Automatic LLM Refactor

Prepared By

Abhishek Kumar

High-Level Design (HLD)

Goal:

Automatically detect design/code smells in Java code, refactor the code using LLMs, and raise a Pull Request on GitHub.

Components Overview:

1. Trigger Mechanism - GitHub Action
2. CI/CD - GitHub Actions
3. Core Python Script for file handling and LLM calls
4. LLM Engine - Gemini (Google Generative AI)
5. GitHub API integration for committing changes and raising PRs

Architecture Flow:

1. GitHub Action is triggered manually.
2. Python script runs to select random Java files.
3. Code is sent to LLM (Gemini) to detect design smells and generate refactored code.
4. New branch is created.
5. Refactored code is committed and pushed.
6. A Pull Request is automatically raised with the summary.

Technologies:

- Python, GitHub Actions, PyGithub, Google Generative AI, GitHub API, OpenAI (optional)

Low-Level Design (LLD)

Module-wise Breakdown:

1. get_repo(): Connects to the GitHub repo using PyGithub and a secret token.
2. pick_files(): Lists all Java files and picks 1-2 randomly from the same directory.
3. refactor_file(): Sends the code to Gemini LLM to:
   - Detect design/code smells
   - Refactor the code
4. apply_refactorings_to_files(): Creates a new branch and commits refactored code.
5. create_pull_request(): Raises a PR from the new branch with design smell summary.

Data Flow:

GitHub Action -> Checkout Code -> Run Python Script -> Connect to GitHub -> Pick Files -> Analyze with LLM -> Generate Refactor -> Commit to Branch -> Raise PR

Design Considerations:

- Secure Secrets via GitHub Secrets
- Try/Except for error handling
- Random selection of files
- Easily extendable to other file types or languages

Future Enhancements:

- Support for test case generation
- Support for multiple languages (Python, JS)

- CI gate integration for quality threshold
- Approvals before auto-committing
- VS Code/GitHub Copilot integration

Conclusion

This design is ideal for LLM-driven developer tooling. The HLD outlines how systems interact, and the LLD provides implementation-level clarity.

For a scripting tool powered by LLMs and GitHub APIs, this structure is suitable and follows standard design practices.