

Report



ProCoder

Version 1.0.0

A tool for pro-learner and programming learners. A practice website for programming students where the students create the exercises themselves.

Team Details

Name	Email	Role
Lingxiu Cai	lcai224@aucklanduni.ac.nz	Team Leader, Full-stack developer
Fengyuan Ran	fran799@aucklanduni.ac.nz	Front-end developer, Tech lead
Jinhua Fan	jfan200@aucklanduni.ac.nz	Back-end developer, DBA
Laura Zhou	rzho558@aucklanduni.ac.nz	UI/UX & art designer, QA
Hetong Wang	hwan223@aucklanduni.ac.nz	Full-stack developer, DBA

Roles	Responsibility
Team Leader	<ul style="list-style-type: none"> Plan fortnight client meetings and arrange team meetings. Follow up on project progress and report to the client.
Tech Lead	<ul style="list-style-type: none"> Assigning job duties to team members. Plan and lead technical solutions.
Quality Assurance	<ul style="list-style-type: none"> Conduct user testing. Identify issues with progress.
Back-end Developer	<ul style="list-style-type: none"> Design database Develop backend controllers and APIs
Front-end Developer	<ul style="list-style-type: none"> Develop frontend website, make it functional Server deployment
Full-stack Developer	<ul style="list-style-type: none"> Participate in both back-end and front-end development
UI/UX & art Designer	<ul style="list-style-type: none"> Design user interface elements, front-end aesthetic and images required
Database	<ul style="list-style-type: none"> Design and implement database

Administrator	
---------------	--

Executive Summary

“Practice makes perfect.”

Students rely heavily on feedback when solving programming problems rather than thinking for themselves. As an example, students using Coderunner would use the Precheck function to see their results rather than write their program and test it out themselves. It develops a bad habit – the students start to rely on feedback provided by the test cases given by the “expert”. This leads to the problem that students might not understand the learned concept profoundly and are not able to solve real-life problems. This will be a severe problem after the students finish university and enter the workspace.

In order to prevent students from over-relying on the precheck result, our goal is to encourage students to make up their own questions. We created this web application, ProCoder, for students to develop their programming skills and help them learn. Our users are able to post programming questions by providing descriptions and several test cases. During this process, they can develop the skill to describe a problem clearly, come up with suitable test cases, and understand the programming concepts involved in their problem. The posted questions will be available for any users to try and solve it.

To distinguish from other handy tools that already exist, our “ProCoder” focuses more on social aspects. There is a whole community section for users to discuss programming, either related to a question or just general discussions.

“ProCoder” also introduced incentives like achievements and a ranking system. Users will be rewarded for participating in the activities we provide. Based on the ranks, there is a leaderboard on the home page, which shows everyone who is the most active and admired user. Users also have their profiles, where they are able to see their recent activities and achievements.

By encouraging users to participate and interact with each other, ProCoder would be a platform where users can develop a deeper understanding of the concepts and thus improve their programming skills.

Table of Contents

1. Team Details	2
2. Executive Summary	3
3. Table of Contents	4
4. Introduction	5
a. Aims and objectives	5
b. Target audience	5
c. Scope	5
d. Approaches	6
e. Important outcomes	6
5. Background	7
6. Specification & Design	9
a. Specification	9
b. Design	12
7. Implementation	21
a. Posing page implementation	21
b. Question page implementation	21
c. User Profile page implementation	22
8. Results & Evaluation	24
a. Results	24
b. Evaluation	26
9. Future Work	30
a. Goals Not Achieved	30
b. Future Direction	30
10. Conclusion	32
a. Aims	32
b. Key findings	32
c. Major outcomes	32
d. Acknowledgements	32
11. Table of Authorship	33
12. Reference	34
13. Appendices	35
a. Links	35
b. Gantt Chart	35

Introduction

Aims and objectives

This project aims to develop a web application that involves students not only in programming, but also in creating problems for other students to solve. We want students to have the ability to learn concepts and apply them to other problems. The specific objectives of this project can be divided into three main categories:

1. Develop students' programming skills.
2. Encourage students to make up their own questions.
3. Build up a community where students can discuss with others.

Target audience

The project's target audience is university students taking programming courses, especially the ones taking introductory courses. To attract students to use our web application, we made the process of posting questions as easy as possible and added some incentive mechanisms.

Scope

The project is to develop a practice website where students can publish programming exercises, attempt existing exercises by making code submissions, receive immediate feedback on those submissions, and be able to view and share submitted code.

Software deliverables:

1. Create questions by providing descriptions and test cases.
2. Explore created questions
 - a. Provide correctness feedback for the submissions to those questions.
 - b. Post comments to the questions and join the existing discussions.
 - c. View the user's submission history.
3. A community page shows all general comments and comments on questions.
4. A leaderboard shows the top ten active users.
5. User profile
 - a. Achievements achieved by posting, answering and commenting on questions for a certain amount.
 - b. Student's weekly data and recent submissions.
6. Databases: AdminUsers, Users, UserAchievements, Solutions, Questions, Comments, FollowComments, Achievements.

Documentation deliverables:

1. Project Proposal
2. Readme
3. Project management plan
4. Final Report

Approaches

The project's minimum requirements are supporting some form of authentication, supporting the creation and storage of programming exercises and handling submissions to the practices. Before starting to work on the project, we compared the existing programming practice tools and identified their most promising features. In order to develop the best possible web application in ten weeks, we decided to split the project into two parts: front-end and back-end. We chose to start with the backend development, worked on the backend and tested using Swagger and Postman in the first half of the ten weeks. For the next half, we worked on the front-end development based on the APIs we developed in the back-end. We used HTML, CSS, Javascript, JQuery and Bootstrap. Other tools that we used included Jobe server, ACE Editor and Monaco Editor. We basically followed our original plan and achieved most of the tasks. The management of our project is done by using Jira software, which provides planning and roadmap tools to facilitate our team to manage functional requirements from the beginning.

Important outcomes

Building on the minimum viable product, our web application 'ProCoder' supports users using Python to post and answer questions. 'ProCoder' allows users to view others' posted questions, submit their solutions, and join the discussion under the questions. Users need to log in to experience most of the functionalities. Apart from the basic functionalities, any successful actions such as posting, answering and commenting on one question will bring users achievement badges and points, which are visible in the user profile. 'ProCoder' also provides a community closely related to questions, so users will not need to copy the question to other community platforms. The ten users with the highest points will be shown on the leaderboard to encourage users to participate in activities. Our application is also user-friendly and supports multiple platforms.

Background

“Practice makes perfect.”

Practising is the most helpful way for people to learn and develop their skills, especially for coding.

The university is using Coderunner for students to practise and learn programming skills. There are also many fully functional websites out there that provide these exercises of different difficulty levels.

However, a fact that the Computer Science teaching team discovered is that students rely too much on the feedback that the website gives them. Students stop testing and debugging their code by themselves. Instead, they click on the “Precheck” button repeatedly until they pass all the test cases. Most students doing so would not have the ability to bring up comprehensive test cases.

Once they enter the workspace and start developing a program with this lousy habit, there is a much higher chance that their program will not work as expected.

Existing Solutions

The existing tools include CodeRunner, which the university is using. This platform is more for tests and exercises that the teaching team needs to provide for students to practise. It is created more on examination aspects. As the university uses it for assignments and tests, CodeRunner does not contain a forum for discussion, or allow students to create questions themselves.

LeetCode and HackerRank are the two most popular platforms for practising coding, especially for preparing for interviews. Both are fully functional websites that provide exercises of different difficulty levels and contain a discussion forum under each question. We tested and compared the layout of the post-solution interfaces of the three websites and asked for students' advice. They provide descriptions of the question, example outcomes and test cases. Users can run their submitted codes, receive feedback, see the sample solutions and discuss the question with others. The design of LeetCode and HackerRank has question descriptions on the left and code boxes on the right. While CodeRunner's question descriptions are at the top of the code boxes, users need to scroll back and forth to view the details of questions during practice, which gives users a bad experience. Hence, in this project, we tried to work towards the LeetCode's solving page.

Both LeetCode and HackerRank have a ranking system to encourage users' involvement. We took inspiration from these two sites, and our product introduces a ranking system based on users the points gained by users from creating, answering and commenting on questions.

Community is an essential part of learning. From the discussion page in LeetCode and HackerRank, we found that we need to add interaction between users to our project to meet our needs better.

CodeRunner has an achievement system; after students have done a particular exercise, they will gain a corresponding achievement badge. We gained the idea of building our own achievement system from it to encourage students' participation.

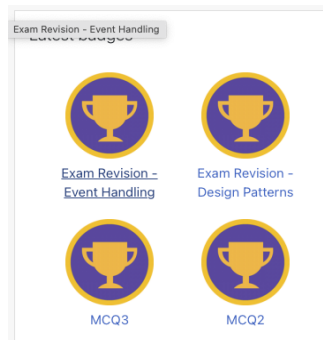


Fig.1 CodeRunner Badges from <https://coderunner.auckland.ac.nz/>

These sites for learners all focus on the ability to develop their skills to solve a problem rather than the ability to debug and test codes by themselves. Our project 'ProCoder' allows users to develop the ability to test their codes.

Value of our work

Using ProCoder, and getting involved in posting and solving problems will help starters learn. Our product provides a platform for students to create a programming question easily. Based on our learning experience, no platform allows students to create the exercises. ProCoder is a platform where students have the freedom to create exercises and discuss programming with each other. With incentives like leaderboard and achievements, we are also encouraging students not just to solve questions, but to participate in posting and commenting.

Methods and tools

The tools we used can be roughly divided into three parts:

Back-end development using C# .Net 6, a database storing with SQLite.

Our front-end development uses HTML, CSS, Javascript, JQuery and Bootstrap.

Moreover, we are using swagger and postman for testing.

Other tools that we used include Jobe server, Monaco Editor, and AWS.

Jobe server provides feedback for users' submissions, and we used Amazon AWS to run our Jobe server. We have also deployed our .Net 6 backend and static front-end website on AWS.

Specification & Design

Specification

ProCoder is a code practice website that allows users to publish programming exercises, attempt exercises created by other users, and receive immediate feedback on their submissions. Our design aims to help students develop the habit of coming up with test cases and provide a coding community for social aspects. In Addition to posting and solving, ProCoder also provides a comprehensive forum system and user achievements system. We implemented social aspects and incentives to make our users more willing to participate in all question-related activities.

The features included in the project can be divided by the following users' actions:

1. View Actions

• Explore questions:

The explore page is a list of questions posted by users. At most five questions will be shown on each page, auto-ordered by question id in descending order. On the left, there is the search function. Users can search by question content, including the words in question titles, descriptions and tags. Alternatively, they can select one or more tags that they want to browse.

• Question details:

There are four sections for different aims for each question. Details of this question are shown in the first section on the left. Users can see the function name, the description of what they need to achieve, and some examples provided by the author.

• Submit history:

After a user makes a submission, they check their past solution in the submit history.

• Community interface:

There are two types of discussions, general discussion and comments on questions. General comments can cover a wide range of subjects, and they do not need to relate to any of the existing questions. For the comments that are linked to a question, their tags are derived from their questions.

Users can view all comments from the community page. Alternatively, they can go to the discussion section under every question.

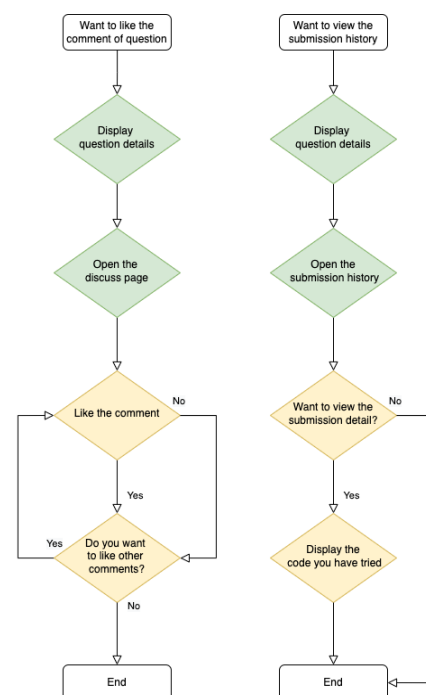


Fig.2 Flow chart of comment and submission

- **User profile:**

The user can change their profile photo or edit info in the user profile. The whole section on the left displays the user's activities. The achievements achieved by posting, solving, and commenting a certain amount of times are viewable from the profile. This page also contains the recent submission history, containing the question title and the results. Green for passing and red for failing. Users can also see a line chart showing their activities in one week and details of each day of this. There is also a pie chart showing overall activities. The user's participation in ProCoder calculates the point below the profile photo.

- **Leaderboard:**

Points are calculated by the user's involvement in ProCoder. Any successful actions may bring them points: posting or solving questions and interacting with others. Similar to achievements. The ten users with the highest points will be shown on the leaderboard on the home page.

2. **Submit Actions**

- **Post questions:**

Users must provide the title, function name, description, sample solution, and test cases to create a question. The preview window will give the author an idea of what will show up for the user trying to solve this problem. We made the sample solution a required field not only to simplify the process of generating test cases, but also to reduce the chance for an error to occur in the test cases. Users can get their expected outputs by running the sample solution and test cases with Jobe. After users provide the required details for creating a question, they can post it. Then all users can find the created questions on the explore page.

- **Submit solutions:**

ProCoder can satisfy the user's basic need to compile and run code in the code editor. With the support of Jobe server and Monaco editor, the result of the execution of the code will display at the bottom of the code editor to indicate whether the user has passed the question's test case or not. Therefore, users can gain immediate feedback from the code editor and adjust their solution if needed.

- **Send comments:**

- a. *Comments*

Users can also share their thoughts by posting a general comment and choosing one to two labels that match the content. Alternatively, they can post their comments under a particular question.

- b. *Followup comments and likes*

Under a comment, users can join the discussion by sending a follow-up comment or clicking the like button to show their favour.



Fig.3 Flow chart of posting, answering and commenting on questions

3. Register and Login Actions

Most of the functionalities in ProCoder need users to log in before they are able to use it. Users can create their own account by providing a user name between one and fifteen characters, an email address and setting up a password with at least six and at most twelve characters. They must provide information that meets the above criteria to create an account successfully.

Design

Technologies Used

The technologies used C#, .Net 6, SQLite, Jobe Server, Monaco Editor, HTML, CSS, Javascript, JQuery and Bootstrap. We tried to use some frameworks, such as Vue.js, but the learning journey of such a framework will take a long time. Therefore, we decided to give up using these frameworks. The diagram below shows the technologies used for ProCoder.

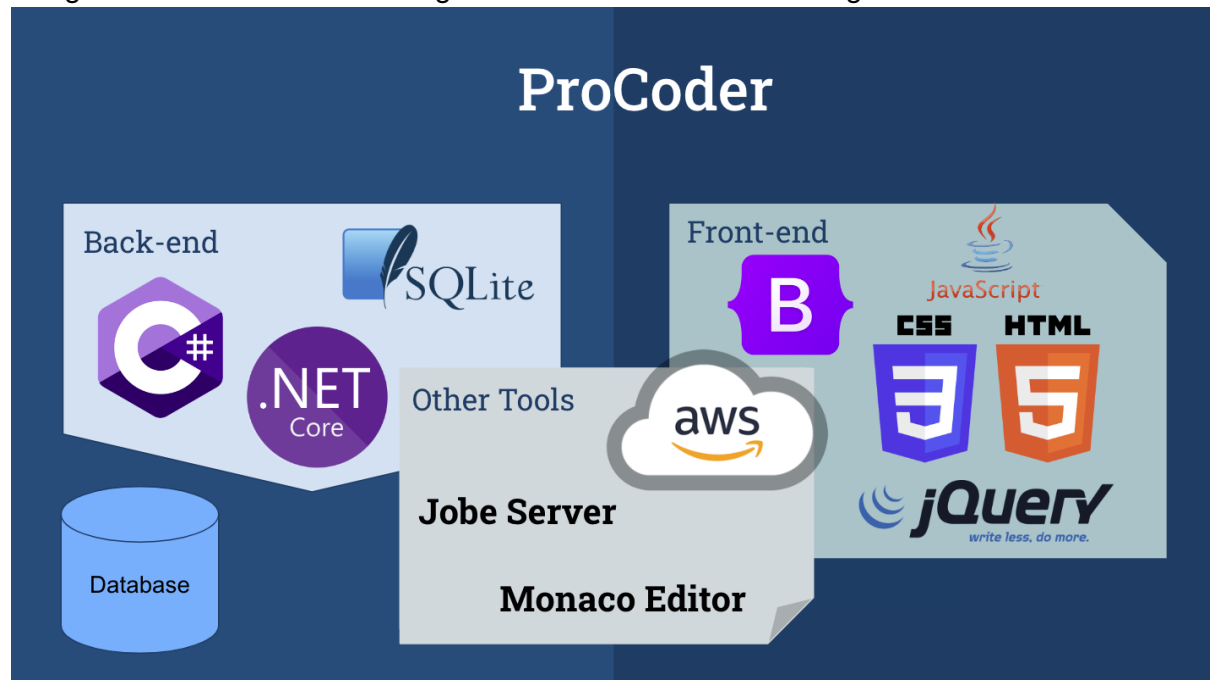


Fig.4 Technologies used by ProCoder

Prior to creating the website, prototypes were designed on how the website should look and after much discussion, the following prototype was decided. We used this as a foundational design and something we strived to recreate.

Navbar



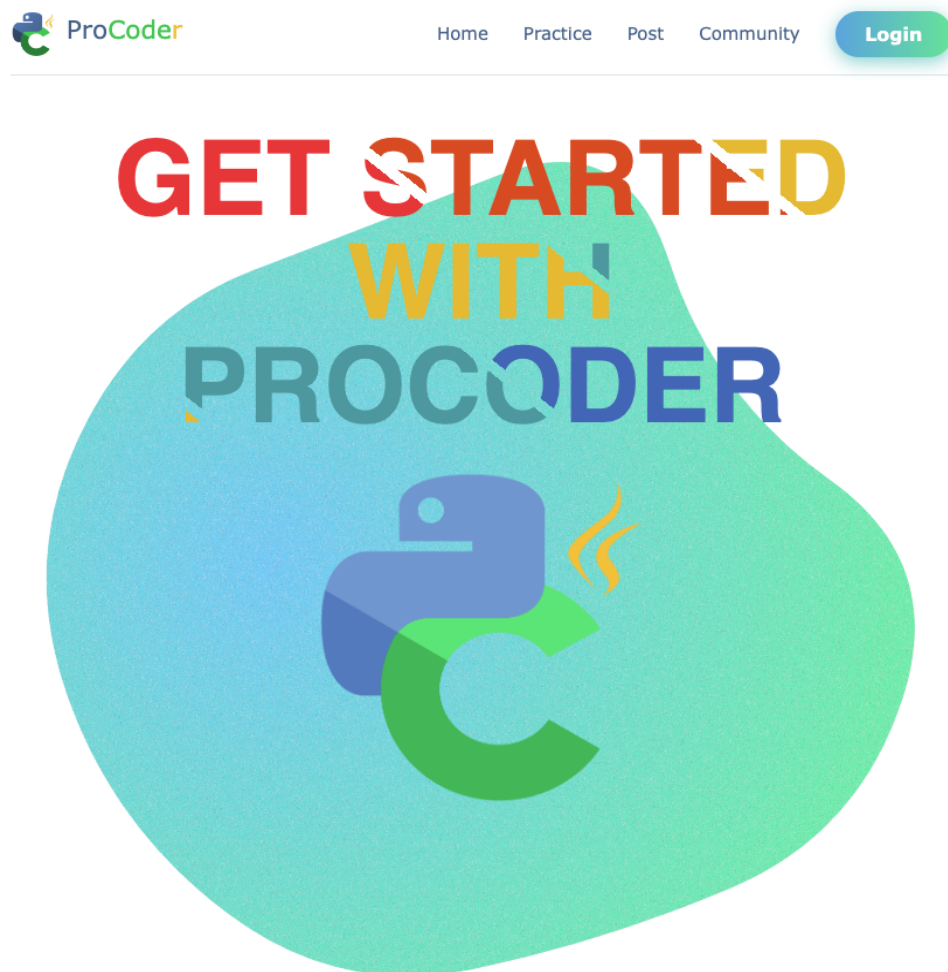
Home Practice Post Community

Login

Fig.5 ProCoder Navigation bar

The navbar is displayed at the top of all pages and contains all the information our website provides. On the left is the logo of ProCoder, it will reload to the home page by clicking on it. On the right, we have five different links that give the user a clear first impression of the website's primary focus. These designs improved the website's usability, which enhanced the ease of use and decreased the learnability. Users may click on the Login button to get Login and register to the ProCoder, which will create a long-term personal relationship with the website (Meaningfulness). The navbar will always appear at the top of the website, and users can directly access the other pages by clicking on the links in the navbar, which increases the use efficiency. Each link on the right-hand side of the navbar has a button and is close together, which visually makes the navigation bar belong to the same group. Without a doubt, the design meets the similarity and proximity principles.

Home Page

**Fig.6** ProCoder Homepage

The above image is a screenshot of the home page. The elements are arranged from top to bottom in order of navbar, body and footer. The logo of the homepage image on the homepage is mainly related to the coders, which is large and clear to attract the user's attention by the emphasis principle. "P" in our logo represents pro, inspired by the python logo, and "C" represents coder, inspired by the C# logo.

Check Our ProCoders








Pro	Coder	Ranks
		226
		112
		76
		42

Fig.7 ProCoder Ranking on Homepage

The above image shows the leaderboard of our website. This will show up the top contributors of ProCoder with their rank and points. The achievement image and the user photo next to a text are helpful for users to understand the text intuitively. This design meets the balance principle as the image seems to have more 'weight' than the text. This also meets the principle of symmetry and unity.

The info section is one of the homepage's primary functions, allowing users to find descriptions for specific areas, thus increasing the website's usefulness. These four cards are closed together with the same size picture, background colour and CSS design. Therefore, this meets the principle of similarity and proximity in the gestalt perception and the unity principle. These four cards belong to one group visually and are related to each other. The info section also meets the balance principle, providing an equal 'weight' of the info section, left and right, top and bottom.

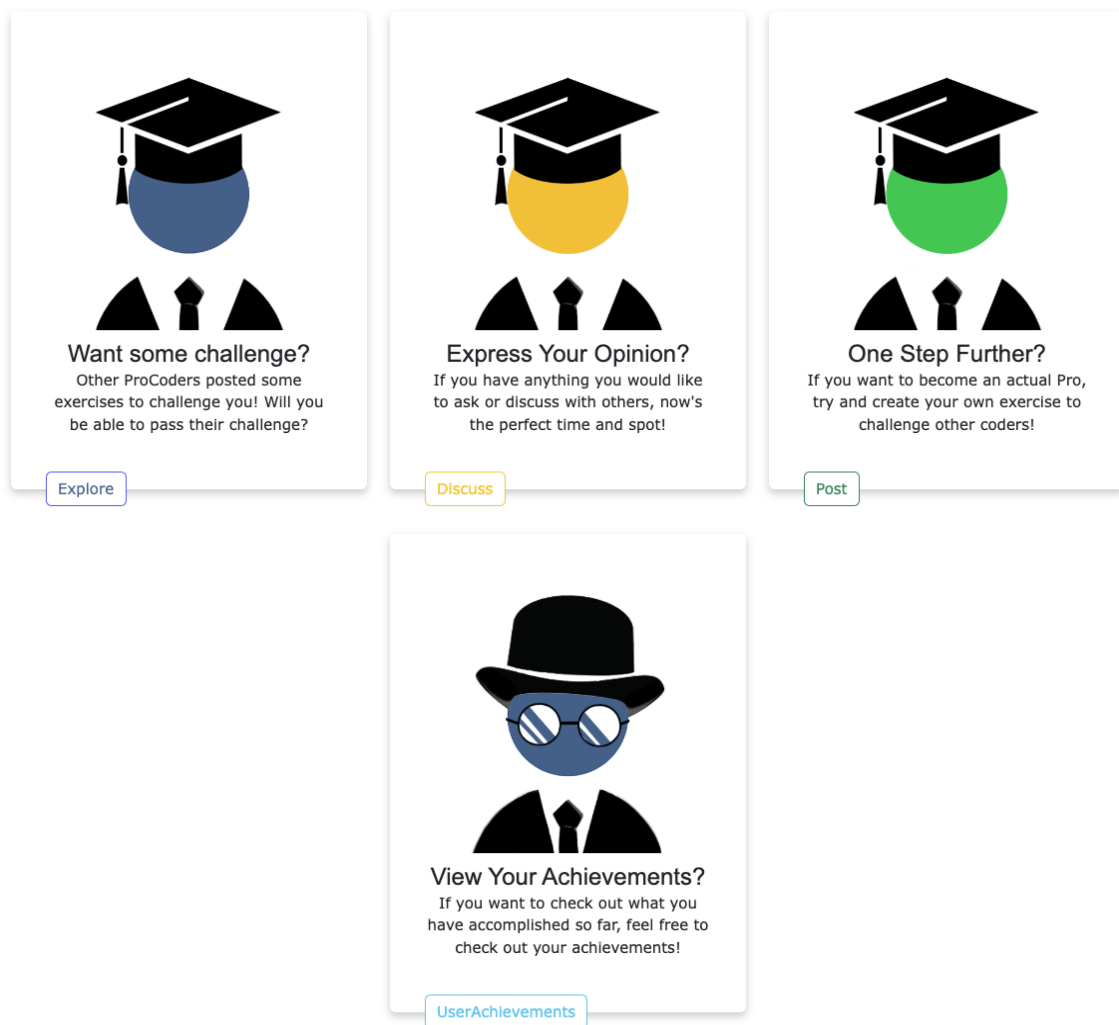



Fig.8 Info Section on ProCoder Homepage

Practice Page

 ProCoder

[Home](#)
[Practice](#)
[Post](#)
[Community](#)
[Login](#)

Explore Some Questions!

☰

Search by Tags:

Easy

Medium

Hard

Python

Java

C

Algorithm

Array

Sorting

Math

Counting

Tree

Sort by Time

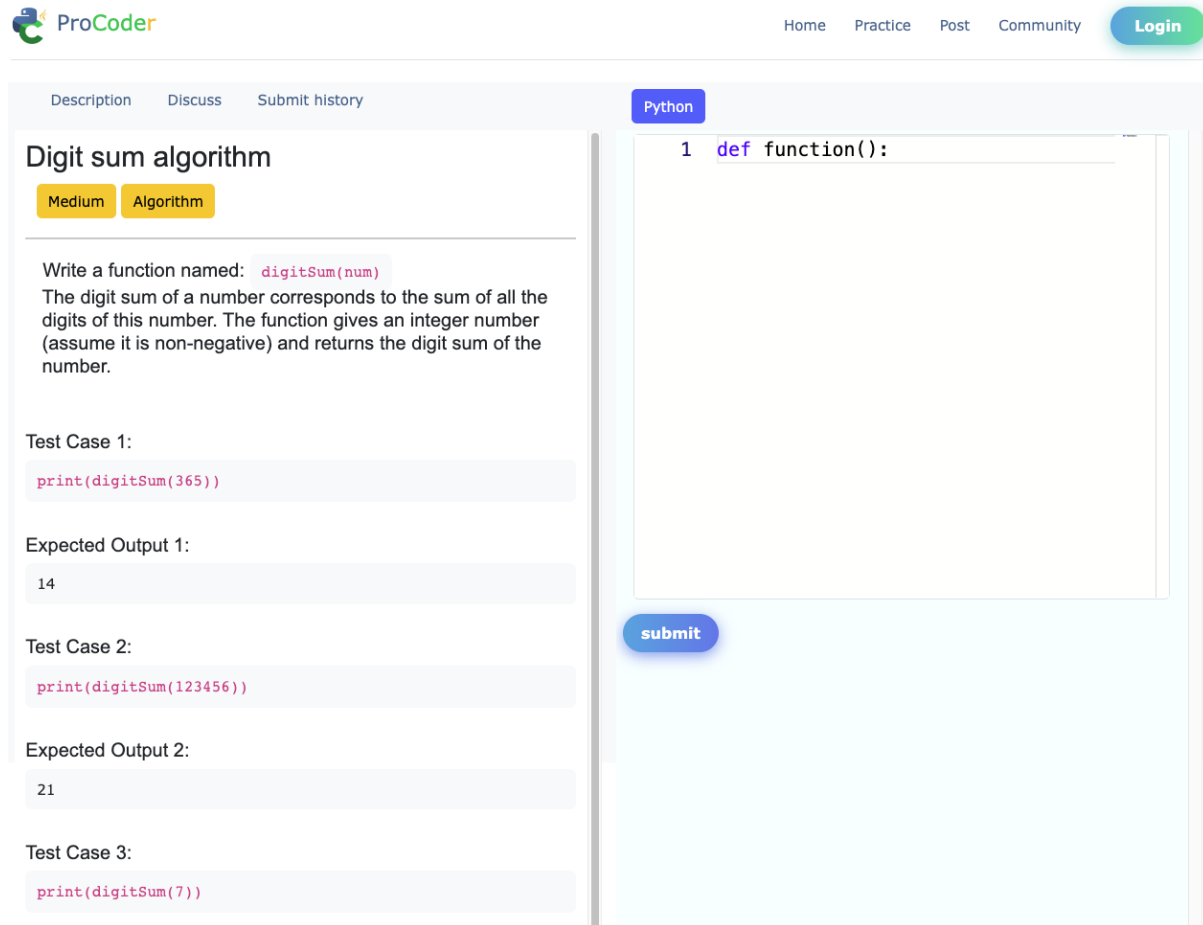
Content	Tags	Post Time
Count Even numbers in a list: This function takes in a...	Python Easy Math	2022-10-21
Largest Divisible Subset: Given a set of distinct posi...	Python Hard Array	2022-10-19
Greatest Common Divisor: The greatest common divi...	Python Medium	2022-10-19
Digit sum algorithm: The digit sum of a number corres...	Python Medium ...	2022-10-19
Shuffle cards: cards = ["TWO", "THREE", "FOUR", "FIV...	Python Medium Array	2022-10-19

1
2
3
4
Next

Fig.9 ProCoder Practice Page

The above image is the screenshot of the practice page. On this page, we have a search area on the left and a table of questions on the right. These two sections have a colour contrast that tells the users they represent two different contents. These tags' different colours will help users navigate the search functions. This design meets the balance principle as the table section seems to have more 'weight' than the search section. The tags in the search area appear close to each other, making them belong to one group visually. This is the principle of proximity and unity. Each question content in the content table appears to have the same length and style, making the content table symmetric visually. This meets the principle of symmetric and unity.

Question Page



The screenshot shows the ProCoder website interface. At the top, there's a navigation bar with 'Home', 'Practice', 'Post', 'Community', and a 'Login' button. The main content area is divided into two columns. The left column, titled 'Digit sum algorithm', contains a 'Medium' difficulty tag, an 'Algorithm' tag, and a description: 'Write a function named: `digitSum(num)`. The digit sum of a number corresponds to the sum of all the digits of this number. The function gives an integer number (assume it is non-negative) and returns the digit sum of the number.' Below this, there are three test cases: 'Test Case 1' with input `print(digitSum(365))` and expected output '14'; 'Test Case 2' with input `print(digitSum(123456))` and expected output '21'; and 'Test Case 3' with input `print(digitSum(7))`. The right column is a code editor for Python, showing the start of a function definition: `1 def function():`. A 'submit' button is located at the bottom of the code editor.

Fig.10 ProCoder Question Page

On the question page, we divide the website into two different sections. One is the question description section on the left, and the other is the question-answering section on the right. On the left, we can see all the details provided for this question. On the right, there is the code editor section which allows users to answer the question. There is an equal 'weight' between the description section on the left and the question-answering section on the right. This design follows the balance principle, which gives users a better user experience.

Posting Page

[Home](#) [Practice](#) [Post](#) [Community](#)[Login](#)[Create Your Own Question!](#)

Language*

Choose a Language

Difficulty level *

Choose a Difficulty Leve

Category

Choose a Category

Question Title *

Title ...

Question Description *

Write a function named: * functionName(parameter1, parameter:
Hint: Please at least include the data type of Input parameters and a description of the outcome.
Description ...

Sample Solution *

Hint: Please choose a language before enter sample solution.

Question Title

Difficulty

Language

Write a function named:
functionName(parameter1, ...)
Question description

Sample solution
sample solution

Test Case 1
Test case 1

Expected Output 1
Expected Output 1

Test Case 2
Test case 2

Expected Output 2
Expected Output 2



Post

Fig.11 ProCoder Post Page

On the posting page, we divide the website into two different sections. One is the user post prompt section on the left, and the other is the user prompt display section on the right. On the left, we can see all the details the user required to prompt to create the question. On the right, there is a section that displays the user's prompt content simultaneously while the user is typing. The purpose of the user prompt display section is to remind users what they have prompted to avoid any mistakes occurring during posting. There is an equal 'weight' between the user post prompt section on the left and the user prompt display section on the right. This design follows the balance principle, which gives users a better user experience.

User Profile

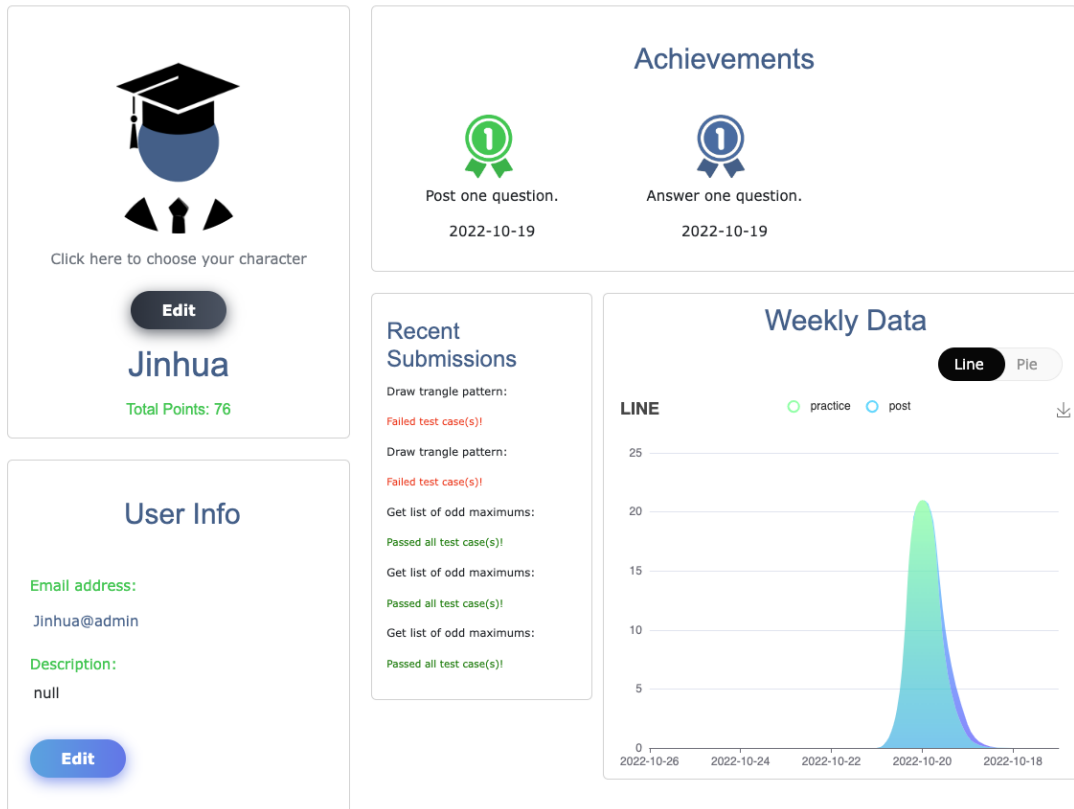

[Home](#) [Practice](#) [Post](#) [Community](#) [Jinhua](#)


Fig.12 ProCoder User Profile

The user home page is divided into five distinct sections corresponding to the user profile photo, user information, user achievements, recent submission and weekly data respectively. Using the card and row/col properties on div offered by the Bootstrap framework, all five sections are close to each other, following the proximity and unity principle. This design visually makes all five information sections belong to one group to indicate that this page contains all the user information. The design of the recent submission also matches the system and the real world as in the real world. Red is used to indicate danger or failure in this case, and green is used to indicate safety or success in this case. The colour contrast is aimed to remind users of their submission status.

The ER Diagram below demonstrates how the data is stored in the database:

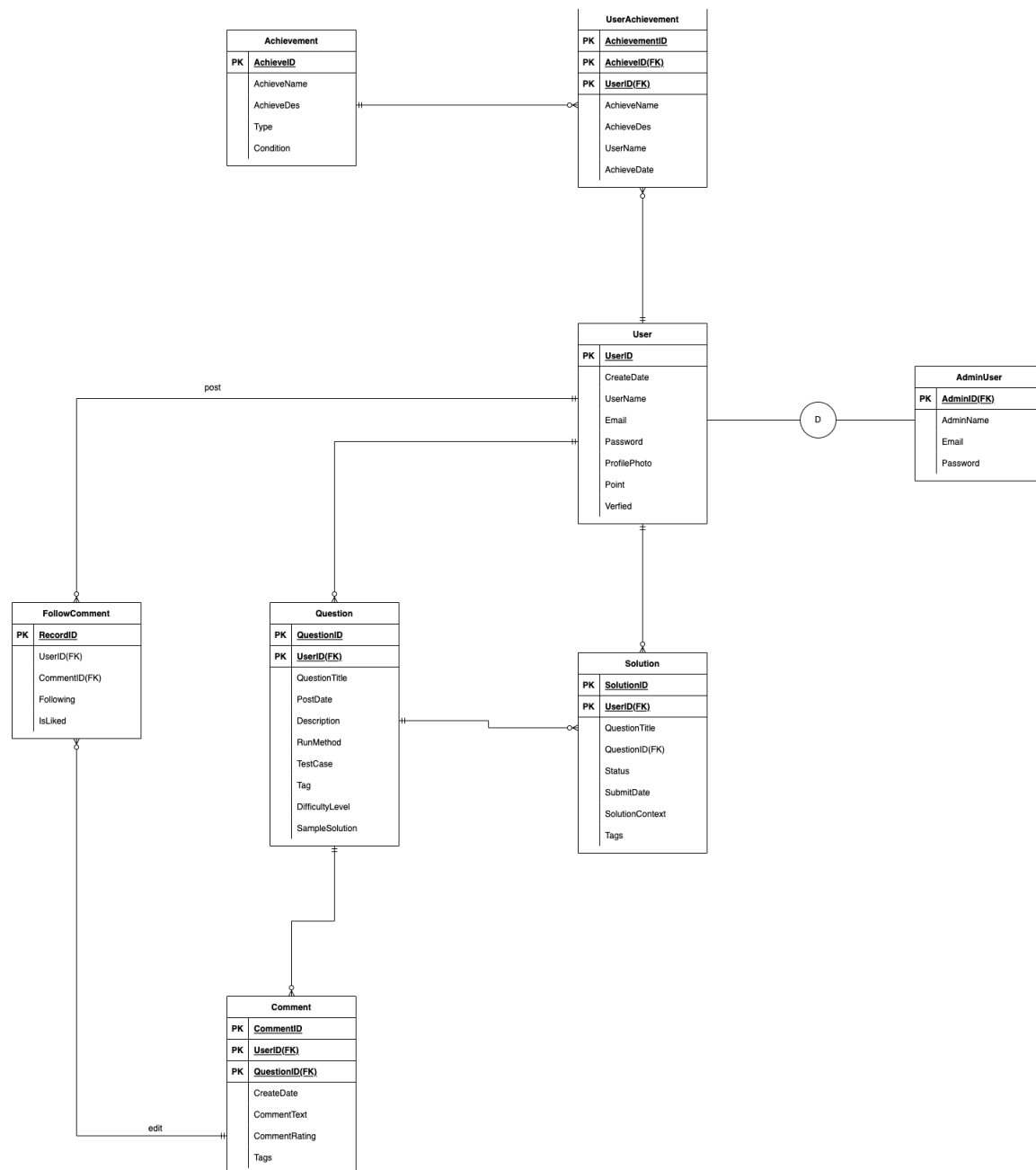


Fig.13 ProCoder ER Diagram

Implementation

In our project implementation, we followed the timeline in [Appendix Fig.24]. The implementation can be divided into the following three parts: posting page, question page and user profile page.

Posing page implementation

For the question-posing page, we had two generations. After implementing our low-fidelity design, the website is not full of content. We felt it should have more features that should decrease running costs for users and make it easier for users to post more questions. Therefore, we considered redesigning the posting page from the brainstorming, and we thought the posting page should be straightforward. Then, we tried to have a preview window of the question; it should automatically display all the information provided by the user in the preview window. Without a doubt, now the users may have a brief view that the question they were posing should be shown on the practice page.

During the redesign of the second generation, we tested the posing page many times, and a problem occurred frequently. There were errors when the user may fill the wrong test cases and except output. Then we wanted to implement some functions to decrease the chance of getting errors when they were filling out the test cases. After a long brainstorming session, we concluded that the users might use their sample solution to run test cases and get the expected outputs. Then we implement a check function to these test cases; it will automatically run the test cases and get the results. By doing that, users may check their sample solution and the except outputs for these test cases. Before posting this question, the user may double-check all the information they want to be shown in the preview window.

Question page implementation

For the question page, we have also adopted a two-column display scheme to facilitate user queries and code writing. The left side can show the question description, submitting history, and discussion section. The question description page details the requirements, tags, and test cases set by the question's author. The discussion section provides an environment for users to express their opinions and seek help. Furthermore, the users can look up all commits in the submit history, the time of each commit, its success or failure, and the content of the submitted code(presented as a code box).

The core function of the question page is the code editing panel displayed on the right side. The user enters and runs the code here, and the system automatically compares the results with the test cases set by the question's author and records the results in the submit history.

There are two core technologies on this page, Jobe (run code) and Monaco editor (edit code).

1. Jobe [1]:

Jobe is a server that supports running small compile-and-run jobs in various programming languages. Jobe specifies a programming language, the source code, the standard input to the run and an optional list of additional files. Jobe compiles the

source code and runs it with the given input data. It returns a `run_result` object containing various status information plus the output and error output from the run.

We faced many challenges because Jobe can only be deployed on the server. Because of the lack of server experience in the previous learning experience, all the configuration and basic operations need to be learned, including the use of AWS. For example, some commands in the terminal can only be performed by the root user.

2. Monaco Editor [2]:

The Monaco Editor is the code editor that powers VS Code. It is a package that Node can install. It cannot be directly imported into the site because import is ES6 syntax, and browsers cannot recognise it, so we use Babel to convert it to ES5 syntax and find a function called `require` in the converted file. The function's function is to introduce modules.

We introduced some packages in the node environment, packaged them with Webpack and put them in the browser environment. The browser reports error required is not defined. The reason is that it requires a built-in node function and is not supported in the browser. The solution is to compile the js file through the tool Browserify or Webpack and turn it into a recognisable function on the browser side.

User Profile page implementation

The implementation of the user profile page can be divided into two parts: backend and frontend:

1. Backend implementation

All the backend APIs related to user achievements are developed using ASP.NET 6 and database storing with SQLite. All the database-related code is stored in the model folder, which represents the user achievement and achievement entity. The user achievement entity is used to resolve the 'many to many' relationships between the user and achievement (see the ERD above). Therefore, it contains both user information and achievement information. All the operations that are performed on the user achievement and achievement are stored in the data folder. These methods are used to fetch user achievement information out given certain user/achievement information as input. All the user achievement APIs are used to connect the backend and front end.

During the back-end implementation, the largest unforeseen problem we have encountered with the database table `userAchievement` is how to deal with adding achievements to users. We came up with three stages of solutions. The first version of the solution is adding achievement by checking if the user has achieved the achievement conditions manually. The next version is to use different methods defined in the repo file to add all the achievements that the user has achieved. There are three types of achievements. Therefore, three functions correspond to solve, post and comment, respectively. After discussion and consideration, we thought this approach would cause future updating to become inefficient. It will increase the complicity while adding new achievements. This leads to the final version of the solution following several steps. The first step is to create question achievement, solution achievement, and comment achievement methods that add the corresponding achievement

to the user account when the condition is fulfilled. The second step is to update the existing user achievements by calling one of the update methods depending on the achievement type. Therefore, the user achievement will be added automatically after checking for eligibility every time when a user answers, posts, or comments on a question.

2. Front-end implementation

We use Echarts [3] to draw the weekly data figure. Echarts is a tool for data visualisation, representing a single piece of data through the right type of visual chart, making it more intuitive to show trends, comparisons, and peaks. Users can switch directly between line and pie charts. Line charts can show a comparison of the last week or month of questions made and questions asked, and can also be selected individually, and users can also download icons for easy recording of data.

Results & Evaluation

Results

As for our goals, we followed our original plan and achieved most of the tasks in time. Most of the basic and bonus functionalities we planned are all implemented. We came up with a few new ideas every now and then, but managed to fit them in while fulfilling our plan.

We also encountered some unexpected challenges, including half of our team being offshore, implementing achievement, and the budget limit reached for AWS, etc. We manage to resolve them in different ways. We did not plan the testing too detail, but we test any method or functionality once they are done.

Backend Testing

So far, we have developed seven controllers for controlling how a user interacts with ProCoder. It contains over forty custom routes to implement specialised routing requirements, such as creating user accounts and posting questions, solutions and comments. For back-end testing, we were still at the stage of manual testing. We tested our controllers and custom routes using Swagger and Postman by comparing the outcome we got with our expected outcomes.

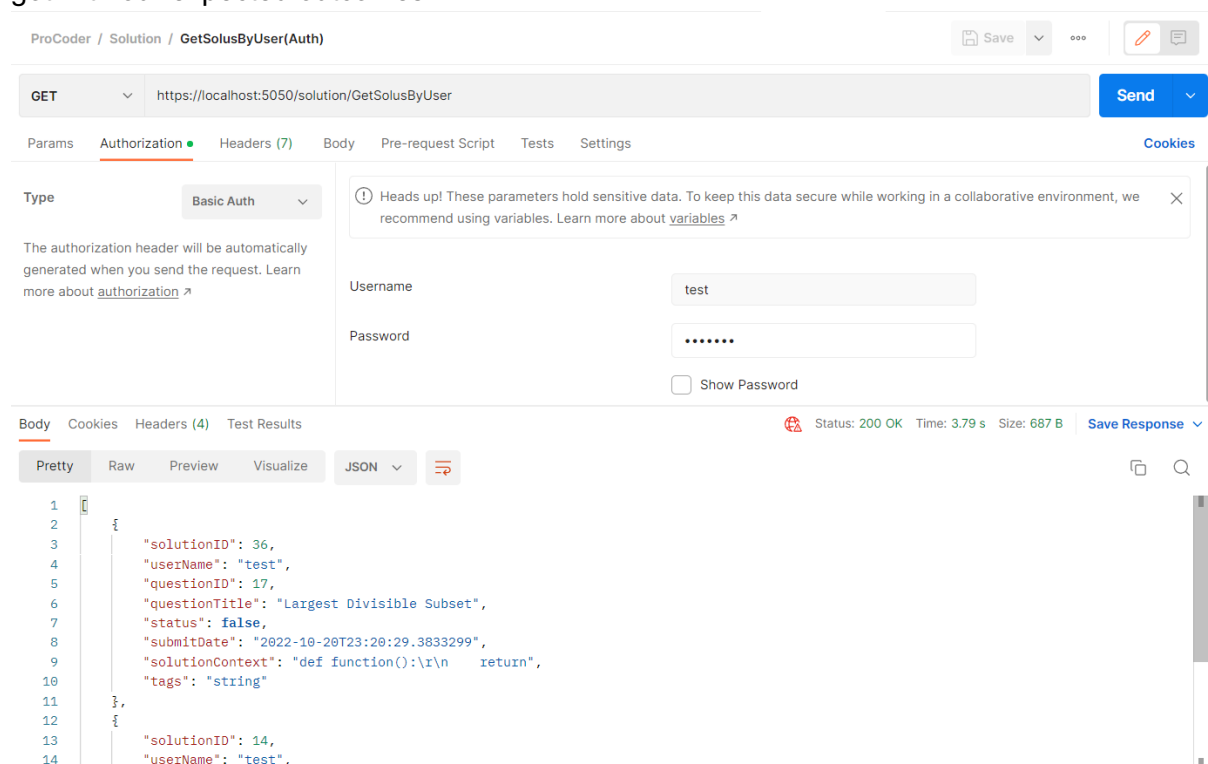


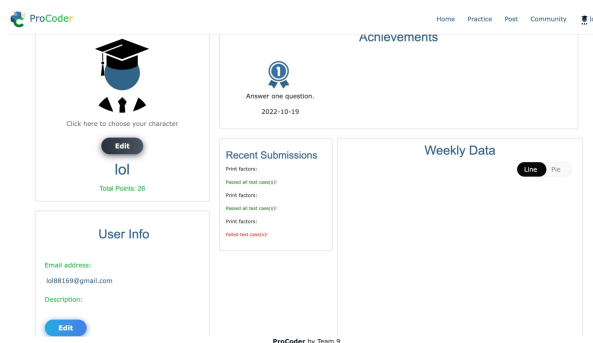
Fig.14 ProCoder Postman Testing

Front-end Testing

For front-end testing, we were also at the manual testing stage. For the functions that do not require user input or database, for example, navigation bar links and homepage display. We test the availability and compatibility on different devices and browsers. Windows with any

browsers are working perfectly, while Safari and Chrome on Mac will lose some special effects. The basic functions still work, so we left that aside. We did not attempt to make a mobile UI, but we tested the display on mobile devices. iPhones still lose some special effects, and the font size is overly tiny. The posting page preview window is located in the wrong spot, but other than that, the website works perfectly.

For the functions that require database input, for example, leaderboard, profile points and weekly stats, we compared the displayed output with our data from several test accounts. The only bug we found here was that the charts in the user profile would only display if the user had posted at least one question.



If a user has not posted any questions yet, this space does not display anything, even if this user solved a question and achieved the “answer a question” achievement.

This was caused by the fact that we did not catch the error when there was no posting data for the user. This display error was fixed by setting up an initial graph.

Fig.15 ProCoder User Profile Error

Finally, the most complicated part requires user input, including registration, post questions, solutions and comments. We have restrictions on what kind of inputs will be accepted for registration and comments. Any kind of solution will be accepted, with a status telling if this solution passes or fails the test cases.

After we set up the Jobe server to compile and run the codes on the posting and solving pages, we tested the posting function manually. We used test accounts to post questions, create different test cases, and then try to solve them. We immediately found an error, as it is shown in the screenshot. The expected and got results are exactly the same but were judged as failures:

test cases	Expected	Got	
<code>int(product_list_of_lists([[1, 60, 6, 16], [3, 4, 5], [3, 2], [1, 4, 4, 1]]))</code>	[1, 60, 6, 16]	[1, 60, 6, 16]	×
<code>int(product_list_of_lists([[0, 2, 15, 48], [2], [3, 5], [2, 4, 6]]))</code>	[0, 2, 15, 48]	[0, 2, 15, 48]	×
<code>int(product_list_of_lists([[], []]))</code>	[0, 0, 0]	[0, 0, 0]	×

We found that the result stored in our database and the actual running result is different when it contains tabs, spaces or newline characters. It was due to how the SQLite database stores space and newline characters.

Fig.16 ProCoder Question Page Error

We were not able to find a solution, so the way we solved it was to replace all newline characters with ‘%%n’ while storing. However, we did not find a way to replace the space and tab characters, so they still do not work as we expected. Instead, we need to manually replace the spaces in the expected output with ‘ ’ in the database.

Evaluation

Then, with a survey, we invited some people to experience our product. Most of the people who participated in this survey are students that learned coding in their university courses. Most of them have used at least one programming practice website. Below are screenshots of statistics from the survey:

What's your age?
20 responses

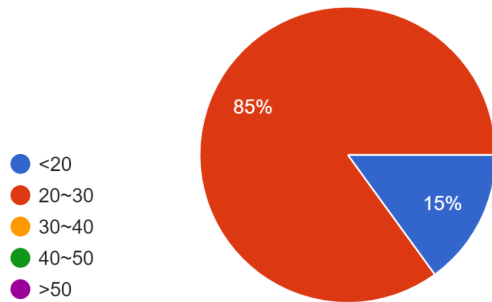


Fig.16 (Left) Tester gender, from ProCoder Survey

Does your major require programming?
20 responses

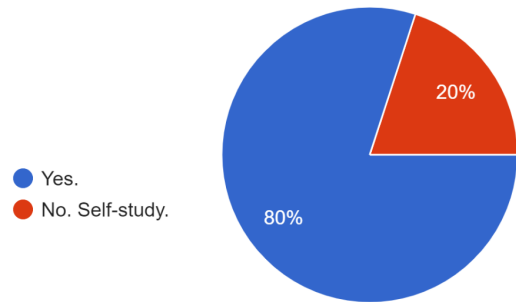


Fig.17 (Right) Tester background, from ProCoder Survey

Have you ever used the following websites?
20 responses

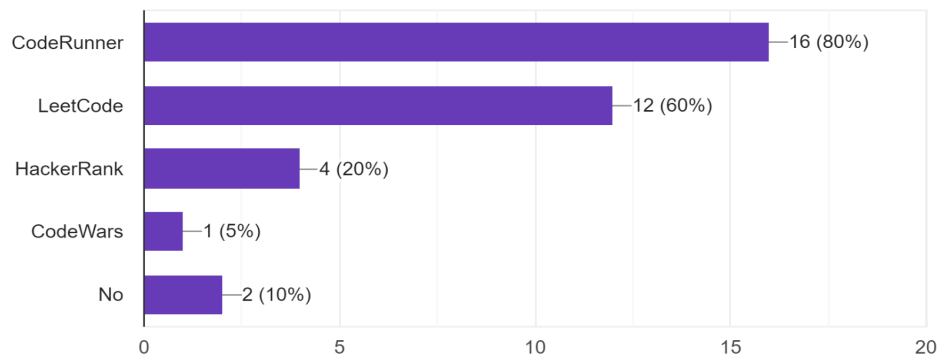


Fig.18 Whether the tester has used code practising websites, from ProCoder Survey

One of ProCoder's key strengths is the user community. The social aspect is what we are trying to focus on. Other coding platforms usually support commenting on questions but usually do not support follow-up comments. The following are the results we got from the survey regarding the community:

How do you find the community page?

20 responses

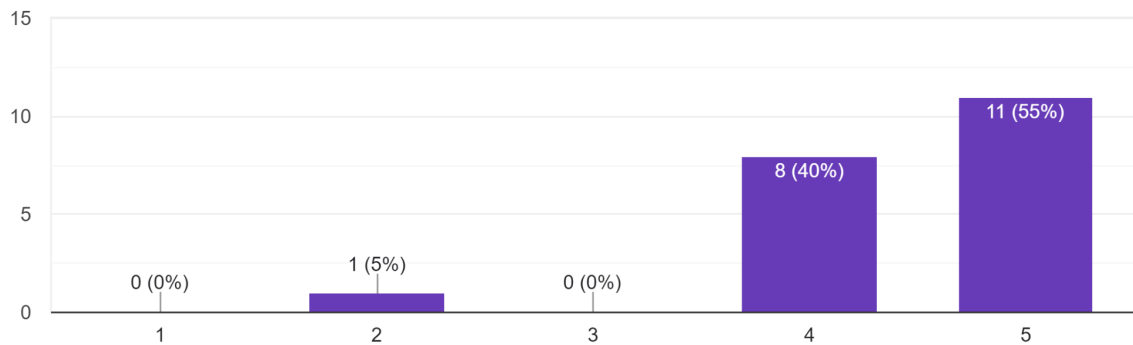


Fig.19 Community Page, from ProCoder Survey

Users, are genuinely pleased with the forum ProCoder supports.

Another strength is our posting function with a preview window that gives the author an idea of what will show up for the user trying to solve this problem. However, the posting is still not fully functional due to the bug included earlier. This leads to a weakness that would really affect the user experience. Users who create exercises with a space or tab in their expected output will never be solvable. Those who attempt to solve these questions will never pass it, even if the expected and run outputs are all the same.

How do you find the posting question functionality?

20 responses

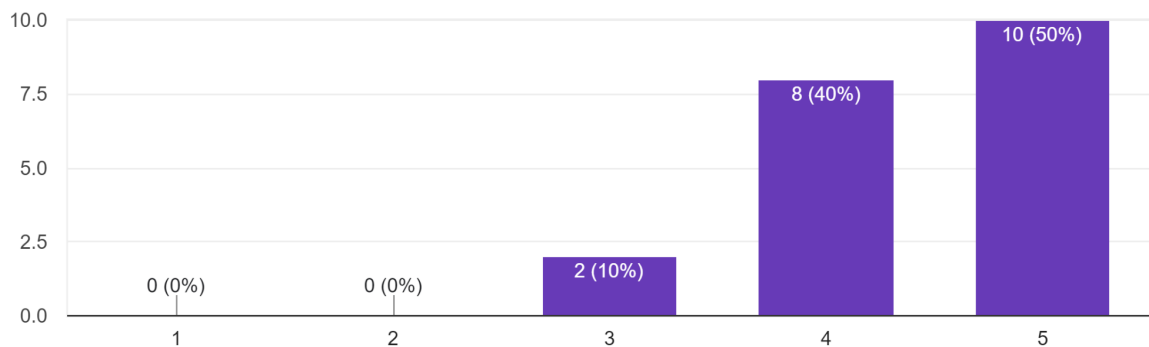


Fig.20 Posting question functionality, from ProCoder Survey

Above is the feedback we got for the posting page.

We are performing exploratory testing through the survey as well. The feedback we gained through the online survey results contains some suggestions that could improve UI/UX:

Do you think these function entrances are easy to find?

20 responses

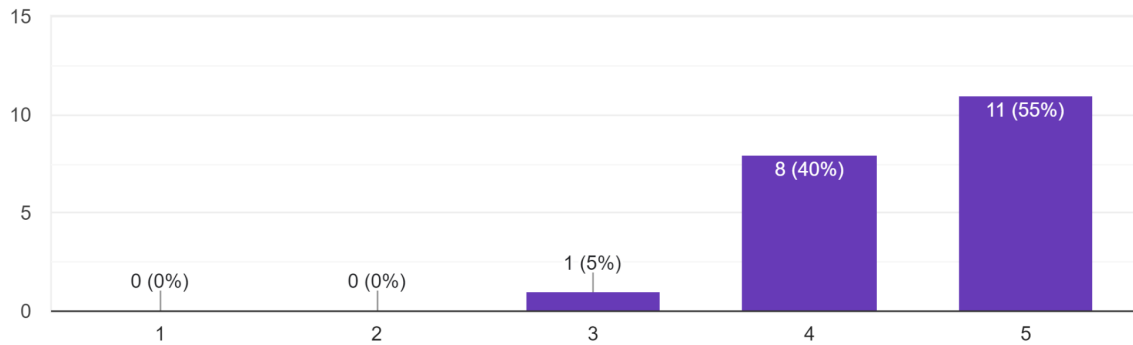


Fig.21 Function Entrances, from ProCoder Survey

What do you think of the UI design of this web page?

20 responses

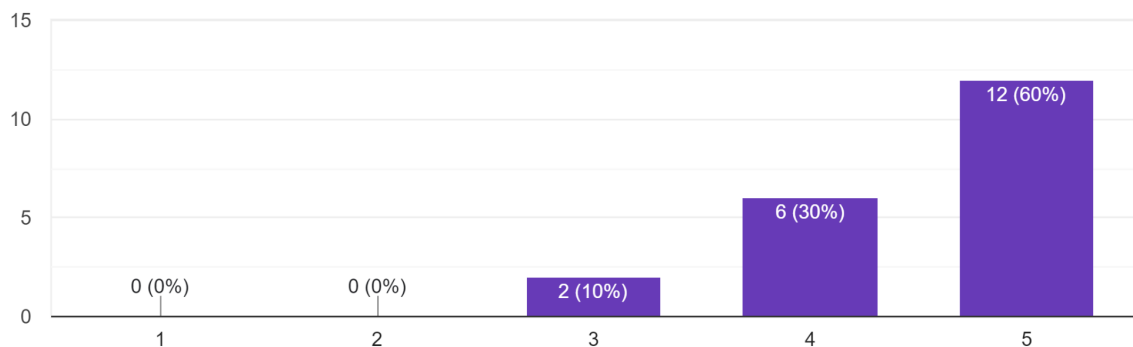


Fig.22 UI Design, from ProCoder Survey

Do you think the webpage is smooth to work with?

20 responses

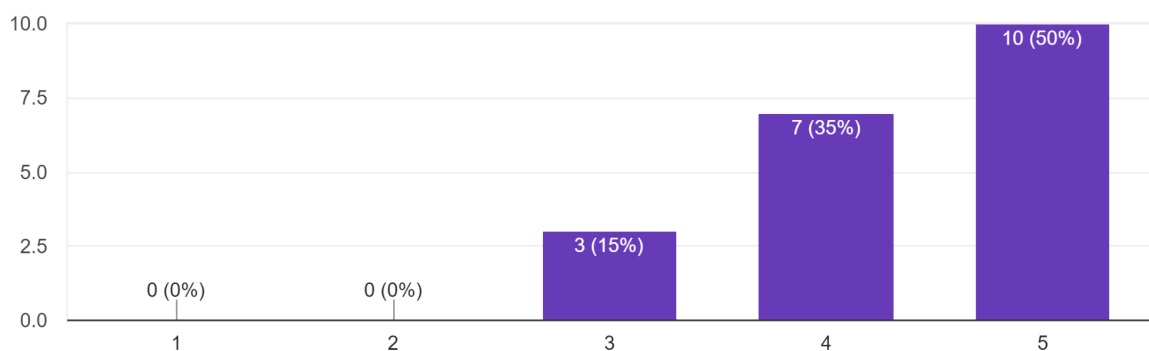


Fig.23 Webpage smooth, from ProCoder Survey

We looked through them and found some of our weaknesses. However, that is late in development, and we did not have time to iterate and improve based on exploratory testing results.

Most of the people who took this survey give 8 to 10 out of 10 for our website, as shown in the screenshot:

How would you rate our website?

20 responses

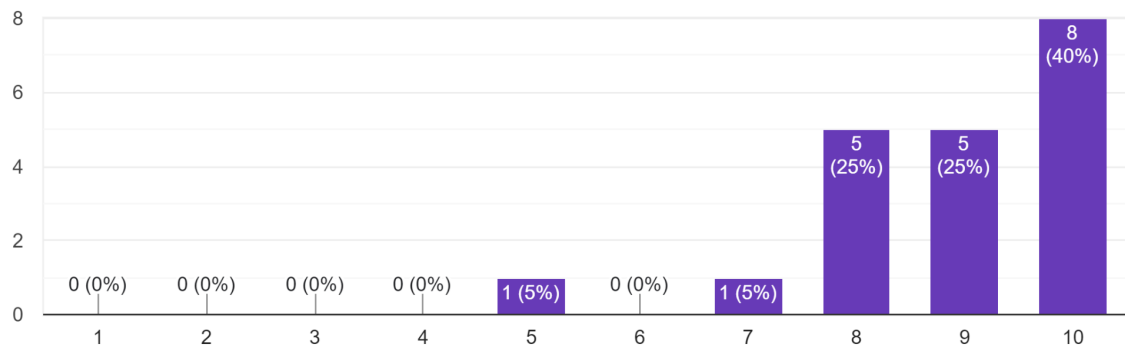


Fig.22 Rates, from ProCoder Survey

There is the feedback that points out some of our other weaknesses: No auto save!
No intellisense for coding

Why is leaderboard displayed before the discover cards? Could've be on the side or under the cards.

Fig.23 Feedback, from ProCoder Survey

We have not used automated testing for either back-end or front-end. Our lack of development experience results in an unstable web environment.

Future Work

Goals Not Achieved

Setting up the Jobe server is quite challenging for us. We spent more time than expected figuring out how it works, and we ended up having only Python fully functional, as our original plan was to at least support Java and C as well. Through the meetings with our client, we decided to defer this and focus on other functionalities of higher priority.

Our backend supports users to rate the solved question without needing to provide comment content and make the ratings a reference for users that are interested. However, we came up with new ideas, and it became part of a larger idea that we came up with late in development, which is still an idea due to time and knowledge limits.

Future Direction

The idea is that users who successfully solve this question will have multiple permissions on it. They can add more test cases. If the author of the question did not come up with comprehensive test cases, we could let the ones solving them add more to it. Our purpose is to encourage students to come up with more test cases, and the tests do not have to be based on a question they created. Authors will also be able to edit the test cases. They can rate this question. For people exploring questions, they will see the higher-rated questions first. Furthermore, if an author's average rate is too low, we will make this author's questions harder to get discovered. They can also post highlighted comments, maybe to explain the question better, or some hints that might help people. The highlighted comments will be ordered by the likes they received, so the most helpful ones will be on top. All users can add solving tags for a question. An example would be an author posting a question with a difficulty level of 'easy', but few users are able to solve it. For the ones that are aiming for an easy question, they can give this question a 'hard' tag with a failed submission. Then we will display the solving tags that this question received to the users that are browsing in a new feedback section.

In summary, the following is a list of future updates:

- Support more languages for the questions, including Java, C.
- Support better intellisense for solving the questions.
- Support auto-save for both solving and posting questions.
- Automation testing for back-end and front-end.
- Leaderboard badges with more details. The badges will show when and on which leaderboard.
- More user customisation options, such as profile decorations and avatars.
- Night mode for the whole website.
- Comparison between the author's sample solution and the user's submitted solution.

- Author changing the question details after posting.
- Users who solved the question will have permission to add more test cases.
- Users who solved the question can post highlighted comments in the discussion area to clarify the question description or leave some hints.
- Users who solved the question will be able to rate the question.
- Display higher-rated questions first. If an author's question average rate is too low, we will make this author's questions harder to get discovered.
- Make posted questions viewable from the profile, and make the profile viewable for anyone to access from leaderboard links.
- All users can add solving tags for a question, and the number of each tag this question receives will be shown on this question's detail page.
- Custom tags for posting and solving.
- Generate test papers with the user's choice of difficulty, tag, language and number of questions and choose the top-rated questions.
- More variations of the leaderboard will be introduced, like the most-rated questioner and most-liked communicator.
- Better calculation formula for the points, reward users who get positive feedback.
- Use similarity to detect posting the same questions repeatedly to avoid cheating in gamification.

Conclusion

Aims

Our project, ProCoder, is a code practice website that aims to allow users to publish programming exercises, attempt exercises created by other users and receive immediate feedback on their submissions. The website will benefit users by improving their ability to create test cases themselves. ProCoder provides authentication, social aspects, gamification, creation and storage of programming exercises, and can handle submissions to the exercises. It is made to highlight the importance of identifying problems and coming up with the most effective ways to solve them via programming.

Key findings

Throughout the development of the ProCoder, we discovered how to deploy Jobe on the server to compile and run the code input by the users. Monaco Editor has been introduced as our code editor to give users hints when they are editing the code.

Major outcomes

ProCoder has fulfilled the user's basic needs of post and practice programming exercises. ProCoder also introduced two new features: the forum system and the user achievements system. The unique forum and user achievement systems have differentiated ProCoder from other code practice websites. Users can freely discuss any posted questions and achieve various types of achievements depending on their level of participation in ProCoder. The user achievements system and forum system have encouraged users to participate more often in ProCoder. ProCoder has many advantages compared to other code practice websites, it has a unique ranking system that allows users to score depending on their level of participation, and the top ten users with the highest score will be displayed on the leaderboard.

Acknowledgements

We would like to express our gratitude to the following individuals for their expertise and assistance throughout all aspects of our project. Our completion of this project could not have been possible without the support of our instructor (Asma Shakil), tutor (Qiming Bao) and client (Paul Denny). Lastly, we would like to thank the anonymous who used our website and filled out the survey.

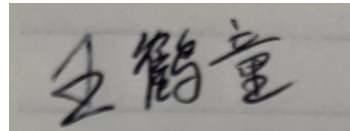
Table of Authorship

Sections	Team Members
Executive Summary	Laura Zhou, Lingxiu Cai
Introduction	Lingxiu Cai, Laura Zhou
Background	Laura Zhou
Specification & Design	Lingxiu Cai, Hetong Wang, Jinhua Fan, Fengyuan Ran
Implementation	Fengyuan Ran, Jinhua Fan, Hetong Wang
Results & Evaluation	Laura Zhou, Lingxiu Cai
Future Work	Laura Zhou
Conclusion	Hetong Wang, Lingxiu Cai
Gantt Chart	Fengyuan Ran

Signatures

Name	Signature
Lingxiu Cai	
Fengyuan Ran	
Jinhua Fan	
Laura Zhou	

Hetong Wang



Reference

[1] Richard Lobb. (September 2022). *Jobe*. GitHub.

<https://github.com/trampgeek/jobe>

[2] Alexandru Dima. (February 2022). *Monaco Editor*. GitHub.

<https://microsoft.github.io/monaco-editor/>

[3] Deqing Li, Honghui Mei, Yi Shen, Shuang Su, Wenli Zhang, Junting Wang, Ming Zu, Wei Chen. (April 2018). *ECharts: A Declarative Framework for Rapid Construction of Web-based Visualization*. Science Direct.

<https://www.sciencedirect.com/science/article/pii/S2468502X18300068>

Appendices

Links

- **Links to Project Repository:**

<https://github.com/uoa-compsci399-s2-2022/ProCoder>

- **Links to Website:**

<http://13.210.13.15>

- **Links to Demo Video:**

<https://youtu.be/INGIY8iQ18Y>

- **Links to Presentation Video:**

<https://youtu.be/IWCcWdKVTJ0>

Gantt Chart

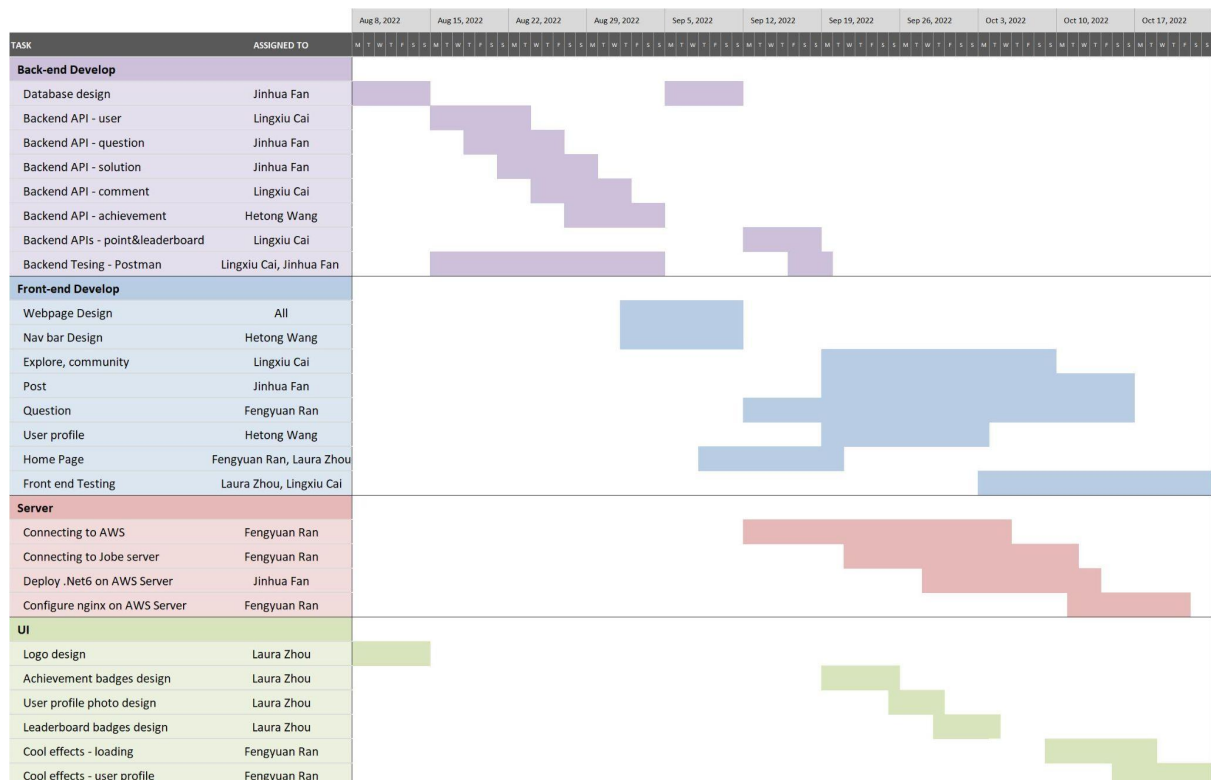


Fig. 24 Project Timeline