionChangedEventArgs args) in
        ↪   C:\Algorithm-Dynamics\src\Algorithm
        ↪   Dynamics\MainWindow.xaml.cs:line 43
```

It says object reference not set to an instance of an object when executing `ContentFrame.Navigate(pageType);`. This means the `pageType` does not ex-

ist. This is because I forget to create an actual HomePage, so it attempts to load a null object, failed, and throws the exception.

I should create all pages under the `Algorithm_Dynamics.Pages` namespace. For now, all of them will only contain a text block showing their name. Here is an example of the HomePage.

```xml
<Page
    x:Class="Algorithm_Dynamics.Pages.HomePage"

    ↪  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:Algorithm_Dynamics.Pages"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

    ↪  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="{ThemeResource
    ↪  ApplicationPageBackgroundThemeBrush}">

    <Grid>
        <TextBlock Text="HomePage"/>
    </Grid>
</Page>
```

```csharp
using Microsoft.UI.Xaml.Controls;

namespace Algorithm_Dynamics.Pages
{
    public sealed partial class HomePage : Page
    {
        public HomePage()
        {
            this.InitializeComponent();
        }
    }
}
```

Now, the software build and run successfully. The HomePage is correctly selected when it is launched and the TextBlock HomePage shows up correctly means that the right page is loaded.
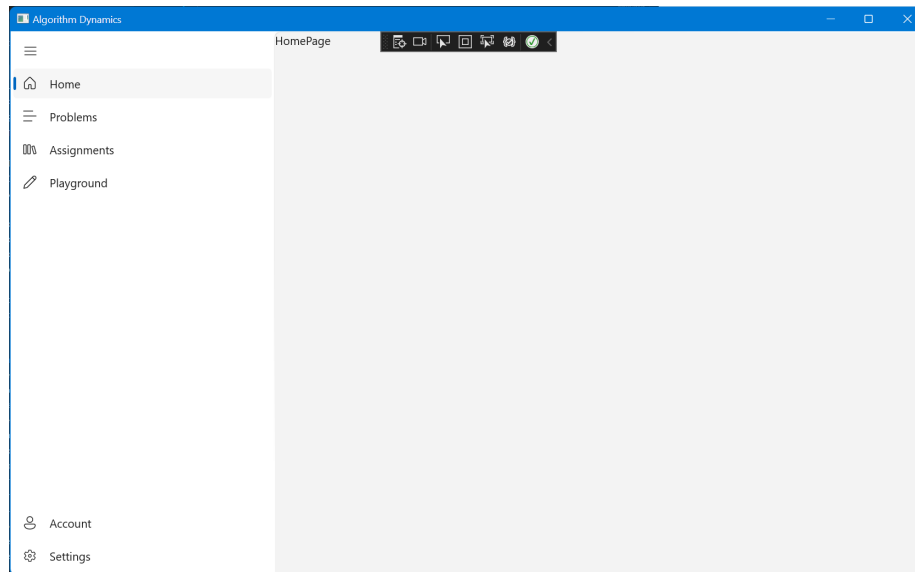
When I click on other tabs on the NavigationView, the corresponding page is loaded correctly as well.
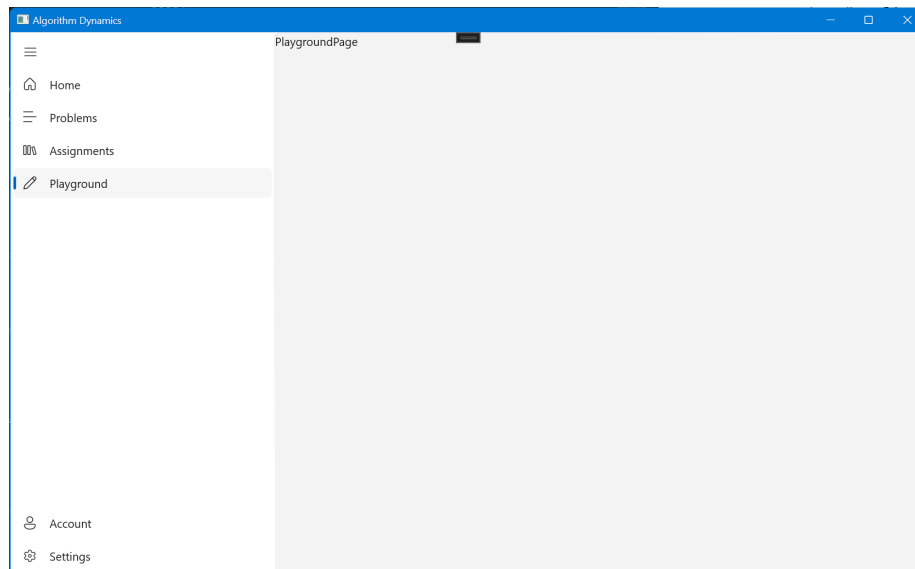
Now the NavigationView is built and tested. I will continue to build other components in the next section.

3.1.4 Create the HomePage

The HomePage is made of three components, the background image, the quick access tools and the recommendations. They are wrapped by a scroller so they can be accessed on any size screen. I place a grid with two rows inside the

ScrollViewer, the first row contains the background image, the second row contains the remaining tools.

```xml
<Page
    x:Class="Algorithm_Dynamics.Pages.HomePage"

        ↪ xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:Algorithm_Dynamics.Pages"
    xmlns:models="using:Algorithm_Dynamics.Models"
    xmlns:controls="using:CommunityToolkit.WinUI.UI.Controls"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

        ↪ xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="{ThemeResource
        ↪ ApplicationPageBackgroundThemeBrush}">
    <ScrollViewer>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="auto"/>
                <RowDefinition Height="auto"/>
            </Grid.RowDefinitions>
            <Grid Grid.Row="0">
                <!-- Background Image -->
            </Grid>
            <Grid
                Margin="24"
                Grid.Row="1">
                <!-- Quick Access tools and Recommendations -->
            </Grid>
        </Grid>
    </ScrollViewer>
</Page>
```

Inside the background image grid, I place a Image control with the source of the background image. It is stretched to fill the entire grid so Strecth is set to UniformToFill.

```xml
<Image
    Source="/Assets/HomePageBackgroundImage.jpg"
    Height="256"
    Stretch="UniformToFill"/>
<TextBlock
    Text="{x:Bind WelcomeMessage, Mode=OneTime,
        ↪ FallbackValue='Welcome User'}"
    HorizontalAlignment="Right"
    VerticalAlignment="Bottom"
```

```
9        Margin="8"
10       Foreground="White"
11       Style="{ThemeResource TitleTextBlockStyle}"/>
```

The `TextBlock` is used to display a greeting message. It is placed on the bottom right. Because its content needs to be adjusted by the current time and the current user, it is binded to the a variable `WelcomeMessage` so I can change its value programmatically.

```
1   namespace Algorithm_Dynamics.Pages
2   {
3       public sealed partial class HomePage : Page
4       {
5           /// <summary>
6           /// The WelcomeMessage is displayed to the user on the
7           ↪  HomePage
7           /// </summary>
8           public string WelcomeMessage { get; set; }
9           public HomePage()
10          {
11              InitializeComponent();
12          }
13      }
14  }
```

The welcome message is adjusted by different time in the day. It should also display the user name, but since the setting is not implemented yet, this part is left to be implemented later.

```
1   /// <summary>
2   /// Set the WelcomeMessage based on the current time.
3   /// Morning: 00:00-12:00
4   /// Afternoon: 12:00-17:00
5   /// Evening 17:00-0:00
6   /// TODO: Add the username after it is implemented
7   /// </summary>
8   private void SetWelcomeMessage()
9   {
10      TimeSpan now = DateTime.Now.TimeOfDay;
11      if (now >= new TimeSpan(00, 00, 00) && now < new TimeSpan(12,
11      ↪  00, 00))
12      {
13          WelcomeMessage = "Good morning, user!";
14      }
15      else if (now >= new TimeSpan(12, 00, 00) && now < new
15      ↪  TimeSpan(17, 00, 00))
16      {
```

```
17        WelcomeMessage = "Good afternoon, user!";
18    }
19    else if (now >= new TimeSpan(17, 00, 00) && now <= new
   ↪    TimeSpan(23, 59, 59))
20    {
21        WelcomeMessage = "Good evening, user!";
22    }
23 }
```
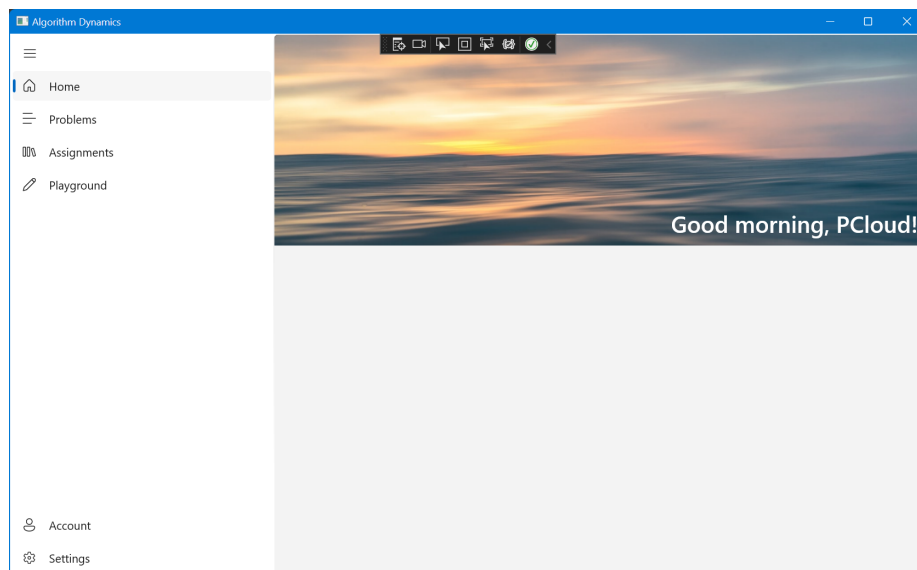
Now I just need to set the welcome message in the constructor. Notice that it needs to be called before all components are initialized, so the text block can bind to the updated value.

```
1 public HomePage()
2 {
3    SetWelcomeMessage();
4    InitializeComponent();
5 }
```


The image and the greeting message show up correctly, and because I am writing this in the morning, the correct greeting message is displayed.

For the quick access toolbar, I choose to use the `AdaptiveGridView` control for its `OneRowMode` function, which allows the toolbar to scroll horizontally on small screens. The controls in the quick access toolbar should be loaded dynamically, so they can be changed easily in the future. So I create a `QuickAccessItem` helper class. Each object will contain a name, an icon and an action. The name and the icon will be displayed to the user, and the action will be invoked when the user clicks on the control.

```
1    using Microsoft.UI.Xaml.Controls;
2    using System;
3
4    namespace Algorithm_Dynamics.Models
5    {
6        public class QuickAccessItem
7        {
8            public string Name { get; set; }
9            public Symbol Icon { get; set; }
10           public Action<MainWindow> Action { get; set; }
11           public QuickAccessItem(string name, Symbol icon,
     ↪   Action<MainWindow> action)
12           {
13               Name = name;
14               Icon = icon;
15               Action = action;
16           }
17       }
18   }
```

Then I create a `ObserableCollection` of `QAItems` that binds to the grid.

```
1    public ObservableCollection<QuickAccessItem> QAItems { get; } =
     ↪   new ObservableCollection<QuickAccessItem>();
```

```
1    <TextBlock
2        Text="Quick Access"
3        Grid.Row="0"
4        Style="{ThemeResource SubtitleTextBlockStyle}"/>
5    <controls:AdaptiveGridView
6        Grid.Row="1"
7        Margin="4"
8        ItemsSource="{x:Bind QAItems}"
9        StretchContentForSingleRow="True"
10       OneRowModeEnabled="True"
11       ItemHeight="160"
12       DesiredWidth="160"
13       SelectionMode="None"
14       IsItemClickEnabled="True">
15       <GridView.ItemTemplate>
16           <DataTemplate x:DataType="models:QuickAccessItem">
17               <StackPanel
18                   VerticalAlignment="Center"
19                   HorizontalAlignment="Center"
20                   Orientation="Vertical">
21                   <SymbolIcon
22                       x:Name="ItemIcon"
23                       Symbol="{x:Bind Icon}"/>
```

```
24            <TextBlock Text="{x:Bind Name}" />
25          </StackPanel>
26        </DataTemplate>
27     </GridView.ItemTemplate>
28  </controls:AdaptiveGridView>
```

I need to add the items to the collection when the HomePage is loaded. I create a procedure `InitializeQAItems`. Because I have passed the MainWindow object to the function, the MainNavView can be called inside. Because the CodingPage and the Import logic is not completed yet, the Random Problem and Import function will be implemented later.

```
1  public HomePage()
2  {
3      SetWelcomeMessage();
4      InitializeQAItems();
5      InitializeComponent();
6  }
7  private void InitializeQAItems()
8  {
9      QAItems.Clear();
10     QAItems.Add(new QuickAccessItem("Random Problem",
       ↪ Symbol.Shuffle, (m_window) =>
11     {
12         m_window.MainNavView.SelectedItem = null;
13         // TODO: Navigate the ContentFrame to the Coding page
              ↪ manually
14         throw new NotImplementedException("[Blocking]: The
              ↪ CodingPage has not been implemented yet.");
15     }));
16     QAItems.Add(new QuickAccessItem("Playground", Symbol.Edit,
       ↪ (m_window) =>
17         m_window.MainNavView.SelectedItem =
              ↪ m_window.MainNavView.MenuItems[1]));
18     QAItems.Add(new QuickAccessItem("Assignments",
       ↪ Symbol.Library, (m_window) =>
19         m_window.MainNavView.SelectedItem =
              ↪ m_window.MainNavView.MenuItems[2]));
20     QAItems.Add(new QuickAccessItem("Problems", Symbol.List,
       ↪ (m_window) =>
21         m_window.MainNavView.SelectedItem =
              ↪ m_window.MainNavView.MenuItems[3]));
22     QAItems.Add(new QuickAccessItem("Settings", Symbol.Setting,
       ↪ (m_window) =>
23         m_window.MainNavView.SelectedItem =
              ↪ m_window.MainNavView.SettingsItem));
24     QAItems.Add(new QuickAccessItem("Account", Symbol.Contact,
       ↪ (m_window) =>
```

```
25        m_window.MainNavView.SelectedItem =
    ↪   m_window.MainNavView.FooterMenuItems[0]));
26      QAItems.Add(new QuickAccessItem("Import", Symbol.Import,
    ↪   (m_window) =>
27      {
28          m_window.MainNavView.SelectedItem = null;
29          // TODO: Handle the import logic
30          throw new NotImplementedException("[Blocking]: The import
    ↪   logic has not been implemented yet.");
31      }));
32  }
```

I need to handle the click event of the quick access toolbar. I create a procedure `QAGridView_ItemClick`, which will invoke the action of the item.

```
1   /// <summary>
2   /// Handle the QAGridView click event. The Action will be
    ↪   invoked.
3   /// </summary>
4   /// <param name="sender"></param>
5   /// <param name="e"></param>
6   private void QAGridView_ItemClick(object sender,
    ↪   ItemClickEventArgs e)
7   {
8       MainWindow m_window =
    ↪   (MainWindow)((App)Application.Current).m_window;
9       if (e.ClickedItem is QuickAccessItem item)
10      {
11          item.Action(m_window);
12      }
13  }
```
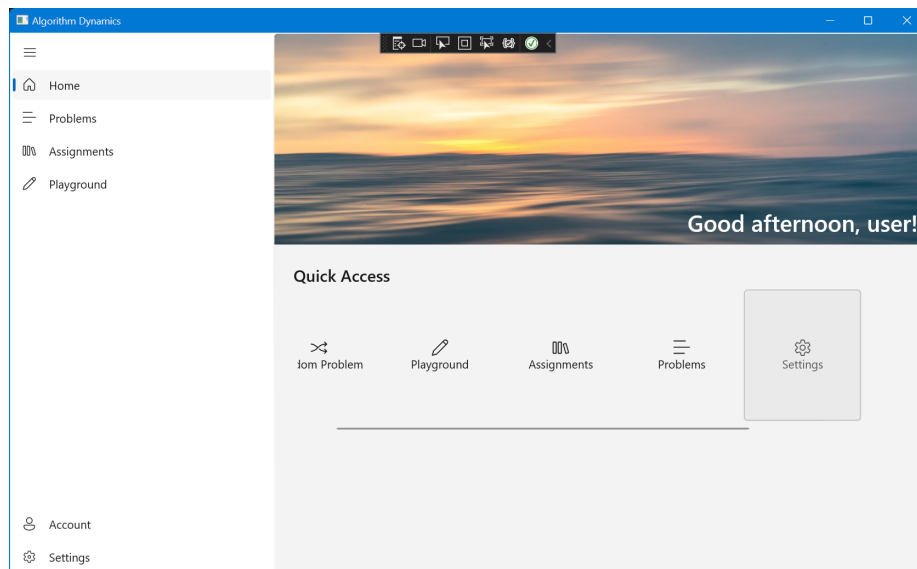
Now, when I build and run the application, the quick access toolbar shows up correctly and scroll horizontally as expected. When I click the different buttons, it navigates to the corresponding page correctly. When I click the Random Problem button, the software panic and a NotImplementedException is thrown with the correct error message.

```
1   System.NotImplementedException
2     HResult=0x80004001
3     Message=[Blocking]: The CodingPage has not been implemented
    ↪  yet.
4     Source=Algorithm Dynamics
5     StackTrace:
6      at
       ↪  Algorithm_Dynamics.Pages.HomePage.<>c.<InitializeQAItems>b__13_0(MainWindow
       ↪  m_window) in C:\Algorithm-Dynamics\src\Algorithm
       ↪  Dynamics\Pages\HomePage.xaml.cs:line 70
7      at
       ↪  Algorithm_Dynamics.Pages.HomePage.QAGridView_ItemClick(Object
       ↪  sender, ItemClickEventArgs e) in
       ↪  C:\Algorithm-Dynamics\src\Algorithm
       ↪  Dynamics\Pages\HomePage.xaml.cs:line 59
8      at
       ↪  WinRT._EventSource_global__Microsoft_UI_Xaml_Controls_ItemClickEventHandler.EventSt
       ↪  sender, ItemClickEventArgs e)
9      at
       ↪  ABI.Microsoft.UI.Xaml.Controls.ItemClickEventHandler.<>c__DisplayClass10_0.<Do_Abi_
       ↪  invoke)
10     at
       ↪  ABI.Microsoft.UI.Xaml.Controls.ItemClickEventHandler.Do_Abi_Invoke(IntPtr
       ↪  thisPtr, IntPtr sender, IntPtr e)
```

The recommendation control is similar. I create the `RecommendItem` class and add the items to the collection.

```csharp
namespace Algorithm_Dynamics.Models
{
    public class RecommendItem
    {
        public string Title { get; set; }
        public string Description { get; set; }
        public RecommendItem(string title, string description)
        {
            Title = title;
            Description = description;
        }
    }
}
```

```csharp
public ObservableCollection<RecommendItem> RecItems { get; } =
    new ObservableCollection<RecommendItem>();
```

Then bind the AdaptiveGridView to the collection.

```xml
<TextBlock
    Text="Recommended"
    Grid.Row="2"
    Style="{ThemeResource SubtitleTextBlockStyle}"/>
<controls:AdaptiveGridView
    x:Name="RecGridView"
    Grid.Row="3"
    Margin="4"
    ItemsSource="{x:Bind RecItems}"
    StretchContentForSingleRow="False"
    OneRowModeEnabled="False"
    ItemHeight="80"
    DesiredWidth="320"
    SelectionMode="None"
    IsItemClickEnabled="True"
    ItemClick="RecGridView_ItemClick">
    <GridView.ItemTemplate>
        <DataTemplate x:DataType="models:RecommendItem">
            <Grid HorizontalAlignment="Stretch">
                <StackPanel
                    VerticalAlignment="Center"
                    HorizontalAlignment="Left"
                    Orientation="Vertical"
                    Margin="8">
                    <TextBlock Text="{x:Bind Title}"
```

```
26                                    Style="{ThemeResource
                                    ↪  BodyStrongTextBlockStyle}"/>
27                     <TextBlock Text="{x:Bind Description}"/>
28                 </StackPanel>
29             </Grid>
30         </DataTemplate>
31     </GridView.ItemTemplate>
32 </controls:AdaptiveGridView>
```

For now, because the database and the data structure are not ready, I just add
some dummy data to the collection. The recommendation logic will be added
later.

```
1  public HomePage()
2  {
3      SetWelcomeMessage();
4      InitializeQAItems();
5      InitializeRecItems();
6      InitializeComponent();
7  }
8  /// <summary>
9  /// TODO: Generate Recommendation from database.
10 /// </summary>
11 private void InitializeRecItems()
12 {
13     RecItems.Clear();
14     RecItems.Add(new RecommendItem("Problem 1", "Easy | Data
        ↪  structure"));
15     RecItems.Add(new RecommendItem("Problem 2", "Medium |
        ↪  Sorting"));
16     RecItems.Add(new RecommendItem("Problem 3", "Hard | Graph"));
17     RecItems.Add(new RecommendItem("Problem 4", "Easy | Data
        ↪  structure"));
18     RecItems.Add(new RecommendItem("Assignment 1", "Due in 2
        ↪  days"));
19     RecItems.Add(new RecommendItem("Assignment 2", "Due in 3
        ↪  days"));
20     RecItems.Add(new RecommendItem("Assignment 3", "Due in 4
        ↪  days"));
21     RecItems.Add(new RecommendItem("Assignment 4", "Due in 5
        ↪  days"));
22 }
```

I need to handle the click event as well. And because neither the Assignment-
Page nor the CodingPage is ready, it will throw an NotImplementedException
too.

```
1  /// <summary>
2  /// Handle the RecGridView click event.
```
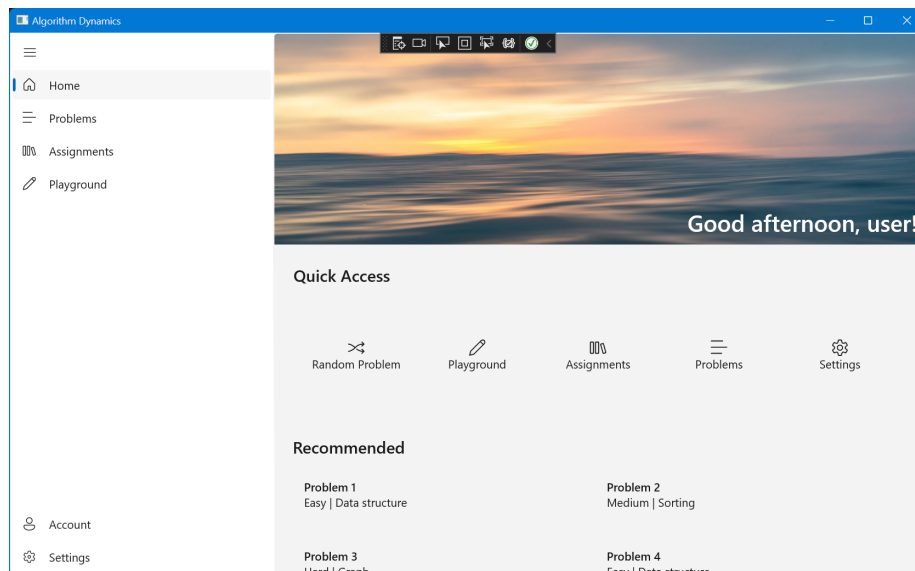
```
3    /// Navigate to the corresponding Problem or Assignment
4    /// </summary>
5    /// <param name="sender"></param>
6    /// <param name="e"></param>
7    private void RecGridView_ItemClick(object sender,
  ↪   ItemClickEventArgs e)
8    {
9        // TODO: Handle the navigation.
10       throw new NotImplementedException("[Blocking]: The CodingPage
     ↪   or the AssignmentPage has not been implemented yet.");
11   }
```
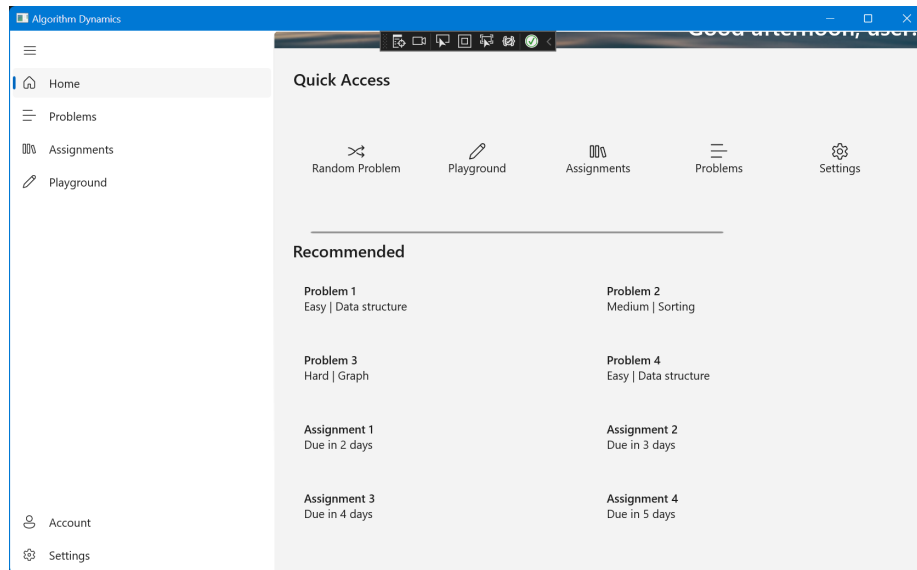
Now, the HomePage is implemented. I run the application and test it.

All components show up correctly. The image is loaded and the greeting message is correct. The Quick Access Toolbar navigates correctly. When I click the recommended item, the correct exception is thrown.

The scroller is working as well, so I can scroll down on a small window to see the recommendation items.

3.1.5 Create the ProblemsPage

I move on to design the ProblemsPage. It contains two parts, the toolbar at the top and the problem list below. I will use a grid containing two rows to hold these two components.

```
1  <Page
2      x:Class="Algorithm_Dynamics.Pages.ProblemsPage"
3
   ↪    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5      xmlns:local="using:Algorithm_Dynamics.Pages"
6      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
7
   ↪    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
8      mc:Ignorable="d"
9      Background="{ThemeResource
   ↪    ApplicationPageBackgroundThemeBrush}">
10     <Grid>
11         <Grid.RowDefinitions>
12             <RowDefinition Height="auto"/>
13             <RowDefinition Height="*"/>
14         </Grid.RowDefinitions>
15         <Grid
16             Grid.Row="0"
17             HorizontalAlignment="Stretch">
```

```
18            <!-- Toolbar -->
19        </Grid>
20        <Grid
21            Grid.Row="1"
22            HorizontalAlignment="Stretch">
23            <!-- Problem List -->
24        </Grid>
25    </Grid>
26 </Page>
```

For the toolbar, I use a grid with 6 columns to hold the search box and the
ComboBoxes.

```
1  <Grid.ColumnDefinitions>
2      <ColumnDefinition Width="2*"/>
3      <ColumnDefinition Width="*"/>
4      <ColumnDefinition Width="*"/>
5      <ColumnDefinition Width="*"/>
6      <ColumnDefinition Width="*"/>
7      <ColumnDefinition Width="*"/>
8  </Grid.ColumnDefinitions>
9  <AutoSuggestBox
10     Grid.Column="0"
11     Margin="4"
12     PlaceholderText="Search..."
13     x:Name="ProblemsSearchBox"
14     HorizontalAlignment="Stretch"/>
15 <ComboBox
16     PlaceholderText="Difficulty"
17     Grid.Column="1"
18     Margin="4"
19     ItemsSource="{x:Bind Difficulties}"
20     HorizontalAlignment="Stretch"/>
21 <ComboBox
22     PlaceholderText="Status"
23     Grid.Column="2"
24     Margin="4"
25     ItemsSource="{x:Bind Statuses}"
26     HorizontalAlignment="Stretch"/>
27 <ComboBox
28     PlaceholderText="Tag"
29     Grid.Column="3"
30     Margin="4"
31     ItemsSource="{x:Bind Tags}"
32     HorizontalAlignment="Stretch"/>
33 <ComboBox
34     PlaceholderText="List"
35     x:Name="ListComboBox"
36     Grid.Column="4"
```

```
37        Margin="4"
38        ItemsSource="{x:Bind Lists}"
39        HorizontalAlignment="Stretch"/>
40  <DropDownButton
41        Content="Add"
42        Grid.Column="6"
43        Margin="4"
44        HorizontalAlignment="Stretch">
45        <DropDownButton.Flyout>
46            <MenuFlyout Placement="Bottom">
47                <MenuFlyoutItem Text="New Problem"/>
48                <MenuFlyoutItem Text="New Problem List"/>
49                <MenuFlyoutItem Text="Import"/>
50            </MenuFlyout>
51        </DropDownButton.Flyout>
52  </DropDownButton>
```

The ItemsSource of the ComboBox is binded to the corresponding variable,
which allows me to set their value programmatically. Because the database is
not implemented yet, I will use some dummy data to test the UI.

```
1   using Microsoft.UI.Xaml.Controls;
2   using System.Collections.ObjectModel;
3
4   namespace Algorithm_Dynamics.Pages
5   {
6       public sealed partial class ProblemsPage : Page
7       {
8           public ProblemsPage()
9           {
10              this.InitializeComponent();
11          }
12          private readonly ObservableCollection<string>
            ↪  Difficulties = new() { "Easy", "Medium", "Hard" };
13          private readonly ObservableCollection<string> Statuses =
            ↪  new() { "Todo", "Attempted", "Done" };
14          public ObservableCollection<string> Lists = new() { "List
            ↪  1", "List 2", "List 3" };
15          public ObservableCollection<string> Tags = new() { "Tag
            ↪  1", "Tag 2", "Tag 3"};
16      }
17  }
```
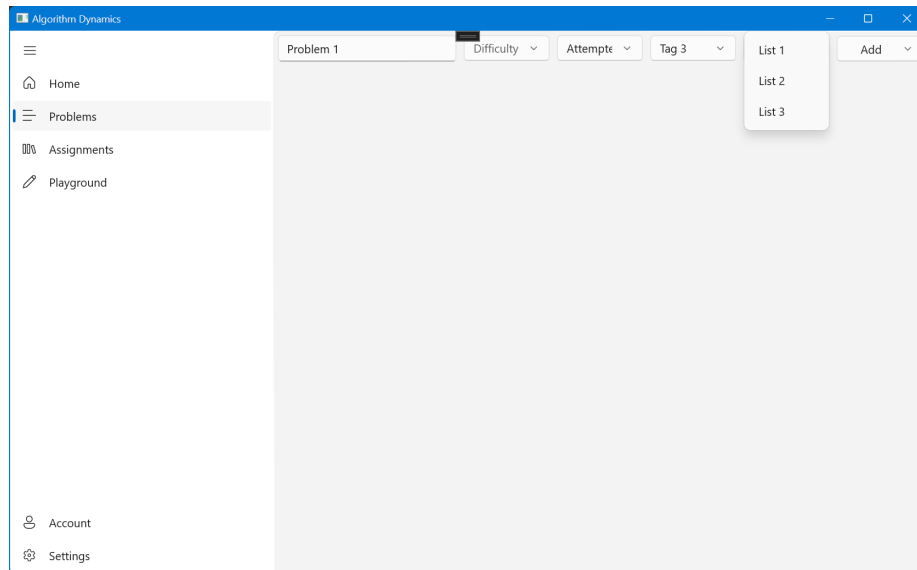
The toolbar shows up with all data loaded correctly. I can select an item correctly, but I cannot clear the selection. I need to empty the ComboBox when the user right clicks it. For the list ComboBox, I also need to allow the user to edit and delete an existing list, so when the user right click the list ComboBox, a menu flyout needs to show up with all options. I create a `ClearComboBox` procedure to handle the right click event.

```
1   /// <summary>
2   /// Clear the <see cref="ComboBox"/>
3   /// </summary>
4   /// <param name="sender"></param>
5   /// <param name="e"></param>
6   private void ClearComboBox(object sender,
    ↪   RightTappedRoutedEventArgs e)
7   {
8       ComboBox comboBox = (ComboBox)sender;
9       comboBox.SelectedIndex = -1;
10  }
```

And then bind the right clicked event to the procedure.

```
1   <ComboBox
2       x:Name="DifficultyComboBox"
3       RightTapped="ClearComboBox"/>
4   <ComboBox
5       x:Name="StatusComboBox"
6       RightTapped="ClearComboBox"/>
7   <ComboBox
8       x:Name="TagComboBox"
9       RightTapped="ClearComboBox"/>
```

116

For the list ComboBox, I need to create the layout for the menu flyout first.

```
1   <ComboBox.Resources>
2       <MenuFlyout x:Name="ListMenuFlyout">
3           <MenuFlyout.Items>
4               <MenuFlyoutItem
5                   Text="Edit"
6                   Icon="Edit"/>
7               <MenuFlyoutItem
8                   Text="Delete"
9                   Icon="Delete"/>
10              <MenuFlyoutSeparator/>
11              <MenuFlyoutItem
12                  Text="Clear"
13                  Icon="Clear"
14                  Click="ClearComboBox"/>
15          </MenuFlyout.Items>
16      </MenuFlyout>
17  </ComboBox.Resources>
```

There are three items in the menu, edit, delete, and clear. The edit and delete function has not been implemented yet, so they will not be handled for now. Because the clear event is sent by the menu button instead of the ComboBox, I overload the `ClearComboBox` procedure to handle the clear event.

```
1   /// <summary>
2   /// Clear the <see cref="ListComboBox"/>
3   /// </summary>
4   /// <param name="sender"></param>
5   /// <param name="e"></param>
6   private void ClearComboBox(object sender, RoutedEventArgs e)
7   {
8       ListComboBox.SelectedIndex = -1;
9   }
```

The menu flyout needs to be displayed when the list ComboBox is right clicked.

```
1   <ComboBox
2       x:Name="ListComboBox"
3       RightTapped="ListComboBox_RightTapped">
```

```
1   /// <summary>
2   /// Display the <see cref="ListMenuFlyout"/> when the <see
    ↪   cref="ListComboBox"/> is right tapped
3   /// </summary>
4   /// <param name="sender"></param>
```
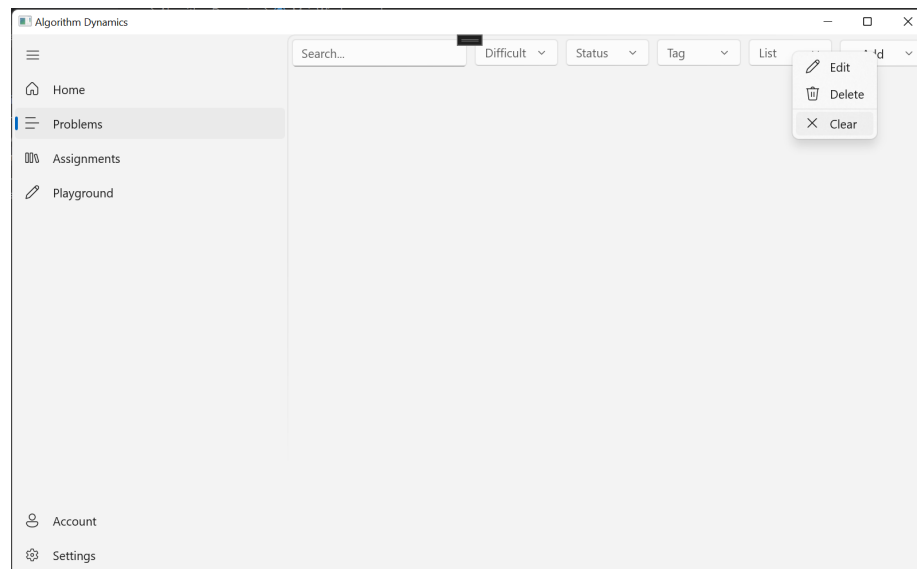
```
5   /// <param name="e"></param>
6   private void ListComboBox_RightTapped(object sender,
    ↪   RightTappedRoutedEventArgs e)
7   {
8       ComboBox comboBox = (ComboBox)sender;
9       ListMenuFlyout.ShowAt(comboBox, e.GetPosition(comboBox));
10  }
```

Now, I build and run the code again. When I right click the difficulty, status and tag ComboBox, their content are cleared correctly. When I click the list ComboBox, the menu flyout shows up. When I click the clear option, the selected list is cleared correctly. When I right click other ComboBoxes, the content is cleared correctly as well.

But there is a potential issue with the current implementation. When I right click the list ComboBox, the menu flyout will show up even when no list is selected. Clear the box does not result in any problem because it is empty anyway, but in the future, when the edit and delete function is implemented, attempting to edit or delete an empty list will cause problem. So I need to change the logic of the menu flyout so that it only shows up when there is something selected. I only need to modify the original function, and only show the flyout when the selected index is -1.

```
1   private void ListComboBox_RightTapped(object sender,
    ↪   RightTappedRoutedEventArgs e)
2   {
3       ComboBox comboBox = (ComboBox)sender;
4       if (comboBox.SelectedIndex != -1)
5       {
6           ListMenuFlyout.ShowAt(comboBox, e.GetPosition(comboBox));
```

```
7        }
8    }
```

Now, the menu flyout does not show up when no list is selected, which fixes a potential issue in the future.

3.1.6 Create the AssignmentsPage

3.1.7 Create the PlaygroundPage

3.1.8 Create the SettingsPage

3.2 Milestone 2: Implement the data structures

3.3 Milestone 3: Implement the Judger

3.4 Milestone 4: Create the database

3.5 Milestone 5: Handle data import/export

3.6 Milestone 6: Handle settings

3.7 Milestone 7: Handle API calls

Chapter 4

Evaluation