IT Ticket Management & Python Code Analysis Web System

A Unified Platform for Code Quality Assurance and Task Management

Team Members: Jiaxuan Yue, Jiahua Zhu, Wenxin Luo, Yanxi Jiang, Baizheng Chen, Linsen Song, Leying Deng

Abstract

Modern software development faces disjointed workflows between code review and task management, leading to inefficiency and miscommunication. This work presents an integrated web application that unifies Python code quality analysis and role-based IT ticket management. Built with Flask and JSON storage, the system eliminates external database dependencies while supporting automated code linting, loop depth analysis, and structured task/request workflows. It delivers a lightweight, modular solution tailored for small-team collaborative development.

System Overview

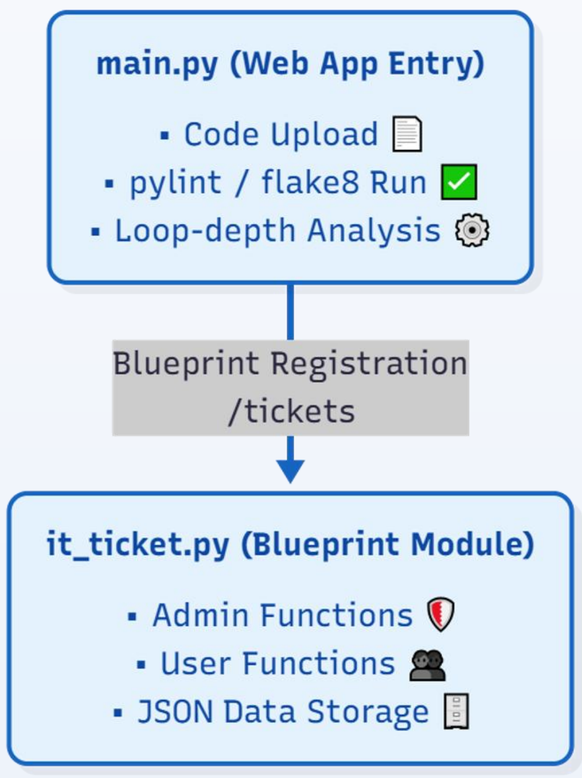
(1) Core Objectives

- Break down workflow barriers between code review and task management to achieve integration
- Automate Python code quality detection for consistent reviews
- Establish role-based permission boundaries to differentiate admin and user functions
- Adopt a database-free design to support rapid deployment and lightweight operation

(2)Technology Stack

- Backend: Flask (main framework) + Blueprint (modular design for decoupling code review and ticket systems)
- Data Storage: JSON files (users.json for user info, tasks.json for tickets, requests.json for user requests)
- Frontend: HTML/CSS responsive templates (unified gradient styles, rounded buttons, role-specific navigation)
- Code Analysis Tools: Pylint (error detection), Flake8 (PEP8 compliance check), custom loop nesting depth analyzer

System Architecture

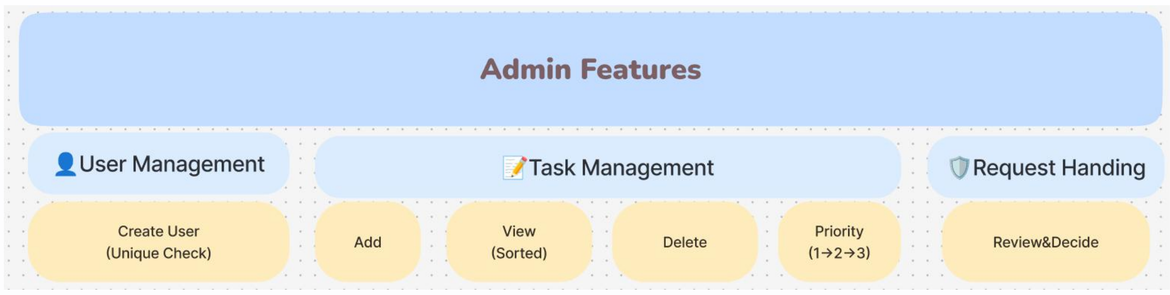


- **main.py:** the primary entry of the web application.
  Core Workflow (3 Steps)
  (1)Upload Code： Files saved to /uploads directory
  (2)Select Tools： Choose 3 analysis options as needed
  (3)Execute & Display： Run tools via backend, show results in HTML
- **/tickets.:** Bridge between main module & ticket module, enabling modular design
- **it_ticket.py:** Independent blueprint for task management

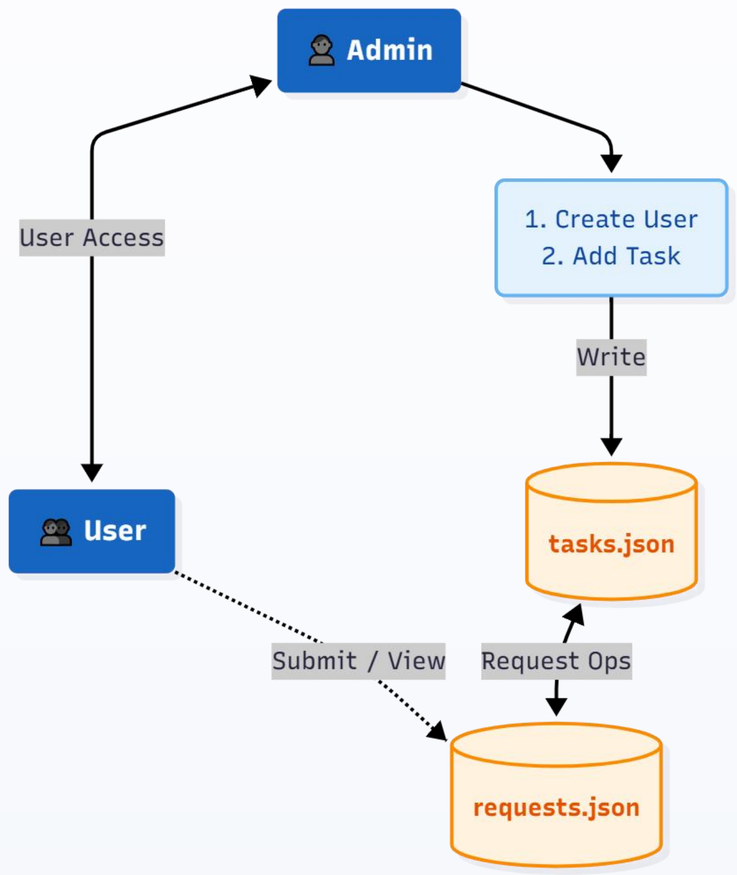
Both modules share the same Flask server, balancing modularity and application unity

Role Function Comparison

Admin Feature Modules	User Feature Modules
(1)User Management • Create User (saved to users.json, unique username check)	**(1)Task View** • View assigned tasks (sorted by priority) • Displayed on personal dashboard: /tickets/user/<username>
(2)Task Management • Add Task (Title, Assigned User, Priority 1-3) • View Tasks (table presentation, priority-sorted) • Delete Task (by task ID) • Change Priority (cycle: 1→2→3→1)	**(2)Request Submission** • Submit task deletion request • Submit priority change request (Requests generate JSON records with "pending" status, awaiting admin action)
(3)Request Handling • Review & handle user-submitted task modification requests (approve/dismiss)	**(3)Request Example (JSON)** json { "id":1, "username":"Alice", "task_id":5, "request_type":"delete", "status":"pending" }



System Workflow



- **Admin Actions:** Create User → Add Task → Write to tasks.json
- **User Actions:** View Tasks → Submit Request → Write to requests.json
- **Core Logic:** Admin controls task lifecycle; users only propose modifications; JSON files ensure data persistence

Conclusion and Future Work

Core Achievements

Integrated code review and task management to eliminate workflow silos

Modular design supports independent evolution of the two core functions, balancing scalability and user experience

Lightweight, database-free architecture lowers deployment barriers for small teams

Future Optimization Directions

Short-term: Integrate Flask-Login for user password authentication; migrate to SQLite to resolve JSON concurrency issues

Medium-term: Extend code analysis support to JavaScript/Java; add a custom validation rule configuration interface

Long-term: Add code complexity visualization , ticket status tracking, and operation audit logs