PROJECT REPORT

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

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, test-case 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

Start the Streamlit web application.

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

Provide input by uploading a PDF or code snippet.

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

Review the output, copy or download as needed.

Switch modules or end the session.

3.2 SESSION REQUIREMENTS

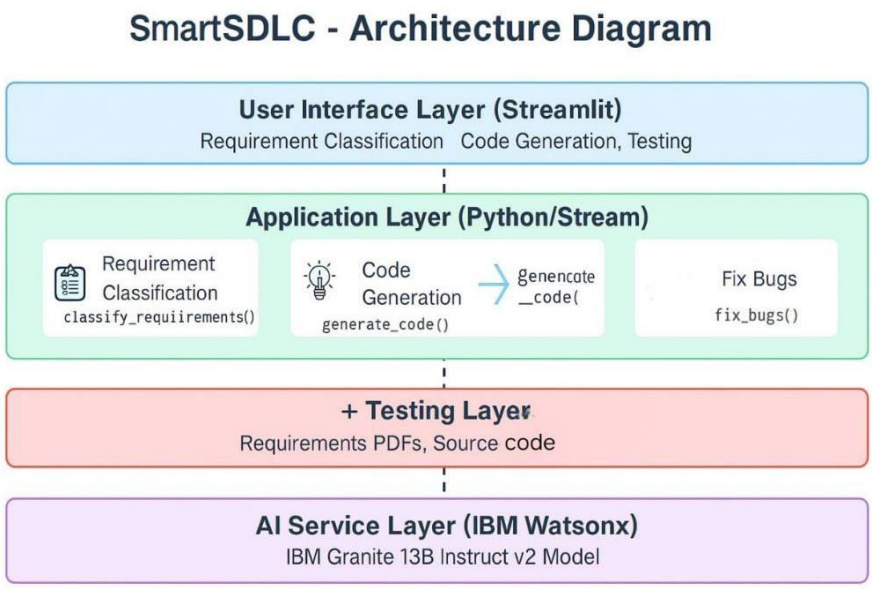
a) Upload PDF and code files.

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
- **AI Model**: IBM Watsonx Granite 3.3 Instruct
- **PDF Reader**: PyMuPDF (fitz)
- **Authentication**: .env + python-dotenv
- **Deployment Target**: IBM Cloud Foundry / Localhost

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: Streamlit interface with sidebar navigation, text inputs, file upload widgets, and chat window

Application Logic: SMART_SDLC.py and routes directory handle module selection, user input processing, and routing

Helper Layer: Includes watsonx_client.py for model calls, pdf_utils.py for text extraction, and cleaning.py for prompt cleanup

AI Layer: IBM Watsonx Granite 3.3 Instruct model with retry handling and rate-limit safeguards

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

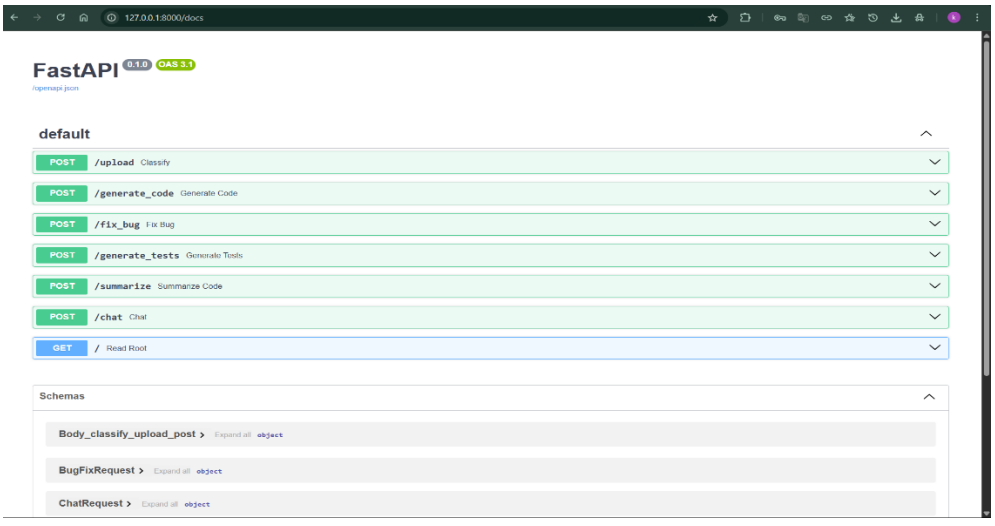
Week 3 (27 Jun – 03 Jul) Bug-fix loop, UI polish, report creation

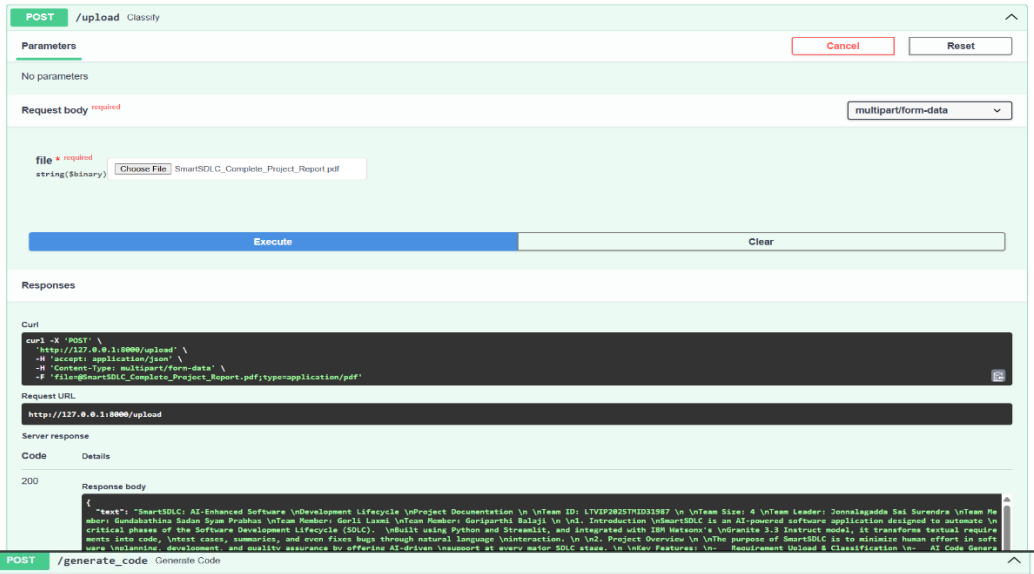
Week 4 (04 Jul – 10 Jul) Final demonstrations, documentation, deployment script

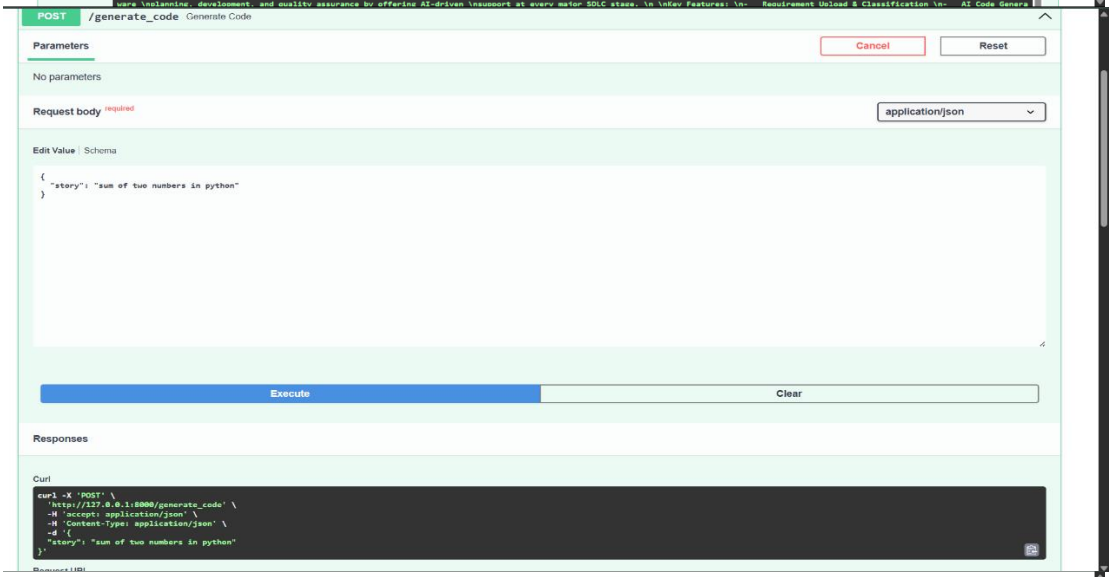
6. FUNCTIONAL AND PERFORMANCE TESTING

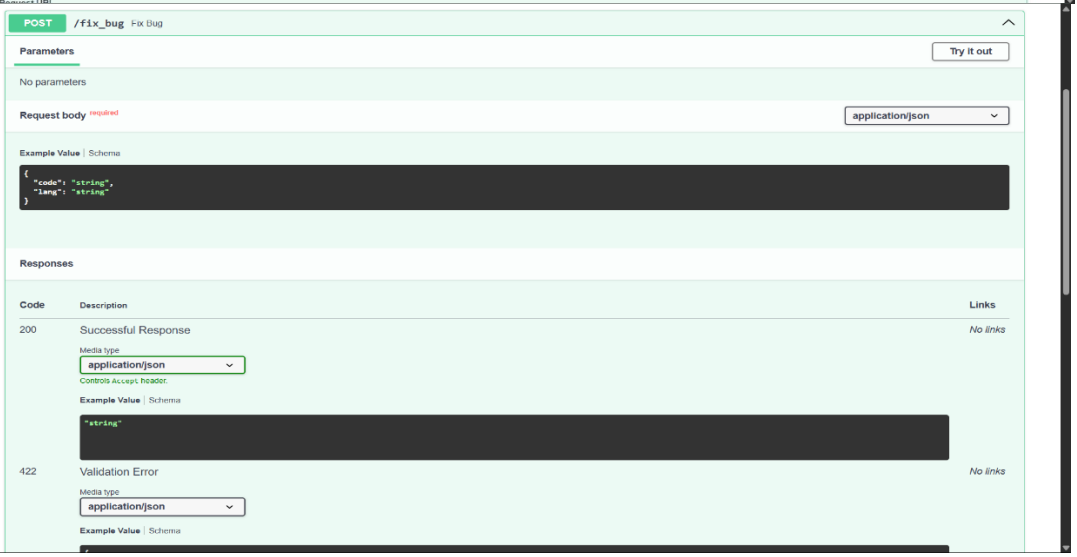
Unit Testing: Validates utility functions like pdf_utils.py and cleaning.py for parsing and prompt cleaning
Integration Testing: Full pipeline testing from Streamlit UI to Watsonx Granite output
Manual Testing: Tested using real-world PDF requirement documents and raw code samples from GitHub repositories
Error Handling: Handles common issues like network failures, large file uploads, and IBM API quota limitations

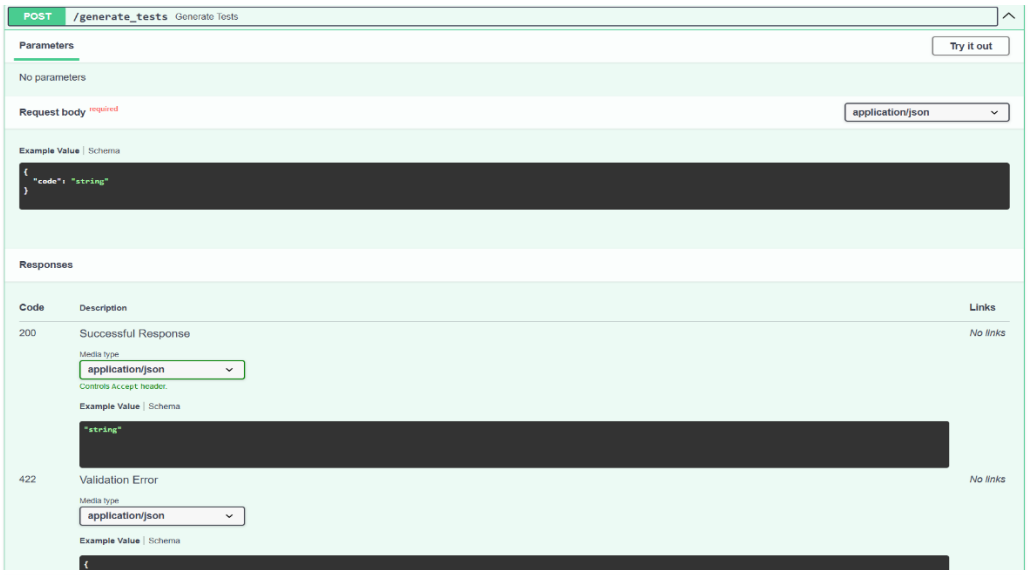
7. RESULTS



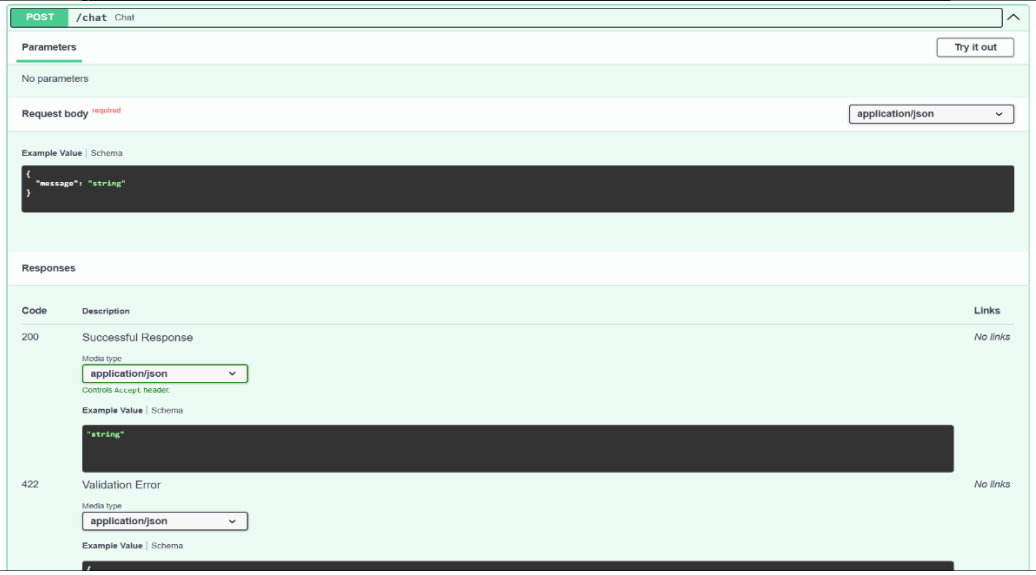

POST /upload Classify
Parameters
Cancel
Reset
No parameters
Request body required
multipart/form-data
file * required
string($binary)
Choose File SmartSDLC_Complete_Project_Report.pdf
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://127.0.0.1:8000/upload' \
-H 'accept: application/json' \
-H 'Content-Type: multipart/form-data' \
-F 'file=@SmartSDLC_Complete_Project_Report.pdf;type=application/pdf'
Request URL
http://127.0.0.1:8000/upload
Server response
Code
Details
200
Response body


POST /generate_code Generate Code
Parameters
Cancel
Reset
No parameters
Request body required
application/json
Edit Value | Schema
{
"story": "sum of two numbers in python"
}
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://127.0.0.1:8000/generate_code' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
"story": "sum of two numbers in python"
}'


POST /fix_bug Fix Bug
Parameters
Try it out
No parameters
Request body required
application/json
Example Value | Schema
{
"code": "string",
"lang": "string"
}
Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
"string"
422
Validation Error
No links
Media type
application/json
Example Value | Schema


POST /generate_tests Generate Tests
Parameters
Try it out
No parameters
Request body required
application/json
Example Value | Schema
{
"code": "string"
}
Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
"string"
422
Validation Error
No links
Media type
application/json
Example Value | Schema


POST /summarize Summarize Code
Parameters
Try it out
No parameters
Request body required
application/json
Example Value | Schema
{
"code": "string"
}
Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
"string"
422
Validation Error
No links
Media type
application/json
Example Value | Schema


POST /chat Chat
Parameters
Try it out
No parameters
Request body required
application/json
Example Value | Schema
{
"message": "string"
}
Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
"string"
422
Validation Error
No links
Media type
application/json
Example Value | Schema

8. ADVANTAGES AND DISADVANTAGES

Advantages

- Unified tool covering major stages of the Software Development Lifecycle
- Accelerates idea-to-deployment process for developers
- Generates quality code, test cases, and summaries using Granite 3.3
- Modular and easily customizable architecture for new features

Disadvantages

- No built-in user authentication system
- Currently supports only Python (multi-language planned)
- Requires constant internet access for Watsonx integration

9. CONCLUSION

SmartSDLC showcases the power of integrating generative AI into software engineering. Leveraging Streamlit for UI and IBM Watsonx for AI reasoning, it effectively automates core development tasks—planning, coding, testing, and summarization—helping teams reduce cycle time and maintain code consistency. It stands as a strong candidate for scaling to enterprise-level solutions.

11. APPENDIX

GitHub Repository	https://github.com/keerthisri1735/SmartSDLC
Key Files	main.py, app1/ directory
Watsonx Model	ibm/granite-3-3-8b-instruct
License	MIT