**PROJECT REPORT**

**TITLE:** **SMART SDLC – AI-POWERED SOFTWARE-DEVELOPMENT-LIFE-CYCLE ASSISTANT USING IBM GRANITE**

Team ID : LTVIP2025TMID32004

Team Size: 4

Team Leader:  Kuraganti Bharath

Team Member: Kasturi Sandeep

Team Member: Kantamaneni Prudhvi

Team Member: Kode Devi Sri Vallika

**CONTENTS:**

1.Introduction

1.1 Project Overview

1.2 Purpose

2.Ideation Phase

2.1 Problem Statement

2.2 Empathy Map Canvas

2.3 Brain Storming

3.Requirement Analysis

3.1 Customer Journey Map

3.2 Session Requirements

3.3 Data Flow diagram

3.4 Technology Stack

4.Project Design

4.1 Problem Solution Fit

4.2 Proposed Solution

4.3 Solution Architecture

5. Project Planning and Scheduling

6. Functional and Performance testing

7. Results

8.Advantages and Disadvantages

9.Conclusion

10.Appendix

**1. INTRODUCTION**

**1.1 PROJECT OVERVIEW**

SMART SDLC is an intelligent assistant that automates and augments every major SDLC phase—requirements analysis, code generation, bug fixing, testcase creation, code summarization and conversational help—using IBM Watsonx Granite 13B and a Streamlit-based user interface.

**1.2 PURPOSE**

The system accelerates software delivery while improving code quality. It enables developers to prototype faster, eliminate repetitive tasks and gain instant insight into legacy code from a single dashboard.

**2. IDEATION PHASE**

**2.1 PROBLEM STATEMENT**

Developers spend significant time reading old code, writing boilerplate and tracking down bugs. Existing tools address only fragments of this workflow. A unified large-language-model copilot can remove those bottlenecks.

**2.2 EMPATHY MAP CANVAS**

SAYS “How do I fix this bug quickly?” “Can the AI write my unit tests?”

THINKS “Will the generated code be safe and maintainable?”

DOES Searches Stack Overflow, copies snippets, writes ad-hoc scripts

FEELS Stressed by deadlines, frustrated by repetitive tasks

PAINS Manual debugging, boilerplate coding, unclear legacy logic

GAINS Faster turnaround, fewer errors, clearer understanding

**2.3 BRAINSTORMING**

Standalone ideas such as bug fixer, test generator and story extractor were combined into one end-to-end assistant covering the complete SDLC.

**3. REQUIREMENT ANALYSIS**

**3.1 CUSTOMER JOURNEY MAP**

4. Start the Streamlit web application.

5. Select a module: Requirement, Code Generation, Bug Fix, Tests, Summary or Chat.

6. Provide input by uploading a PDF or code snippet.

7. The Granite model produces stories, code, fixes or answers.

8. Review the output, copy or download as needed.

9. Switch modules or end the session.

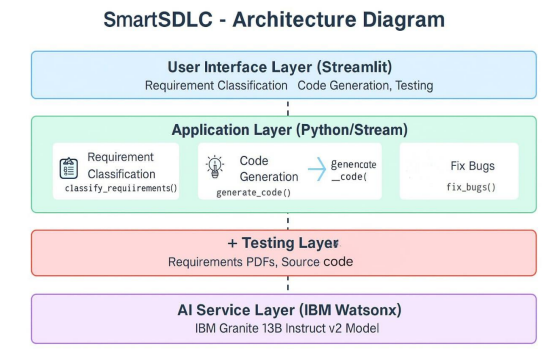
**3.2 SESSION REQUIREMENTS**

a) Upload PDF and code files.

b) Receive real-time AI responses.

c) Download generated assets.

d) Preserve chat history within the session.

**3.3 DATA FLOW DIAGRAM**

**3.4 TECHNOLOGY STACK**

Frontend Streamlit

Backend Python 3.11

AI Service IBM Watsonx Granite 13B Instruct v2

PDF Parsing PyMuPDF

Environment Management virtualenv and .env secrets

**4. PROJECT DESIGN**

**4.1 PROBLEM–SOLUTION FIT**

Teams need faster, higher-quality delivery. Embedding Granite LLMs inside daily tools provides intelligent automation that meets this need.

**4.2 PROPOSED SOLUTION**

Layer 1 User interface: individual Streamlit pages per module

Layer 2 Core logic: Python helpers for PDF handling, code cleanup and API calls

Layer 3 AI layer: cached Granite model accessed with secure credentials

**4.3 SOLUTION ARCHITECTURE**

UI Layer Sidebar navigation, chat window, file widgets

Application Logic app.py and pages route requests

Helper Layer watson.py, pdf\\_utils.py, cleaning.py

AI Layer Granite service with retry and rate-limit control

**5. PROJECT PLANNING AND SCHEDULING**

Week 1 (12 Jun – 19 Jun) Idea finalisation, Streamlit skeleton, PDF ingestion

Week 2 (20 Jun – 26 Jun) Granite API integration, module logic, unit tests

Week 3 (27 Jun – 03 Jul) Bug-fix loop, UI polish, report creation

Week 4 (04 Jul – 10 Jul) Final demonstrations, documentation, deployment script

**6. FUNCTIONAL AND PERFORMANCE TESTING**

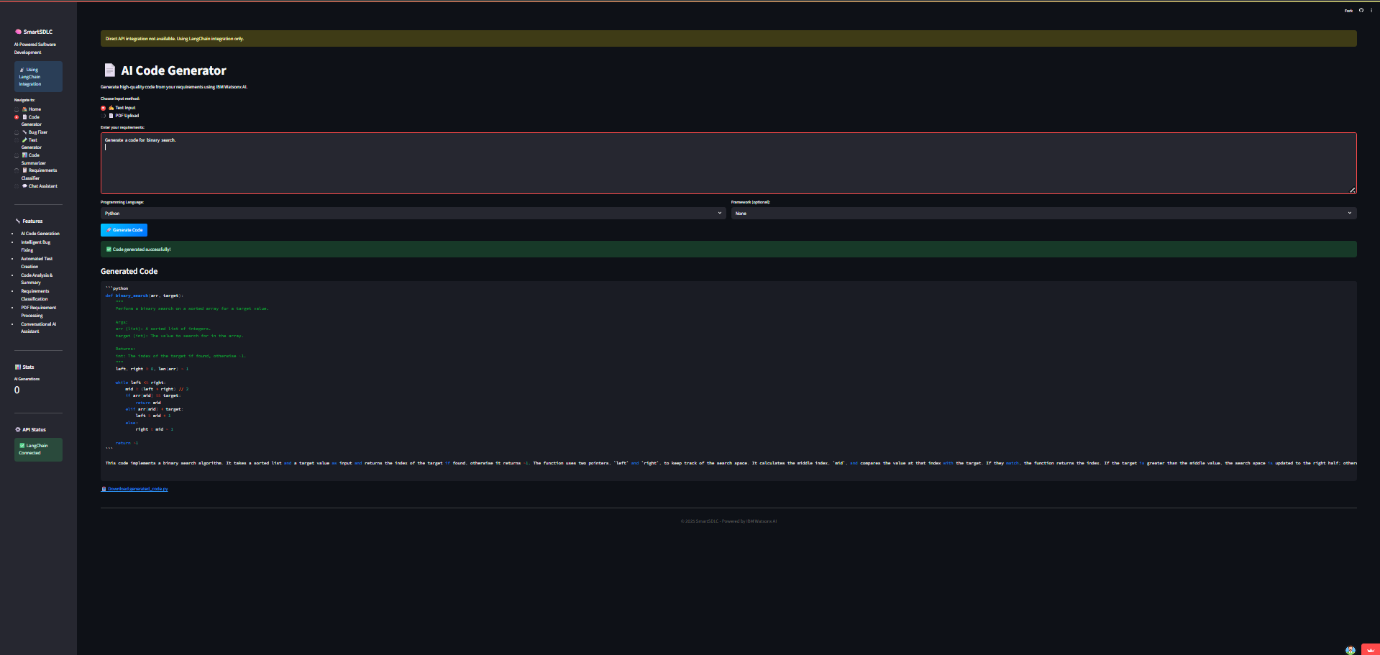
Unit testing PDF parser and code-cleanup utilities

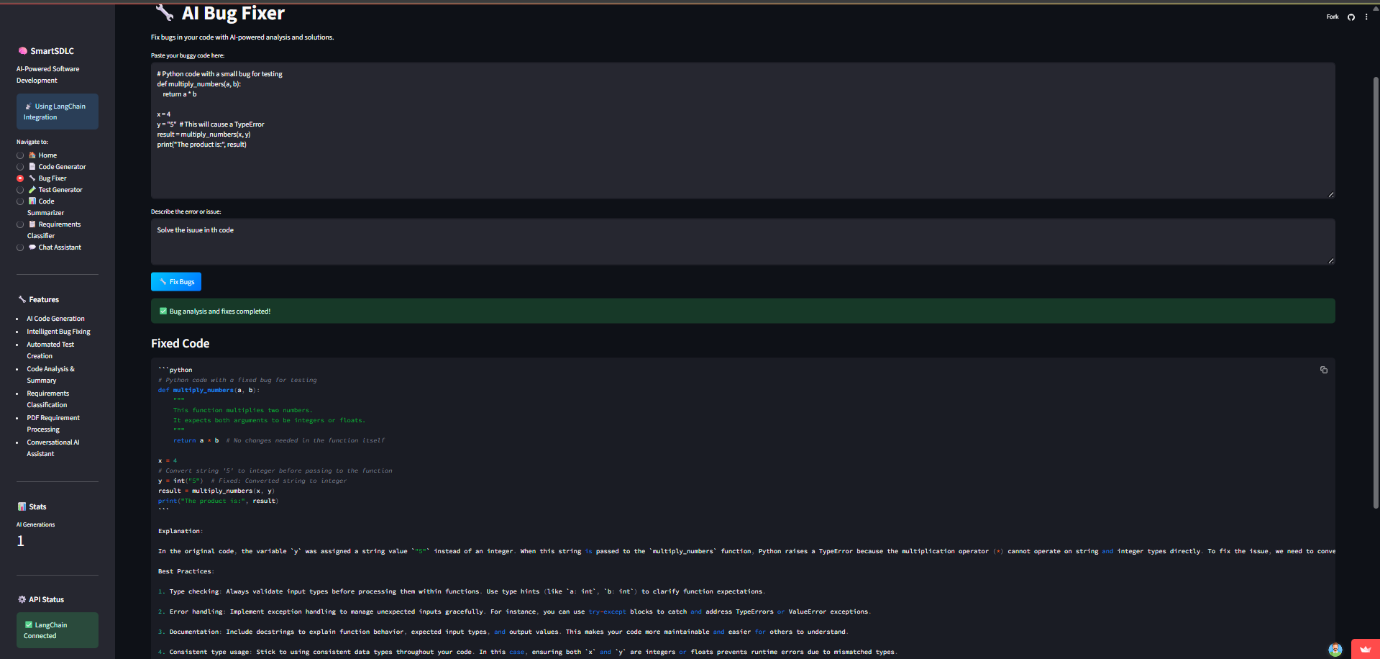
Integration testing End-to-end Streamlit to Granite response

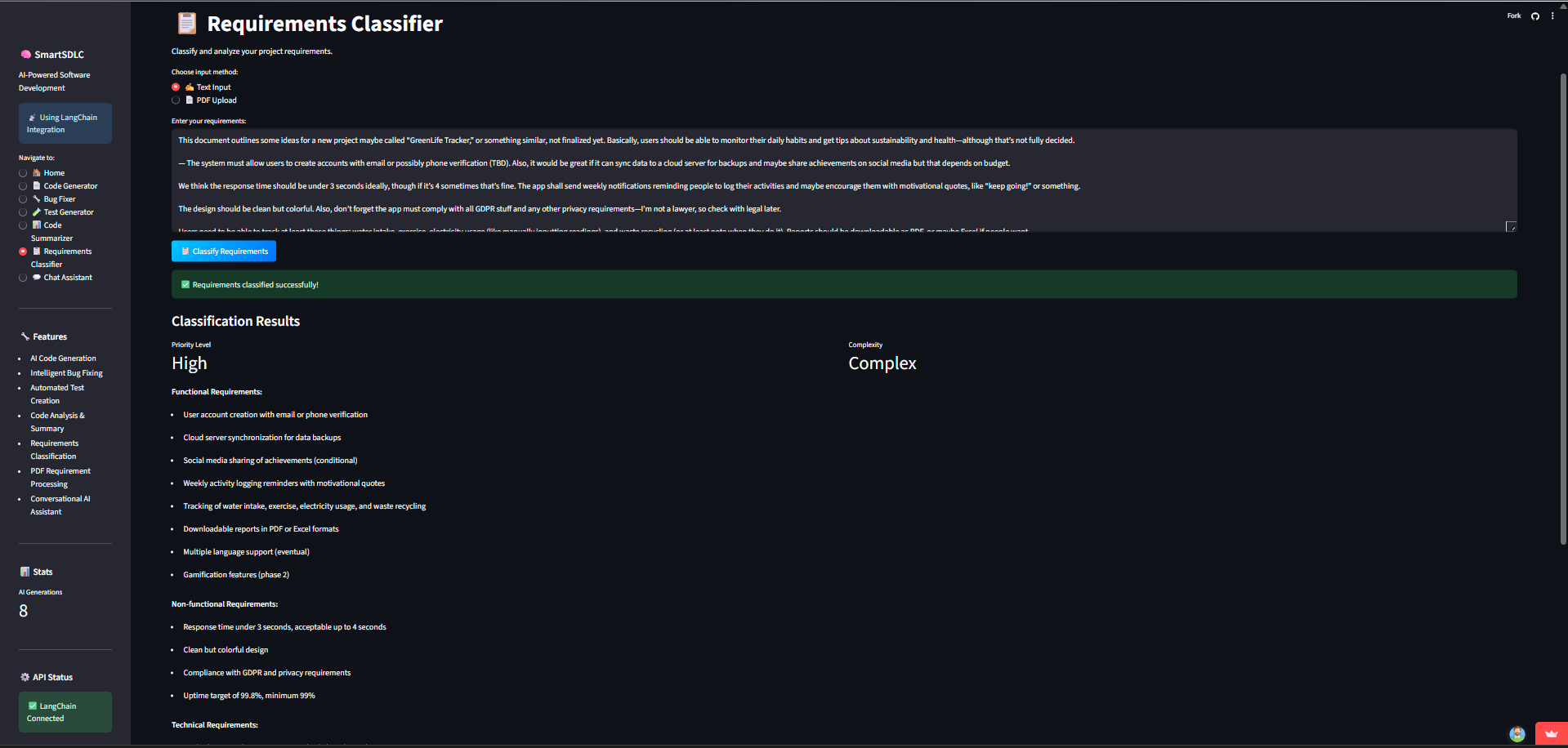
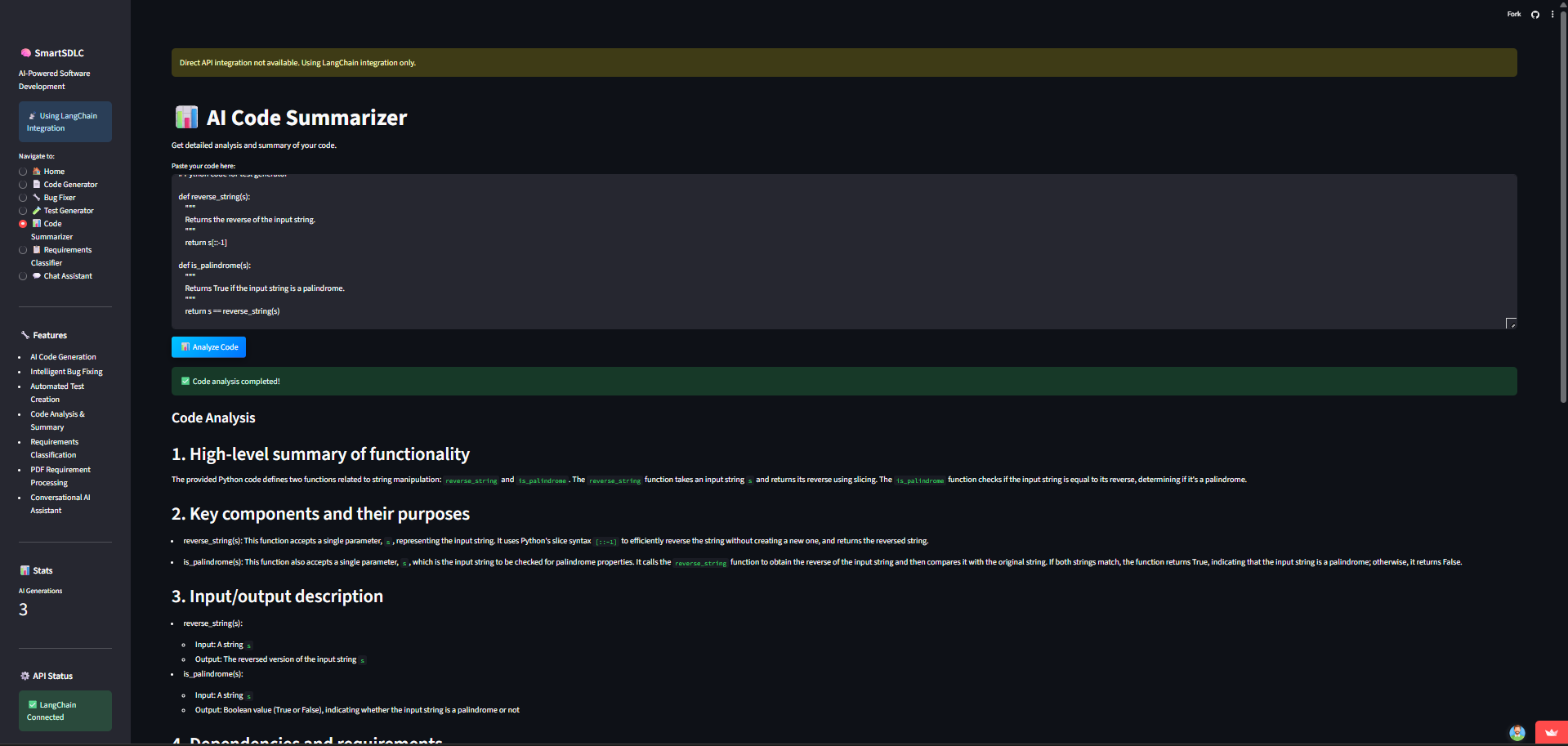
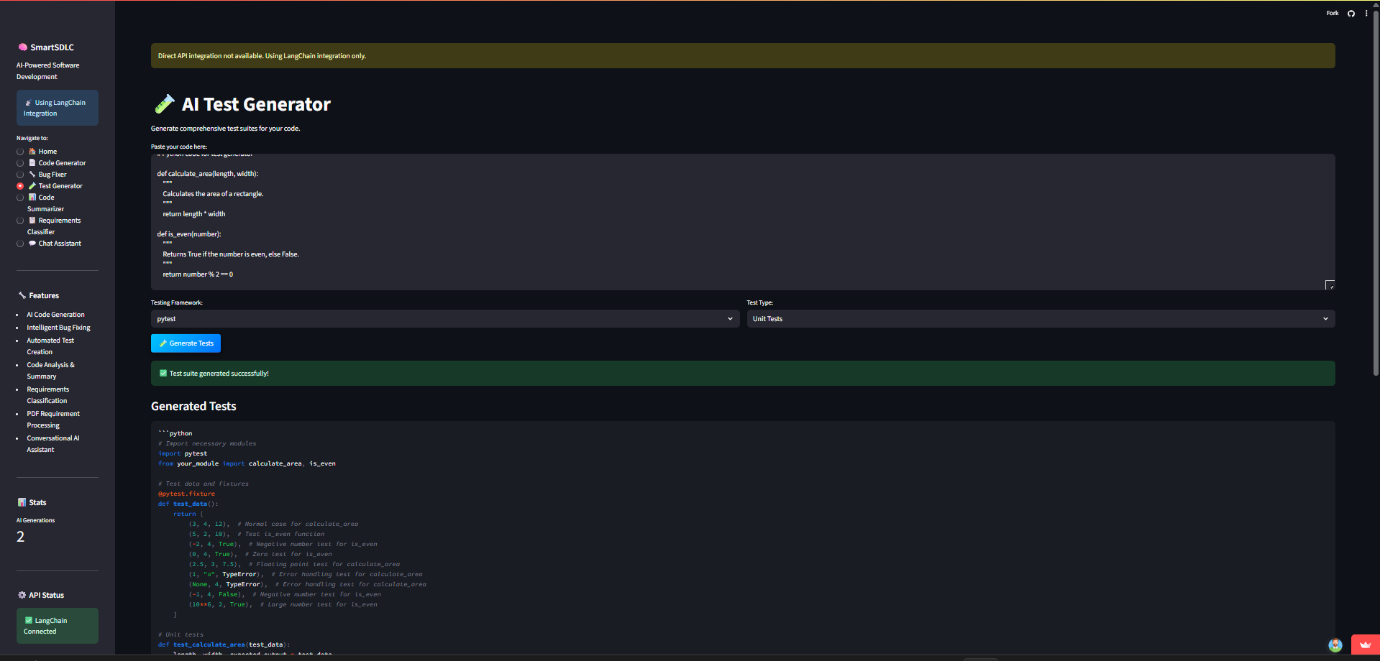
Manual testing Real project PDFs and GitHub codebases

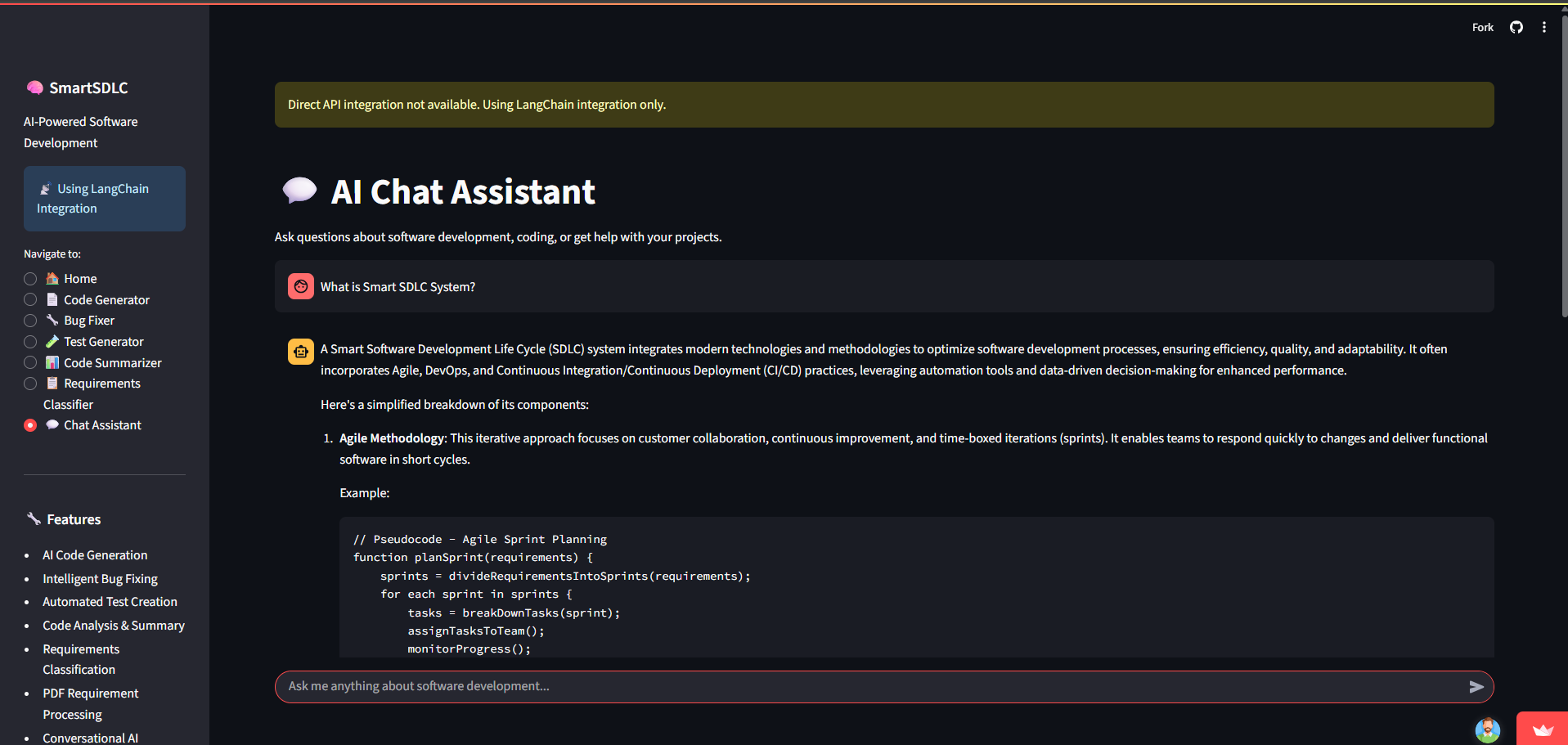
Error handling Network drops, oversized files, API-quota exhaustion

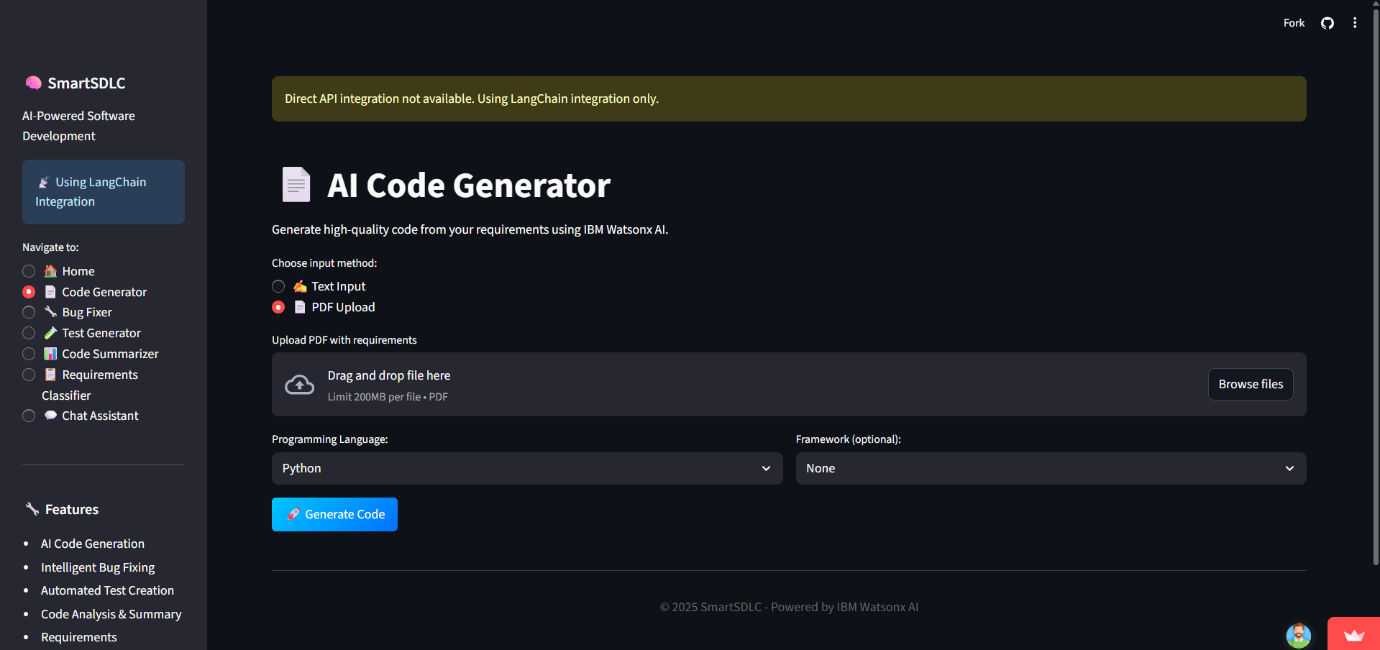
**7. RESULTS**

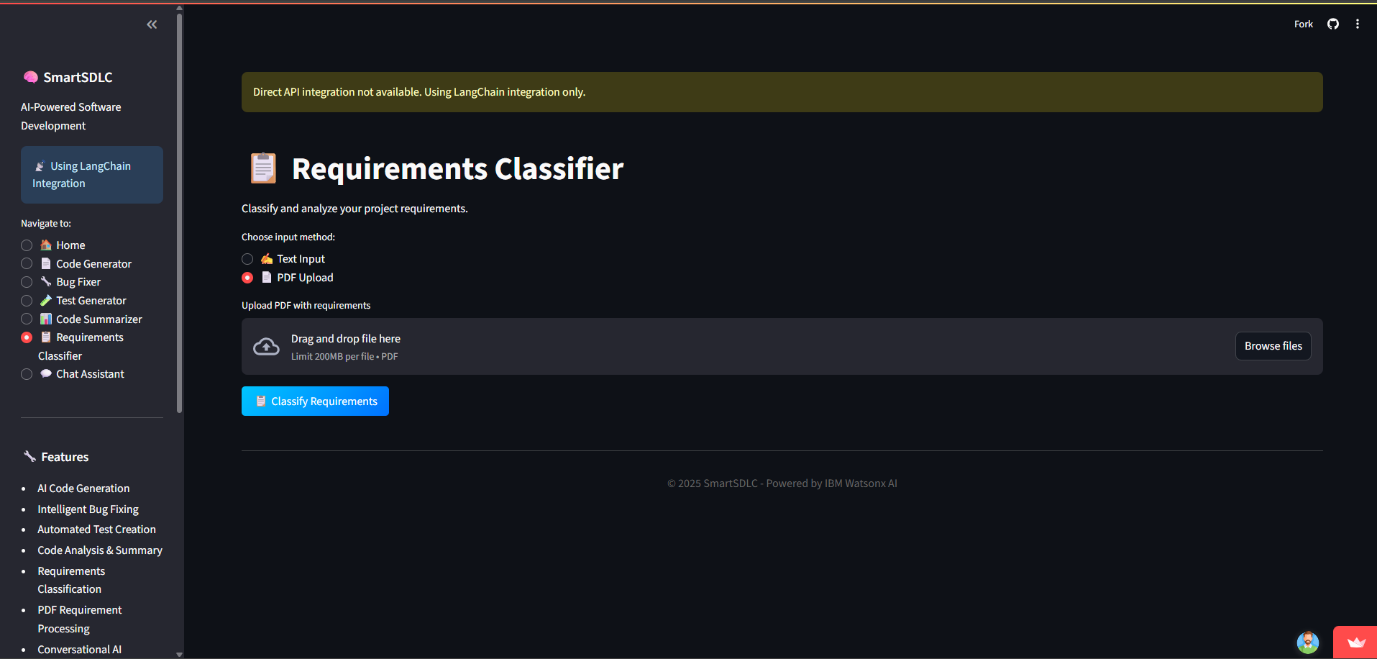
****

****

****

****

****

****

**8. ADVANTAGES AND DISADVANTAGES**

ADVANTAGES

• Complete SDLC coverage in one tool

• Faster prototype-to-production pipeline

• High-quality code and summaries from Granite 13B

• Open and extensible foundation

DISADVANTAGES

• No user authentication yet

• Limited language support beyond Python (road-map)

• Internet connectivity required for AI service

**9. CONCLUSION**

SMART SDLC demonstrates that generative AI can streamline software engineering. By pairing Streamlit’s simplicity with Watsonx power, it reduces development time and improves reliability, forming a basis for enterprise adoption.

**10. APPENDIX**

GitHub Repository https://github.com/Sandeepkasturi/SmartSDLC.git

Key Files SMART\\_SDLC.py, pages directory

Watsonx Model ibm/granite-13b-instruct-v2

License MIT