Automated Code Review Agent

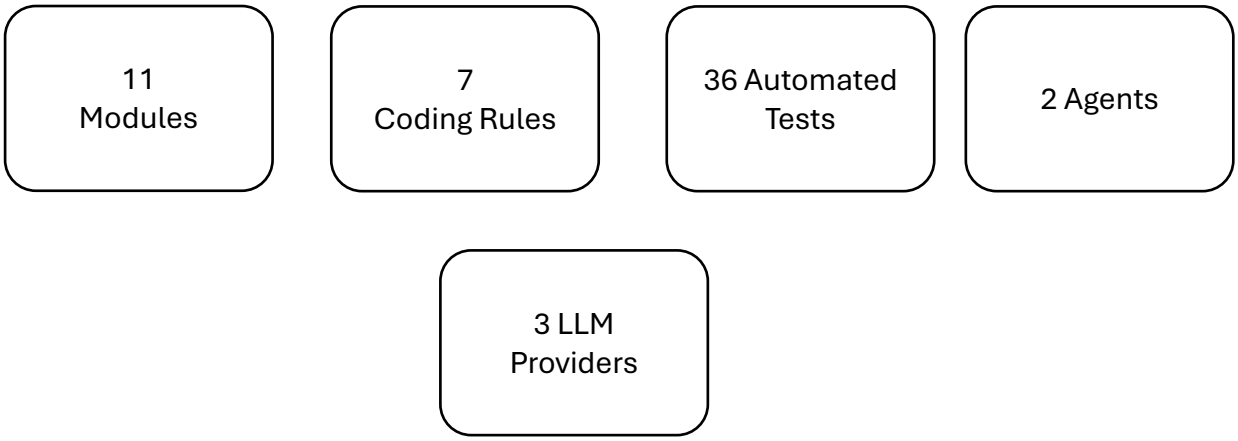
An intelligent, Multi-Agent Code Review System with GitHub PR Integration

Software Engineer Take Home-Home Assignment

Wilson Dagah

Project Overview

An end-to-end agentic code review pipeline that autonomously analyses pull requests, provides actionable inline feedback and delegates to a refactoring agent for automated fixes.



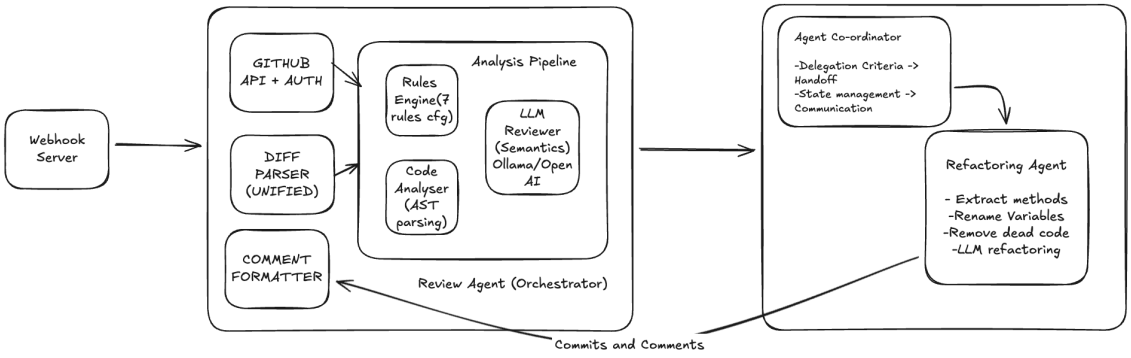
Technology Stack

1. Python
2. FastAPI
3. PyGithub
4. Ollama/OpenAI/Anthropic
5. AST Parsing
6. YAML Config
7. Multi Agent
8. Webhook

Submission Deliverables Checklist Done

- Source Code (Well Structured Python, clear separation of concerns)
- README.md (Setup instructions, architecture overview, usage examples)
- Configuration (YAML coding standards, customisable rules)
- Examples directory (real PR review outputs, JSONL logs)
- Presentation (this pdf)
- Logs/Output (JSON + JSON, machine readable, auditable)

System Architecture



System Architecture - How it works

The system operates as a three-tier pipeline triggered by GitHub pull request events:

1. **Webhook Server (Entry Point)** A FastAPI server listens for incoming GitHub webhook events. When a pull request is opened, updated, or reopened, the server verifies the request signature using HMAC SHA-256, extracts the PR metadata from the payload, and dispatches the review pipeline as a background task. The server also detects commits authored by the refactoring agent itself and skips those to prevent infinite processing loops.

2. **Review Agent (Orchestrator)** The Review Agent is the central orchestrator that coordinates the entire analysis. It uses the GitHub Client to authenticate via a personal access token, fetch the PR diff, and retrieve full file contents at the head commit. The Diff Parser breaks each file's unified diff into hunks and individual changed lines, building a line-to-diff-position map that GitHub's API requires for placing inline comments on the correct line.

The parsed files then flow through three analysis layers in parallel:

- Rules Engine - Applies 7 YAML configurable checks (line length, naming conventions, complexity, unused imports, hardcoded secrets, dangerous functions, docstring coverage), scoped only to changed lines.
- Code Analyzer - Uses Python's ast module to detect structural issues like deep nesting, mutable default arguments, bare except clauses, excessive parameters, and ***print()*** statements.
- **LLM Reviewer** - Sends the code to a language model (Ollama, OpenAI, or Anthropic) for semantic analysis, catching logic errors, design smells, and security patterns that static checks miss.

All findings are collected with severity levels, confidence scores, and file/line references, then formatted into GitHub-compatible markdown by the Comment Formatter and posted as inline review comments directly on the PR diff.

3. **Agent Coordinator & Refactoring Agent (Multi-Agent Layer)** After the analysis phase, the Agent Coordinator evaluates whether any file meets the delegation criteria, more than 3 violations, complexity score above 10, or any critical-severity finding. If so, it initiates a structured handoff to the Refactoring Agent via a delegation request message.

The Refactoring Agent receives the file's source code and list of issues, then applies automated fixes: renaming variables to snake_case, removing unused imports, or using the LLM for complex refactoring like method extraction. Every refactored output is validated with ***ast.parse()*** before being committed back to the PR branch. The coordinator records the full handoff lifecycle, and the Review Agent cross-references the applied fixes with the original findings, marking each issue as resolved or unresolved in the final output.

Baseline: Code Review Agent

GITHUB INTEGRATION

- FastAPI webhook server
- HMAC SHA-256 signature verification
- PAT authentication with rate limiting
- Inline diff-positioned review comments

DIFF PARSING

- Unified diff parser
- Hunk extraction with line mapping
- Line-to-diff-position mapping
- Added/deleted/context detection

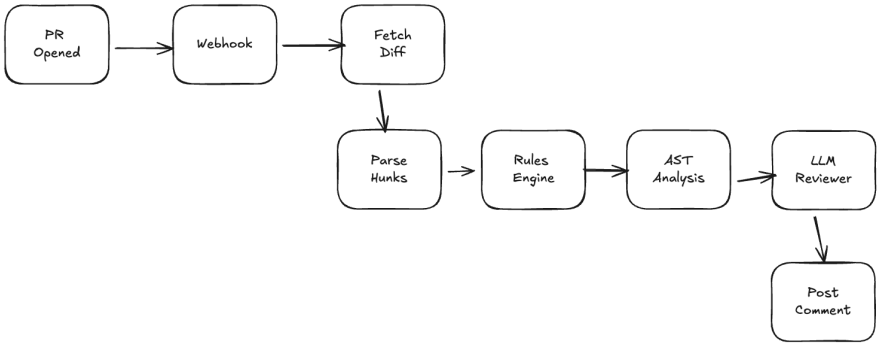
CODING STANDARD RULES

- Line Length, naming conventions
- Function complexity, unused imports
- Hardcoded secrets, dangerous functions
- Docstring coverage

ANALYSIS PIPLINE

- AST parsing with pattern matching
- LLM semantic review (3 providers)
- Severity levels and confidence scores
- Actionable fix suggestions

Analysis Pipeline Flow



Static Analysis Checks

- Mutable default arguments
- Deep nesting (configurable threshold)
- Bare excepts clause
- Excessive function parameters
- Print() instead of logging
- TODO/FIXME/HACK comments

Rules Engine Checks

- PascalCase classes, snake_case functions
- Line length limit (configurable)
- Hardcoded secrets (passwords, tokens, keys)
- Dangerous eval(), exec(), pickle, subprocess
- Function complexity scoring
- Missing docstrings

Multi Agent Delegation System

Agent Co-ordinator Checklist

- Delegation criteria (configurable threshold)
- Structured message protocol (Request -> Results)
- Full trail audit with handoff records
- State management across handoffs

Delegation Triggers

Criterion	**Threshold**
Violation per file	>3
Complexity Score	>10
Critical Severity	Any

Refactoring Agent Checklist

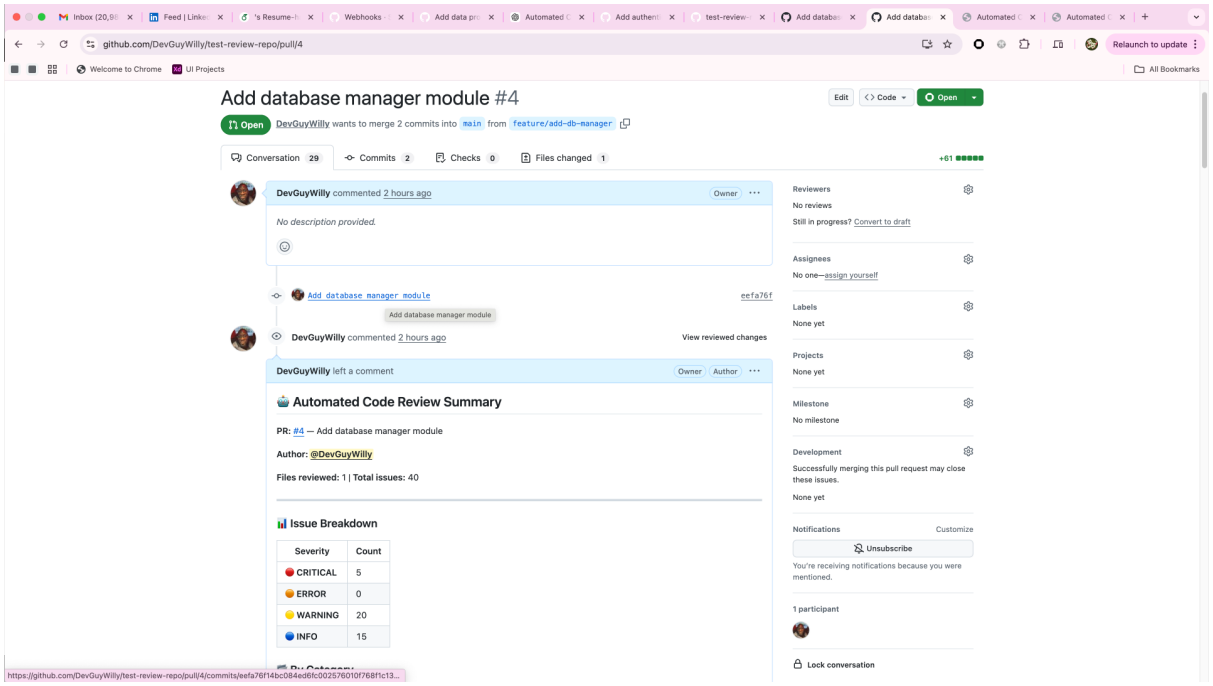
- Rename variables to snake_case
- Remove unused imports
- LLM powered refactoring
- Syntax validation via ast.parse()
- Content size validation before commit
- Commits changes back to PR branch

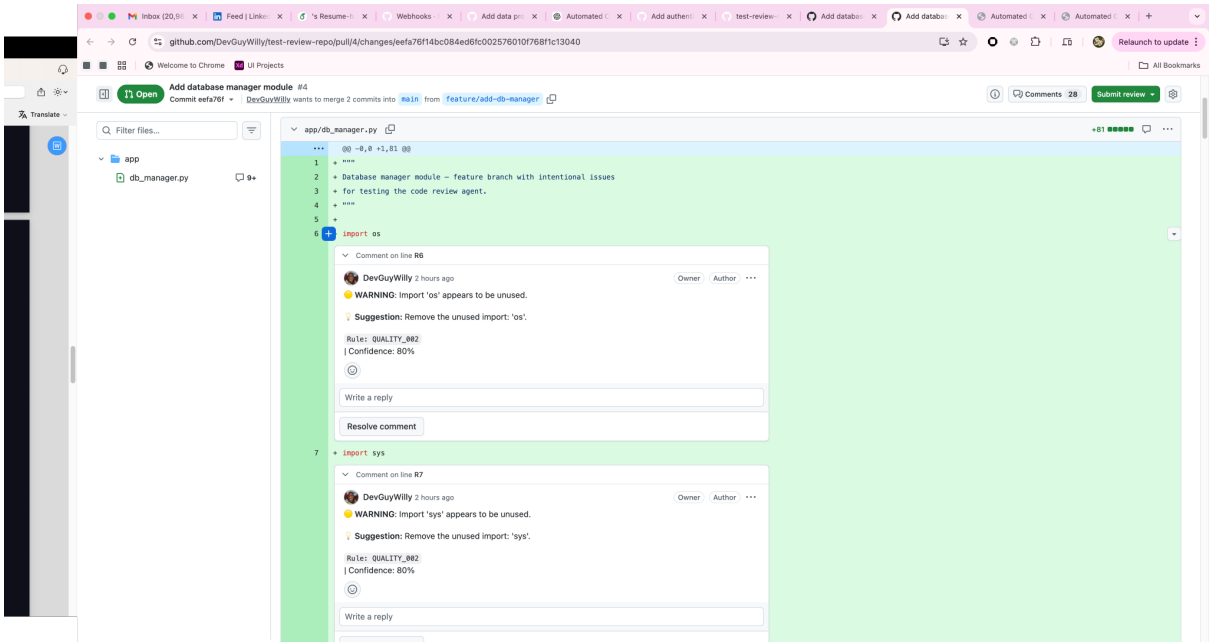
Safety Mechanisms Checklist

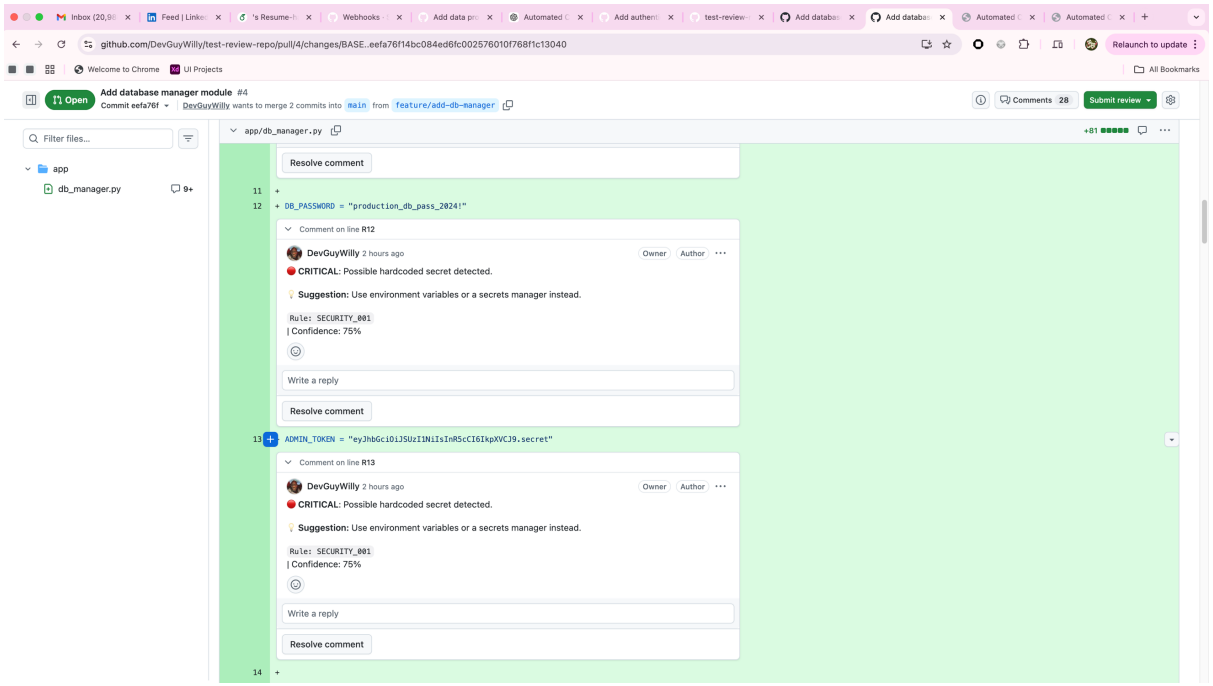
- Infinite loop prevention (agent commit detection)
- Content corruption safeguards
- Batch commits per file (Avoids SHA conflicts)

RESULTS: LIVE TEST REVIEW

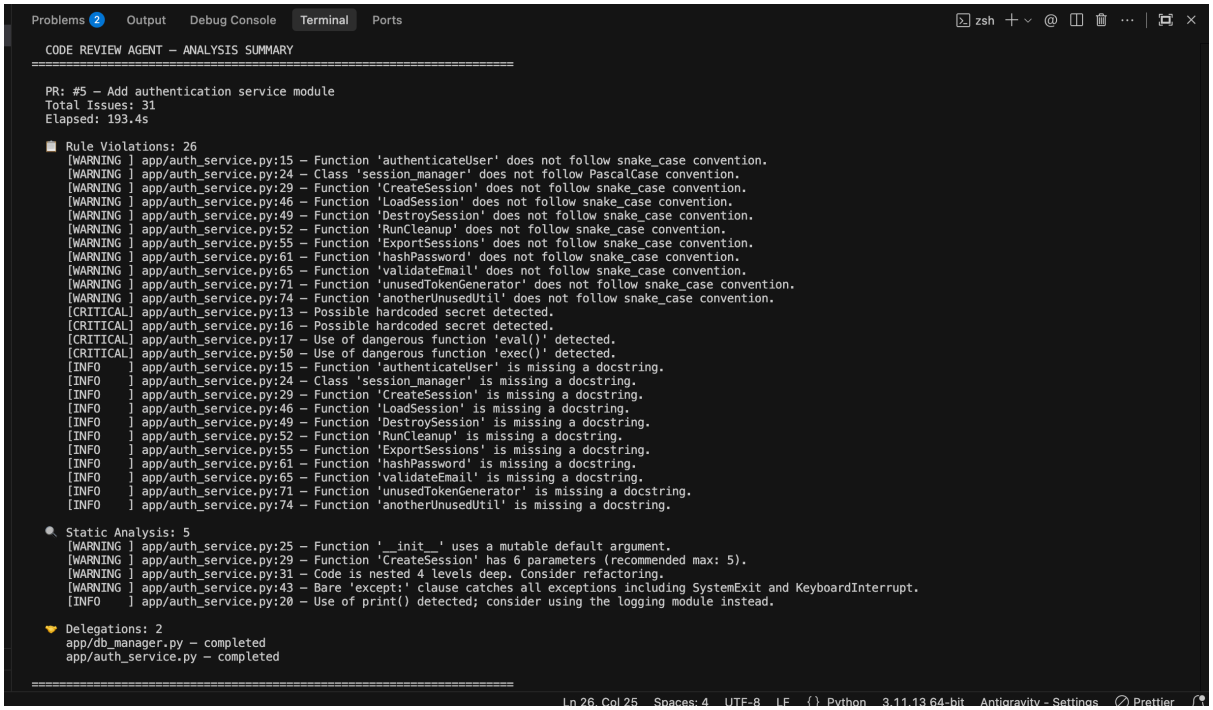
I’ve tested with real PRs containing intentional flawed Python code (hardcoded secrets, SQL injection, eval(), nesting, bad naming etc)



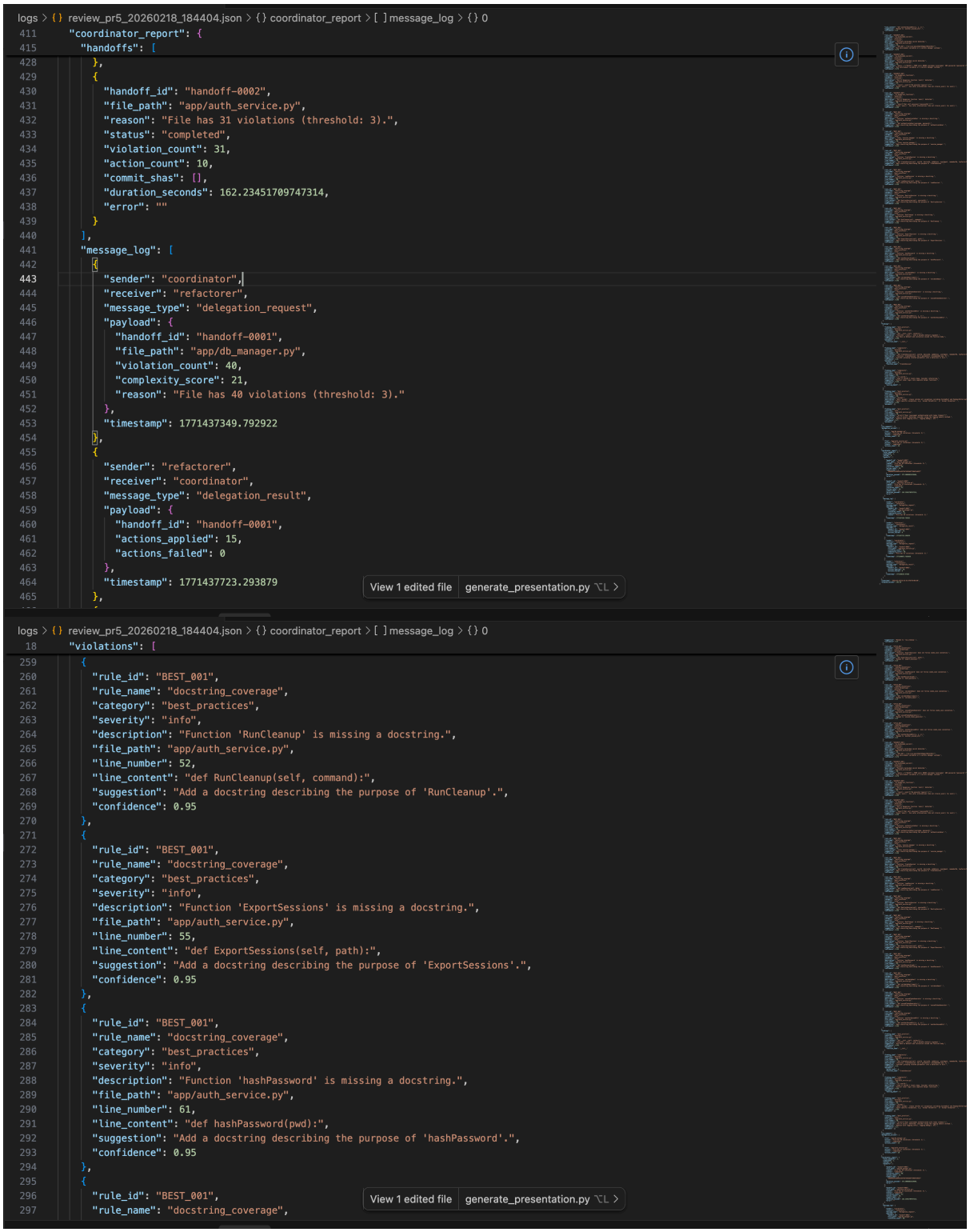




Console Output with Resolution



Logs Output (JSON)



Architectural Decisions

Decision	Choice	Rationale
Language	Python	Rich ecosystem, stdlib ast module, PyGithub library
LLM Integration	Multi provider	Ollama(free/local), OpenAI and Anthropic (switchable via env)
Rule System	YAML Config	Easy configuration without code changes
Static Analysis	AST and Regex	Zero external deps (tree-sitter as future option)
Multi-Agent	Custom Protocol	Light weight, deterministic delegation, clear handoff semantics
Webhook	FastAPI	Async support, auto OpenAPI docs, Pydantic validation
Comment Placement	Diff Position	Comments on exact diff line, not fallback issue comments

Known Limitations and Next Steps

Current Limitation

- LLM review requires Ollama or API Key
- Complex refactoring depends on LLM quality
- Single file refactoring scope
- Inline comments only for lines within the diff

Planned Enhancement

- Verification agent – run test after refactoring
- Rollback capability on test failure
- Multi Agent language support via Tree sitter
- Test coverage analysis and suggestion
- Caching Layer for LLM responses
- GitHub Actions, reusable action package
- Dashboard for analysis history and trends

Link to GitHub Repository - **https://github.com/DevGuyWilly/pr-review-agent.git**

Link to mini project Code Review Agent was tested on - https://github.com/DevGuyWilly/test-review-repo.git