Product Vision and Scope

Product Vision

This project aims to create an AI-assisted programming platform that helps students learn by giving them structured guidance instead of direct answers. The goal is to improve students' problem-solving and coding skills while maintaining academic integrity.

Many current tools either solve problems for students or give only basic feedback. In contrast, this system uses artificial intelligence as a learning assistant. It helps students understand their mistakes, write better code, and think through problems, but students are still responsible for their own solutions.

For instructors, the platform is designed to reduce repetitive work by offering AI-supported feedback and help with evaluation, but final grading decisions remain with the instructor. The project's main aim is to show that artificial intelligence can be used responsibly and in a balanced way in programming education.

Problem Statement

In many programming courses, students often rely on external tools that give them full solutions, sometimes without understanding the reasoning behind them. This situation can hinder students' development of strong problem-solving and analytical skills. At the same time, because instructors have to review a large number of assignments, it becomes difficult to provide detailed and personalized feedback to everyone. Most existing platforms use automatic checks for accuracy but offer very little support for learning-focused guidance. This creates a gap between independent problem-solving and helpful instructional support. This project addresses the lack of a system that can guide students in a controlled, educational way while still protecting academic integrity.

Scope Boundaries

- The system is developed as a **web-based platform** accessible through standard desktop browsers.
- The platform supports **C, C#, C++, Python, and Java**, suitable for undergraduate courses.
- Artificial intelligence is used **only to give guidance and feedback**, not to generate complete solutions.
- Student code is checked with **automated test cases** in a secure environment.
- The platform is designed for use with **assignments, lab exercises, and practice sessions** in academic settings.
- Instructors are still essential for **final evaluation and grading decisions**.

- The platform supports both cloud-based and locally hosted AI models, using **OpenWebUI** as a unified interface for managing and interacting with different LLM backends.

Deployment & Containerization

- The platform uses Docker containers to make it modular, secure, and easy to deploy. Each main part of the system, like the frontend, backend API, database, AI services, and code execution sandbox, runs in its own container.
- With this container setup, the system can be deployed in the cloud or on-premise without changing its architecture. It also lets you easily turn AI features on or off for different situations, like practice or exam use.
- This way of deploying the platform makes it secure, flexible, and easy to maintain, which helps support the project's educational goals.

Core Workflow

- Instructors create programming assignments and define submission deadlines.
- The system supports **AI-assisted generation of question variations** for instructors to create similar or alternative problems.
- Assignments are made available to registered students through the platform.
- Students write and submit their solutions using the browser-based development environment.
- During problem solving, students may request **AI-generated hints and code-level feedback**, without receiving complete solutions.
- Submitted code is executed in a **secure sandbox environment** using automated test cases.
- The system generates **AI-assisted feedback and scoring suggestions** for instructors based on student submissions.
- Instructors review the AI-generated suggestions and perform the **final assessment and grading**.
- Instructor-side AI interactions are handled through **OpenWebUI**, which connects the system to different LLM backends.

Out of scope

1. AI solving problems for the student

This project does not aim to have artificial intelligence solve programming problems for the student. The learning process is expected to remain student-driven.

What is not included here are fully generated code solutions or features that allow the AI to automatically complete assignments.

2. Continuous AI tutoring or chat-based interaction

The AI component is not designed to behave like a personal tutor that interacts with the student at all times. Its role is limited and contextual.

There is no free-form chatting, no voice interaction, and no ongoing conversation with an AI assistant.

3. Real-time instructor involvement during coding

Instructors are not expected to intervene while students are actively working on their solutions. Their role begins after submissions are made.

For this reason, live code reviews or instant messaging between instructors and students are not supported.

4. Mobile-first usage scenarios

The platform is designed for use with desktop browsers. Mobile use is not the primary purpose of the project.

Therefore, local mobile applications and writing or sending code directly from mobile devices are outside the scope.

5. Social or collaborative learning features

This system is mainly designed to support individual practice and assessment. The emphasis is placed on personal learning progress rather than on group-based or collaborative activities.

For this reason, features commonly found on social learning platforms, such as discussion forums, student-to-student messaging, collaborative coding sessions, or public ranking systems, are intentionally excluded from the scope of this project.

6. Advanced monitoring and proctoring technologies

Although academic integrity is an important consideration, this work does not attempt to implement complex monitoring solutions.

Technologies such as biometric verification, eye-tracking, or full online proctoring systems are beyond the scope of this project.

Feature Breakdown

Web-Based Programming Platform

Browser-Based IDE

- Students solve programming assignments directly in the browser without requiring local setup.
- Supports multiple programming languages defined by the instructor.
- Provides real-time syntax highlighting and basic error feedback.

Assignment Management

- It is possible for teachers to create, edit, and assign programming problems with deadlines.
- The system supports different modes, such as practice, homework, and exams.
- Submissions are saved and tracked automatically.

Code Execution & Sandbox

- The execution of the student code is conducted within a secure and isolated environment to prevent unauthorized system access and ensure a fair evaluation process.
- The execution results, which include the output, runtime errors, and time limits, are reported to both students and instructors.

AI-Guided Feedback (Student Side)

- It is possible for students to submit requests for hints, explanations, or feedback on the quality of their code.
- AI never gives you the whole solution or code that's ready to submit.
- Feedback focuses on:
    - Logical mistakes
    - Algorithmic approach
    - Code readability and best practices

Teacher AI Tools

- The system automatically generates similar problem variations to reduce plagiarism.
- The system suggests rubric suggestions based on problem complexity using AI.
- The system recommends preliminary scores based on the structure and correctness of the code.

AI Model Interface (OpenWebUI)

- OpenWebUI acts as an interface layer between the platform and the underlying language models.
- It enables flexible integration with both local LLMs (e.g., via Ollama) and cloud-based AI services.
- The platform uses OpenWebUI to manage model selection, prompt routing, and controlled AI interactions.
- This design allows institutions to deploy the system on-premise when data privacy or cost constraints are critical.

Submission Analysis & Reports

- Tracks how students are doing over time.
- It shows the most common mistakes.
- This helps teachers spot students who are finding learning difficult at the start of the course.

NLP Engine

Purpose

The NLP Engine is responsible for understanding, analyzing, and responding to requests related to programming in natural language in a manner that respects academic integrity.

Input Processing

- Processes:
    - Student questions about their own code
    - Code snippets submitted for the purpose of soliciting feedback.
    - Teachers request help with grading and creating questions.

Prompt Control Layer

- Makes sure that AI replies keep to the guidance.
- Filters prompts to prevent:
    - Direct solution generation
    - Copy-paste ready answers
- Enforces educational limits, such as:

        - Asking questions that suggest the answer
        - Showing different ways to do something instead of the finished code.

Code-Aware Analysis

- It combines NLP with a kind of code analysis that doesn't change.
- Understands:
    - Control flow
    - Common patterns in algorithms
    - Common beginner mistakes
- Shows the student their feedback, based on their submission.

Role-Based Response Generation

- Student Mode:
    - Hints, reminders, conceptual explanations
- Teacher Mode:
    - Problem variations
    - Rubric suggestions
    - Submission comparison insights

Model Flexibility

- Supports both cloud-based and local LLMs (e.g., Ollama).
- The system can be deployed on-premise by institutions should the necessity arise.
- OpenWebUI is used to operationalize this flexibility by providing a unified interface for managing different language models.

Use Cases

Student – Learning Mode (Beginner)

As a student of programming, I would prefer to receive hints, rather than complete solutions, to develop my problem solving skills without resorting to copying answers.

Student (Exam Mode)

As a student taking an exam, it is essential to have a controlled environment with limited AI help. This helps make sure that the exam is fair and impartial.

Instructor – Assignment Management and Grading Support

As a programming instructor, I want to create assignments and receive AI assisted grading suggestions so that I can reduce my workload while maintaining evaluation quality.

Instructor (Assessment Design)

As a teacher, I want the AI to generate similar versions of programming questions so that I can minimize cheating and reuse assignments safely.

Teaching Assistant – Error Analysis

As a teaching assistant, I want to analyze common mistakes across submissions so that I can address these issues during lab sessions.

System Administrator – Local Deployment

As an administrator, I want the platform to support local AI deployment so that sensitive academic data does not leave the institution.

Frontend

Core User Experience

The central experience is a single workspace with two tightly connected areas:

- Code Editor: where students write or paste code, view formatting, and select relevant parts of the code.
- AI Chat Panel: where students ask questions, receive explanations and suggestions, and iterate on solutions while keeping code context visible.

A typical workflow is:

1. The student writes or pastes code in the editor.
2. The student highlights a snippet or references an error/output.

3. The student asks the AI for help (e.g., explanation, debugging steps, improvement suggestions).
4. The AI responds in a streaming manner so the student can start reading immediately.
5. The student applies the guidance, revises code, and repeats as needed.

Key Frontend Capabilities

- Editor–Chat Integration: the chat can be grounded in the current code context (selected snippet, file content, or user-provided error text).
- Streaming Responses: AI answers appear progressively to reduce perceived waiting time and support step-by-step guidance.
- Readable Technical Output: code blocks, error messages, and structured explanations are displayed clearly and consistently.
- Session-Aware UX: users can keep their session state (e.g., authentication state, ongoing conversation) for a smooth return experience.
- Safe Content Rendering: user- and model-generated content is sanitized before rendering to minimize risks such as script injection.

UI/UX Principles

- Low distraction, high focus: keep the workspace clean and predictable.
- Fast feedback loops: minimize clicks and context switching.
- Beginner-friendly clarity: prioritize explanations that teach, not just "give the final answer.
- Consistency: a stable design system and consistent components for forms, messages, and layout.

Integration Points

The frontend communicates with backend services for:

- Authentication and session validation
- Sending user prompts and receiving AI responses (including streaming)
- Retrieving and displaying conversation history or user-related data (if included in the MVP scope)

Quality Goals

The frontend prioritizes:

Performance: smooth typing and scrolling in the editor, responsive UI interactions

Reliability: graceful handling of network errors, retries where appropriate, and clear user feedback

Maintainability: modular components and a scalable structure to support future features and roles

Backend

API Functions

Authentication & Users Handles account creation, secure login, and user identity verification. It utilizes **JWT (JSON Web Tokens)** for stateless authentication. It implements **Role-Based Access Control (RBAC)** to distinguish between "Student" and "Teacher" permissions, ensuring that assignments and administrative functions are protected and managed correctly.
AI Mentor (Hint Generation & Validation) The core pedagogical feature of the platform, providing guidance to students without revealing direct solutions.

- **Hint Validation Logic:** To prevent the AI from leaking the full solution, the backend implements a "Tutor/Student" simulation. It uses a secondary AI model to validate that the generated hint is helpful but does not provide the complete answer.
- **Contextual Flow:** Relays the student's current code and compiler errors to the AI service to generate targeted, personalized feedback.

AI Teacher Assistant (Question Variation Generation) Manages the automated generation of new programming problems to reduce instructor workload.

- **Structural Code Analysis:** The backend uses an **Abstract Syntax Tree (AST)** parser to structurally analyze the teacher's base code, identifying specific logical components such as loops and conditions.
- **Semantic Generation:** Relays this structural data to the AI service to generate high-quality problem variations or conceptual questions based on the code's logic.
- The backend communicates with language models through **OpenWebUI**, ensuring consistent prompt control and role-based AI behavior.

Code Execution (Sandbox Integration) Manages secure and isolated code execution via the Judge0 API to provide real-time feedback.

- Execution Management: Relays code snippets and hidden test cases to the isolated sandbox environment.
- Result Processing: Formats and returns stdout, stderr, execution time, and memory usage to the student.

Problem & Assignment Management Manages the logic for accessing and assigning programming tasks.

- Instructor Controls: Provides endpoints for creating problems and defining hidden test cases.
- Student Access: Fetches assigned problems and metadata from the Database API to serve to the Frontend.

Submissions & Progress History Tracks student attempts and performance metrics over time.

- Version Tracking: Manages the history of code attempts for each problem, allowing students to view and compare previous versions.
- Data Pipeline: Aggregates raw data (success rates, common error types) to be consumed by the statistics dashboard.

Data Coordination Coordinates communication with the dedicated Database API to ensure data persistence.

- Structured Data: Relays user profiles and problem definitions to the relational database service.
- Activity Storage: Manages the storage of flexible data, such as AI chat logs and raw code submissions, in the NoSQL layer.

Health Check Monitors the status of all external dependencies (Judge0, AI Service, and Database API). It provides centralized logging to track API timeouts or service failures, ensuring system reliability.

Backend Stack Summary

- Runtime: Node.js
- Framework: Express.js
- Communication: Axios (for connecting to Database, AI, and Judge0 APIs)

- **Security:** JWT (Stateless Authentication) & Bcrypt (Password Hashing)
- **Code Analysis:** AST Parser libraries (for structural logic extraction)

Database

Database Architecture

We will use an SQL relational (PostgreSQL) as the core SQL database because it will enable us to have high data integrity when storing academic records. The flexible storage of the project is layered on top of the structured layer to allow for the flexibility of the generated content.

- **Identity & Access Management:** This will be a repository of all educator and student accounts and will enforce permissions and access rights for each user and will have robust authentication methods.
- **Curriculum & Problem Assets:**
    - **Core Task Bank:** This is a central library for coding challenges that include specific technical restrictions, grading benchmarks and other constraints for each challenge.
    - **AI-Driven Iterations:** This is a data store for problem permutations that are created by Large Language Models (LLM) to create a unique testing experience for each student.
    - **Grading Frameworks:** These are standardized rubrics and AI-suggested evaluations of the metrics that will be used to evaluate the consistency of the grading standards.
- **Student Analytics & Performance:**
    - **Versioned Code Snapshots:** These are snapshots of a students' version history of their code so that they can see how their code has progressed over time.
    - **Submission Tracking:** This is a log of all final submissions made by students, including the submission date and timestamp along with whether the student met the deadlines.
    - **Impact Metrics:** These are comparative datasets that show how well AI intervention is improving student performance.
- AI Mentorship Logging:
    - **Interaction History:** This is a complete record of all hints and feedback given to the student by the AI mentor.
    - **Prompt Configuration Registry:** This is a data store of the various template prompts that are used to keep the AI in a 'mentor' role and prevent the AI from giving direct answers.

- System Integrity & Deployment:
    - **Testing Mode Variables:** These variables are flags that can turn AI features on or off to protect against cheating during live exams.
    - **Infrastructure Metrics:** These are metadata focused on managing containerized environment so that the system remains available and responsive under varying server loads.
- Database Technologies:
    - **PostgreSQL**: Used for containing both SQL and NoSQL, providing ACID compliance.
    - JSONB: flexible storage for AI prompts, metadata, and evaluation rubrics within the relational schema.