High-Level Design and Risk Assessment

CPSC 310 - Lab B - Group 2

Benjamin Ward
Cooper Webb
Andrew Park
Farbod Nematifar
Jeremy Gan

In this project, we aim to develop a web application that assesses novice-level code comprehension using generative AI (Ollama). To do this, we will have the students explain what sample code does in plain English, use this to generate new code based on their descriptions, and then test the code for functional-equivalence against pre-written tests.

Students will arrive at a home page, and be prompted to “login”, or go to a “sign-up” page, and once an account is established or logged in be taken to a “home page” where they can begin completing code interpretation problems (figure 1). The user-data will be saved in SQLite. The code interpretation problems will provide the students with a brief function, and a text-input for the student to describe their interpretation of the provided code’s functionality, which will be fed to Ollama rest API and generate the code based on the prompt which will be tested against the corresponding test-cases mapped to the specific function’s unique level ID. The program will then return all tests passed if all tests pass, or the specific tests that fail.

Functional Requirements:

1. Users must be able to log in and select a question to attempt.
2. Users must be able to read the question and have a space to write and submit their response.
3. Users must receive feedback from their attempt (successful or unsuccessful).

Non-Functional Requirements:

Security Measures:

- We will follow industry standards to ensure a high level of security for user data. This will include password hashing for user passwords, as well as parameterized queries and data sanitization to prevent SQL injection.

Performance Consideration:

- Although we anticipate a low volume of users (less than 10) at any given time, we plan to follow industry standards in our development and testing to produce scalable code.

Useability:

- Users will be able to submit responses in English. All provided coding will be given in Javascript.

Error Handling:

- We plan to implement try-catch blocks to catch and handle errors in our code.

Testing Strategy:

- We plan to implement unit tests using Mocha/Chai.

Technology Stack and Database Architecture
Technology Stack:
Front-end: React
Back-end: Node.js/Express
Database: SQLite
Languages: Javascript
Styling: CSS
Testing: Mocha/Chai
Generative AI: Ollama

Database Architecture:
Tables:

- Users: stores user specific information (**User ID**, username, email, password)
- Question: stores information on each attempted question (**Question ID**, **User ID**, question, status (pass/fail), total time spent, total errors)
- Attempt: an attempt at a question (**Question ID**, **User ID,** time spent, errors, hint, date/time, user answer, LLM generated code, results of test cases, pass/fail, [reason], score

User Stories:

User 1: As a student user, I want to improve as a coder and learn more about how functions work.

User 2: As a student user, I want to get the highest score I can as I work through the questions!

Data Flow

The data flow for our project can be viewed in **Figure 1.** Below is a brief description of how data flows throughout our project.

Login Screen: this is where users begin. If they haven't signed up, the transition to the "Sign Up" screen. If they have, they attempt to login, which checks their username/password against what's stored in our database. If this is successful, they progress to the home page.

Sign Up: this is where users create an account. Once created, the username/password are stored in our database.

Home Page: this is our "Landing Page" for the application. From here users can choose to attempt a problem or view the leaderboard.

Coding Interpretation Problem: users select a problem to attempt.

Coding Problem: here users will encounter a problem and a text box where they can input their answer. If successful, users will be informed that they successfully passed the problem, and will return to the Coding Interpretation Problem Screen. If unsuccessful, they will be given the option to try again. When taking the problem, we will record and store the following information to our database: Question ID, User ID, time spent, errors, hint, date/time, user answer, LLM generated code, results of test cases, pass/fail, [reason], score.

Application Flow:
Sign-Up/Login (allows users to sign up or login to app)
Home Page (allows users to select level, change settings, view leaderboard)
Coding Page (allows users to do a coding level)
Leaderboard (allows users to view high scores)

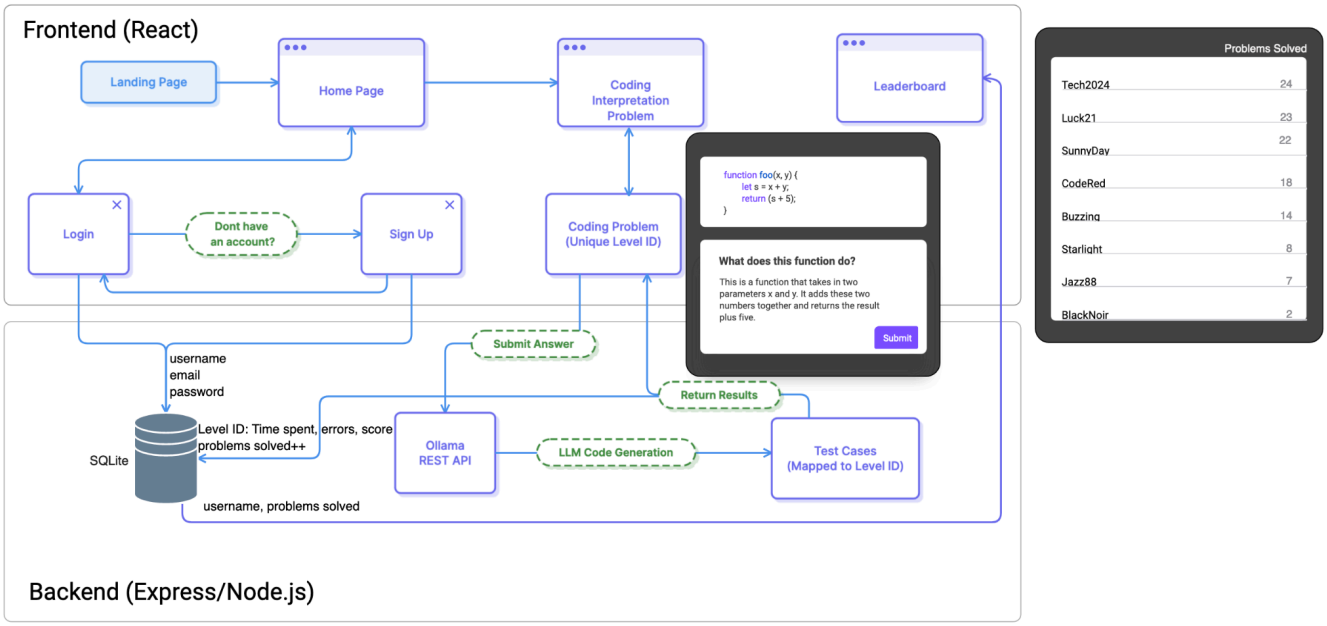

Figure 1: Summary diagram of the code comprehension application.

Unique Features

As unique features, our application will feature a global leaderboard and level performance data unique to each user. The global leaderboard will track the number of problems solved for the top 10 ranking users. This will incentivize users to complete more problems as they can engage in friendly competition with their peers. Additionally, the level performance data will allow users to track and review which problems and/or concepts they have struggled with. The level performance data will provide information on the level id (allows the user to review the problem), score, and time spent on each level.

These features will be implemented and realized using the SQLite database. As shown in the summary diagram (Figure 1), once results from a submitted solution return, they will be immediately mapped to a user in the database. The mapped information will include the level id, score, time spent, errors. If the user successfully completed the problem, they will have their "problems solved" field incremented within the database. When a user views the leaderboard tab as shown (Figure 1), the top 10 users with the highest number of problems solved will appear. This will be implemented by simply pulling the highest ranking users from the SQLite database.

Ethical Considerations

Building a web application that involves both gathering personal and sensitive information from our users along with storing persistent data about our users consists of dealing with significant ethical challenges which we have to navigate carefully by using guided principles such as those from PIPEDA and the ACM Code of Ethics.

Starting with PIPEDA, the first step is ensuring transparency with participants. They need to know exactly what data is being collected, why it's needed, and how it will be used. This means getting clear **consent** from participants before diving into data collection. Additionally, it is vital to stick to the principle of limited collection by only gathering the essential data required for the project and avoiding unnecessary personal details.

Data security can't be overstated. We need robust encryptions and **safeguard** methods for both data in transit and at rest, and strict access controls to keep unauthorized access and data leaks. Regular security audits are crucial to catch and fix vulnerabilities, ensuring that our protective measures stay strong. Transparency also means having a clear privacy policy and **openness** that participants can easily understand and giving them the **ability to access** and correct their data if needed.

The ACM Code of Ethics provides a comprehensive set of guidelines that align closely with the goals of this project. Several key principles from the ACM Code of Ethics are particularly relevant, with the first being to **contribute to society and human well-being**. The project should aim to improve the educational experience of students, helping them better understand and engage with programming concepts without causing harm. This involves designing the application to be user-friendly and supportive, reducing any potential stress or anxiety.

The principle to **avoid harm** emphasizes the importance of protecting participants from any form of damage. This includes ensuring their personal data is secure and that the feedback provided is constructive rather than discouraging. Implementing robust security measures and offering a supportive learning environment are crucial steps in this regard.

Another essential principle is to **be honest and trustworthy**. Transparency is critical in this context. The system's capabilities and limitations should be clearly communicated to participants, and the feedback they receive should be based on honest assessments of their performance. This helps build trust and ensures participants have a clear understanding of how their data and contributions are being used.

Respecting the work required to produce new ideas, as emphasized by the principle of giving **proper credit**, is another key consideration. This means acknowledging the use of open-source tools like Ollama and any auxiliary open-source APIs that are used in the project.
The principle of respecting **privacy** is fundamental. Handling participants' data with confidentiality and respect involves securing personal data and ensuring it is only accessible to authorized individuals. This respect for privacy extends to all aspects of data handling, from collection to storage and usage.

Finally, the principle to **honor confidentiality** reinforces the need to treat all information provided by participants as confidential. This includes their code explanations and any personal data collected during the project. Maintaining confidentiality helps protect participants' rights and fosters a sense of security and trust in the project.

Risks & Mitigation

A significant risk in this project is unauthorized access to personal data. To combat this, we plan to utilize strong encryption that should be used for data both in transit and at rest. Access controls must be robust, ensuring that only authorized personnel have access to sensitive information and our team plans to do this by regularly security audits are essential to maintain the integrity of these measures. One of the most common vulnerability attacks targeting access controls are SQL injection attacks and we plan to circumvent this by enforcing multi-factor authentication and strong password policies to help mitigate such risk.

There is also the inherent risk of students misunderstanding the code, leading them to incorrectly describe the code and providing inaccurate feedback to the module. To mitigate this, it is vital to provide detailed instructions and examples. We plan to offer additional support resources such as tutorials and FAQs that can further aid students in understanding how to appropriately work the module. A feedback mechanism will also be in place to allow students to ask questions and receive clarifications, ensuring they have the support needed to accurately describe the code when providing feedback.

Furthermore, another risk that may not be apparent at first is the potential stress and anxiety that a student might face especially when a feature such as a leaderboard is in place. To minimize stress and anxiety for students, the system will be designed to include an option to opt-out of public visibility from the platform therefore eliminating any concerns with sharing their results in the public domain. The system will also be designed to provide encouraging and supportive feedback to help encourage students instead of putting them down.

Finally, there is also the technical risk of having too many queries being made to the LLM from a single end-point/source (our web application in this context) at once and therefore leading to being rate limited and therefore halting the entire operation of our web application. Our team is also considering this duration of the development of the project and will be looking into ensuring that we look into the limitations that Ollama has for its API calls and make modifications to the project to account for user scalability.

Stakeholders

- *Students*: participate in code comprehension exercises
- *Teachers*: evaluate students' performance in code comprehension and monitor their progress
- *Teaching Assistants*: provide assistance in managing the system
- *Researchers*: study the effectiveness of the system in improving code comprehension performance
- *Developers*: implement new features and fix bugs found in the system
- *Instructor*: facilitate deployment and maintenance of the system

Datasources:

Datasource	*Purpose/Usage*	*Method of Obtaining/Generating*
Code Samples	● Provide students with code sample to describe and comprehend in plain English. ● Code sample will be used by students to provide a plain English description and is the basis for comparison with the generated code returned from the LLM	Created by the development team.

Test Cases	• Verify the functional equivalence of the generated code from the LLM against the original code sample. • Test cases will be executed against both the original code sample and the generated code to determine whether they are functionally- equivalent.	Created by the development team.
Student Responses (EiPE)	• To collect plain English description of what the provided code sample does. • Used to craft a prompt for the LLM that will then generate code based on the description.	Obtained through the web application interface; students input their descriptions directly into the interface.
LLM Ollama Generated Code	• Compare against original code sample to check for functional equivalence.	Generated using the open-source LLM Ollama via its JavaScript API.
Test Case Results	• Provides feedback on correctness of the generated code. • Present tests cases that pass/fail, which can then be used to make changes to their plain English description if necessary.	Obtained by running test cases on the generated code.
Student Feedback	• To understand changes students make to their plain English description when their initial description does not result in functionally equivalent code.	Obtained through the web application interface; students input their feedback directly into the interface.
Student Attempt Data	• Stored to evaluate student's comprehension performance and used for later analysis.	Obtained from each interaction with a code sample in the web application interface; includes student responses, generated code, and test results.
Ollama Language-Learning Model (LLM)	• Generates code based on a student's plain English description.	Downloaded from Ollama official repository and deployed locally in a Docker container.