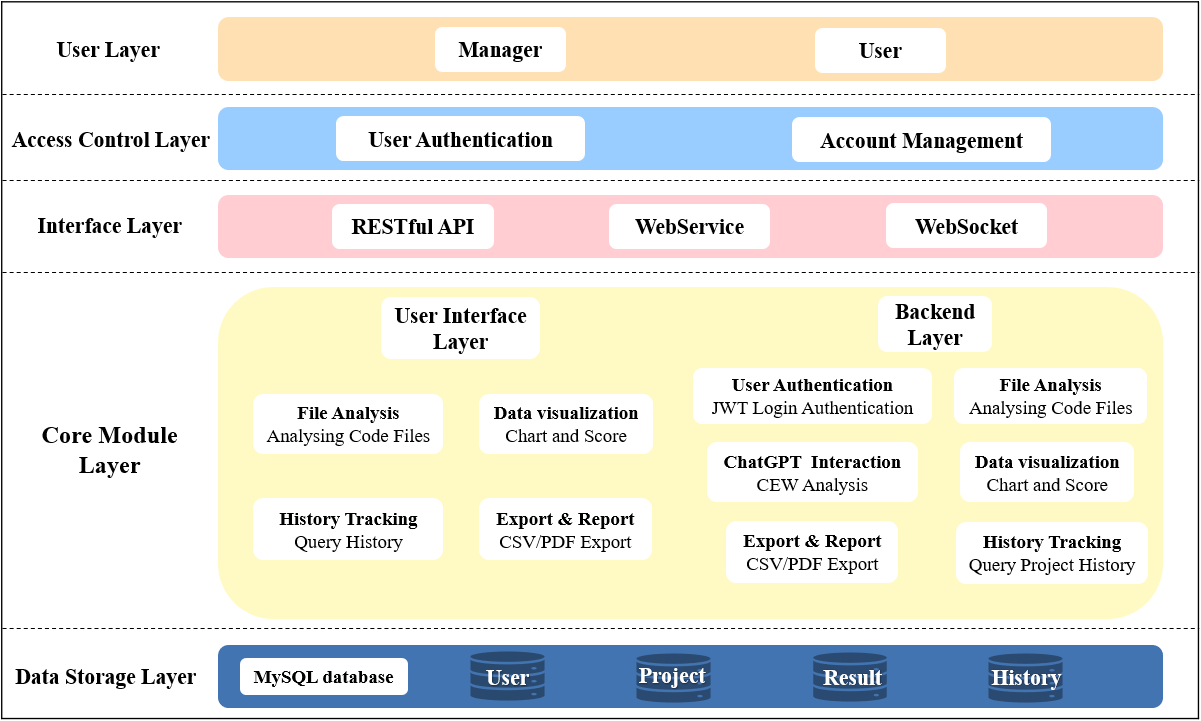
Architecture Design Document

1.System Architecture

Users upload Python project files via the Vue.js frontend, which communicates with the Django backend through RESTful APIs and WebSocket for tasks like authentication, file uploads, and history retrieval. The backend invokes the File Analysis module, integrating Pylint to analyze code and generate CWE issue classifications and code quality scores. These results are passed to the ChatGPT module, which uses the OpenAI API to generate Chinese modification suggestions and optimized code. The Business Modules interact with the MySQL database through the Persistence Layer (Hibernate) to store user, project, and historical data, as well as retrieve system operation logs from the log database. Finally, the frontend dynamically visualizes analysis data using ECharts and supports exporting results in CSV or PDF format. This efficient interaction among components ensures high performance, security, and a seamless user experience.



2.Design Rationale

This architecture adopts a layered design. The frontend leverages Vue.js to create dynamic and interactive interfaces, while ECharts is integrated for data visualization, including line charts, bar charts, and pie charts, offering an intuitive display of analysis results. The backend, built on Django, communicates efficiently with the frontend via RESTful APIs, supporting core functionalities such as user management, file analysis, and log management. Pylint is integrated to perform static code analysis on uploaded Python projects, identifying CWE issues (formatting, warnings, errors) and calculating code quality scores. WebSocket provides real-time feedback, ensuring immediate interaction responsiveness.

At the data layer, MySQL serves as the primary database for storing users, projects, and analysis records, supplemented by Redis as a caching layer to optimize query performance and reduce direct database access. The persistence layer employs Hibernate to encapsulate database operations, simplifying complex data management logic and improving operational efficiency. The access control layer utilizes JWT for authentication and authorization, ensuring secure and reliable access control.

How can this architecture meets non-functional needs:

In terms of performance, asynchronous tasks in Django (via Celery) and database indexing significantly enhance the efficiency of large-scale file analysis and real-time interactions. For security, data encryption (e.g., Bcrypt) and HTTPS communication protect data in storage and transmission. Scalability is achieved through modular design, enabling easy integration of new features, such as support for other programming languages or multi-user collaboration. Maintainability benefits from clearly defined module boundaries and unit test coverage (pytest), reducing maintenance overhead. Usability is enhanced by WebSocket and interactive charts, providing users with real-time, intuitive feedback.

With this layered architecture and carefully chosen technology stack, the system achieves a balance between performance, scalability, security, and user experience, meeting complex business requirements and supporting long-term stable operation.