Interation Design Documents

App Design Summaries:

Code Critiquer is a static analysis tool for finding common anti-patterns in user-submitted code snippets. Similar to grammar or plagiarism checkers in which users can submit either their full source code or erroring segments, and have the app respond with feedback pertaining to the found anti-pattern (bugs, errors, etc.) found within the given code.

Systems Overview:

Stakeholders and Users:

- Leo Ureel(Stakeholder)
    - They are the main person we go to for how the app should be built or what kind information we will need
- Novice Programmers(Main)
    - The app is being built around on the idea that people who are just getting into programming will create anti-patterns(errors,warnings,etc)
- Language Learners(Secondary)
    - People who have some idea about coding, but are trying to learn a new language and will be likely to encounter anti-patterns
- Instructors(Secondary)
    - People who are trying to clean and optimize code that can benefit from resolving anti-patterns, or implementing efficient code

Personas:

Personas Persona 1: Emily- The Novice Programmer

- Demographics: 19 years old, sophomore in computer science at a state university.
- Background: Emily has just declared her major in computer science. She is currently enrolled in CS1122 and is enthusiastic but sometimes struggles with understanding complex coding concepts.
- Goals:
    - To submit code for review and understand feedback to improve her programming skills.
    - Topassher courses with a good understanding of programming fundamentals.
- Challenges:
    - Emily sometimes feels overwhelmed by debugging and identifying issues in her code.
    - She is not always confident in her solutions and seeks reassurance and guidance.

Persona 2: Alex- The Programming Tutor

- Demographics: 26 years old, graduate student and tutor for undergraduate programming classes.
- Background: Alex has been coding for over 7 years and is passionate about teaching others. He tutors students in CS1121 and CS1122.
- Goals:
    - To provide clear, actionable feedback to his students.
    - To help students learn from their coding mistakes and improve over time.
- Challenges:
    - Alex needs to balance his time effectively between many students.
    - He wants to provide personalized feedback without spending excessive time on each student's code.

Persona 3: Sofia- The Language Learner

- Demographics: 34 years old, a professional looking to transition into a software development role.
- Background: Sofia has experience in project management but wants to become more hands-on with coding. She is self-learning Java to potentially shift her career.
- Goals:
    - To learn Java effectively and apply best practices from the beginning.
    - To understand the common pitfalls new programmers might face.
- Challenges:
    - Sofia finds it difficult to get feedback on her self-taught projects.
    - She struggles with knowing if she's "doing it right" without a structured learning environment.

Persona 4: Dr. Liam- The Course Instructor

- Demographics: 45 years old, a professor of computer science with a focus on software engineering.
- Background: Dr. Liam has been teaching for over 15 years and is always looking for innovative tools to improve the learning experience.
- Goals:
    - To integrate effective tools like "Code Critiquer" into his curriculum.
    - To track the progress of his students and the effectiveness of his teaching methods.
- Challenges:

- Dr.Liam is often skeptical of new tools and their long-term benefits.
- He needs to ensure that any tool he uses aligns well with his course objectives and provides real value to his students.

Environment:

The environment that the app would be used is a school environment by students and professors as it will be used to help people improve their coding skills and find anti patterns(errors, bugs, and fails) in their code.

Use scenario description:

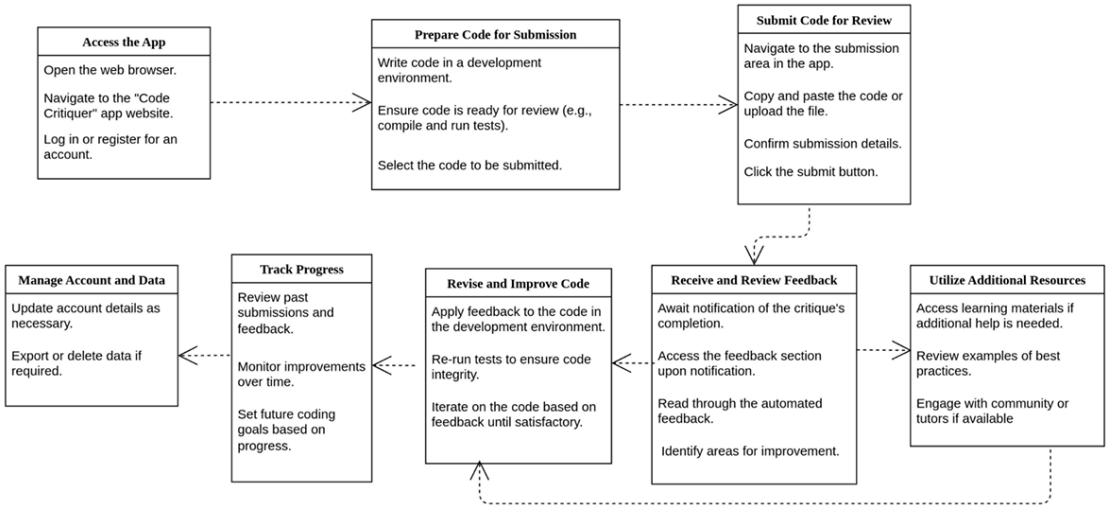
Nominal:

A student writes a program in Java for one of their classes. The program encounters runtime errors that the student doesn't know how to resolve. Their IDE does not give them any useful information. Without logging in, the student uploads their .java file to the app and submits it for critique. After a moment, the app loads a page showing the student their code, with blocks of text displayed at the lines where anti-patterns were discovered. The student downloads the critique file containing the information shown in the app. When they close the app, their information is discarded.

Error:

In this showcase of past projects, delve into The student uploads their project code in a different language then initially expected by the project's description. The code critiquer errors out and informs the user that this is the wrong expected language, and they must submit in the expected language in order to receive any credit for their program.example that underscores my versatility and commitment to excellence. From concept to completion, witness how I navigated complexities and contributed to the success of this [industry/field] initiative

Hierarchical Task Analysis:



The Hierarchical Task Analysis (HTA) for the "Code Critiquer" app details a structured process through which users can achieve the primary goal of submitting code, receiving feedback, and enhancing their coding skills. Here's a summary of the HTA, outlining user goals and the steps to achieve them:

1. **Access the App:** The journey begins with users accessing the app by opening a web browser, navigating to the app's website, and logging in or registering. This step is fundamental to establish a user's presence in the system and prepare for code submission.
2. **Prepare Code for Submission:** Users write and prepare their code in a development environment, ensuring it's ready for review. This includes compiling and running tests to verify the code's functionality before selecting it for submission. This step is crucial for ensuring that the code is in a suitable state for critique.
3. **Submit Code for Review:** Once the code is prepared, users navigate to the submission area within the app, where they can either copy and paste their code or upload the file directly. After confirming submission details, the code is submitted for review. This action initiates the critique process.
4. **Receive and Review Feedback:** Users then await notification of the critique's completion. Upon receiving this notification, they access the feedback section and carefully read through the automated feedback provided, identifying specific areas that need improvement. This step is pivotal for understanding the app's insights into their code.
5. **Revise and Improve Code:** Based on the received feedback, users apply the suggested changes to their code in the development environment and re-run tests to ensure the code's integrity. This

iterative process continues until the user is satisfied with the improvements, reflecting a cycle of learning and application.

6. **Utilize Additional Resources:** If users require further assistance, they can access additional learning materials, review best practice examples, or engage with the community or tutors. This step supports continuous learning and problem-solving beyond the automated feedback.
7. **Track Progress:** The app allows users to review their past submissions and feedback, enabling them to monitor their improvements over time and set future coding goals. This feature fosters a sense of progress and goal-oriented learning.
8. **Manage Account and Data:** Finally, users have the ability to manage their account details. They can also export or delete their data if required. This step ensures users have full control over their personal information and the data generated through their use of the app.

Overall, the HTA of the "Code Critiquer" app demonstrates a comprehensive and user-centric approach to learning and improving coding skills. From initial access to ongoing learning and data management, each step is to be designed to facilitate an effective and engaging user experience.

Database Schema:

List of Domain Classes:

- User
- Critique
- Submission

Domain: User -

- userId: long - unique identifier for this user.
- submissions: Submission[] - list of submissions submitted by user

Domain: Critique -

- pattern: String - a regular expression defining the pattern or anti-pattern to search for.
- patternType: ‘Pattern’ | ‘Anti-Pattern’ - the type of pattern being searched for
- critiqueText: String - the text to be displayed to the user if the pattern is found
- severity: ‘PASS’ | ‘FAIL’ | ‘WARNING’ | ‘ERROR’ | ‘COMMENT’ - the severity of the critique
- title: String - the name of this critique
- language: String - the language this critique is for.

Domain: Submission - information about the submission and byte data for the code snippet

- dateCreated: Date - date of submission.
- fileData: String - the raw text of the submission.
- language: String - the language of this submission.
- critiques: Critique[] - a parallel list of critiques
- critiqueLineNums: int[] - a parallel list of line numbers for the critiques found.
- author? : User - (optional) the user that submitted this file

Existing domains used: Pattern, PatternExemplar, PatternTest