Technical Debt Demo

Project Name: Pylint-based Bug Detection and Code Style Modification Tool

**1. Overview**

This project aims to provide users with a static code analysis tool based on Pylint that automatically detects code issues in Python projects, offering a visual interface and a GPT interaction module. As the project progresses, the team has accumulated some technical debt that may impact maintainability and user experience.

**2. Identification of Technical Debt**

**Code Quality:**

Some modules lack documentation and comments, making it difficult for new members to understand.

There is a lack of unified coding style between different modules, leading to poor code readability.

**Functionality Implementation:**

The visualization interface is not fully implemented, and some data presentations are not intuitive enough.

Compatibility issues exist between the GPT interaction module and other modules.

**Testing Coverage:**

The current project lacks unit tests and integration tests, making it hard to ensure functionality stability.

Not all potential input scenarios are covered, making it difficult to detect potential bugs.

**Database Design:**

The database table structure is poorly designed, with redundant data affecting query performance.

The user login system lacks security validation and data encryption measures.

**3. Impact Analysis**

Impact on Development: The lack of documentation and unified coding style makes it hard for new members to onboard, increasing development time.

Impact on Functionality: Compatibility issues with the visualization interface and GPT module may lead to poor user experience and user attrition.

Impact on Maintenance: The lack of testing coverage and reasonable database design complicates code maintenance, exacerbating the accumulation of technical debt.

**4. Priority Ranking**

Code Quality: Improve code documentation and comments, and unify coding style.

Testing Coverage: Establish unit tests and integration tests to ensure functionality stability.

Database Design: Optimize database structure and enhance user security.

**5. Solutions**

**Code Quality:**

Within the next two weeks, write documentation and comments for each module and refactor the code.

Standardize coding style according to PEP 8 guidelines.

**Testing Coverage:**

Develop a testing plan, aiming to complete unit and integration tests within a month.

Use coverage tools to check test coverage, ensuring that critical functionalities are tested.

**Database Design:**

Redesign the database structure to eliminate redundant data and strengthen security measures.

Implement user data encryption and security validation mechanisms.