Model

The model component of the project is responsible for the intelligent analysis of the code. Instead of only checking syntax, the model focuses on identifying programming mistakes, coding style issues, and possible improvements.

In this project, the model is implemented using a rule-based approach combined with Abstract Syntax Tree (AST) analysis. AST helps in understanding the structure of the source code by breaking it into nodes such as functions, loops, variables, and imports. This structural understanding allows the model to detect issues like unused imports, long functions, and inefficient coding patterns.

In addition, simple AI logic is used to generate optimization and best-practice suggestions. These suggestions are designed to be student-friendly and easy to understand. The model does not execute the code; instead, it analyzes the code statically, making it safe and efficient. Overall, the model acts as an automated reviewer that mimics how a human evaluator would analyze code.

Python

Python is used as the **core implementation language** in this project. It handles code analysis, logic execution, and integration between different components of the system.

Python's built-in **AST module** is used to parse student code and convert it into a tree structure that represents the program's syntax. By traversing this tree, the system can identify syntax errors, unused imports, long functions, and other structural issues without running the program.

Python also manages the **analysis workflow**, including accepting code input, processing it, generating feedback, and displaying results. Its simplicity and extensive library support make Python ideal for building AI-driven systems. Error handling mechanisms are used to ensure the system works smoothly even when incorrect code is submitted.

Thus, Python serves as the backbone of the project by implementing the analysis logic and coordinating interactions between the model and the database.

SQL

SQL is used in the project for **data storage and management**. It helps in maintaining a structured record of student submissions and the feedback generated by the system.

A relational database is designed to store information such as the submitted code, detected errors, style issues, optimization suggestions, and submission timestamps. SQL queries are used to insert new records and retrieve previous submissions when required.

Using SQL allows the system to maintain a **history of evaluations**, which is useful for tracking student progress and for instructor review. It also ensures data consistency and efficient retrieval. In this project, SQL enables persistent storage, making the system scalable and suitable for real academic environments.

What Work Does This Project Do?

This project works as an **automatic code reviewer** for student programming assignments.

Instead of a teacher manually checking each program, the system allows a student to **submit their code** and then automatically analyzes it. The project checks whether the code has any **syntax errors**, **logical mistakes**, or **coding style issues**. It also gives **suggestions to improve code quality and readability**.

The system uses **Python's AST** to understand the structure of the program without executing it. Based on this analysis, it detects issues such as unused imports, inefficient loops, and poor coding practices. After analysis, the project generates **clear and simple feedback** that helps students understand their mistakes and improve their coding skills.

All the submitted code and feedback are **stored in a database**, which makes it possible to keep a record of submissions and track improvements over time. This reduces the workload for instructors and provides instant feedback to students.