

SmartSDLC: AI-Enhanced Software Development Lifecycle

Project Documentation

Team ID: LTVIP2025TMID32090

Team Size: 4

Team Leader: Gogineni Venu Keerthi Sri

Team Member: Dokku Ravi Sankar

Team Member: Devarakonda Jogeswara Rao

Team Member: Eede Veda Srivalli

1. Introduction

SmartSDLC is an AI-powered software application designed to automate critical phases of the Software Development Lifecycle (SDLC).

Built using Python and Streamlit, and integrated with IBM Watsonx's Granite 3.3 Instruct model, it transforms textual requirements into code, test cases, summaries, and even fixes bugs through natural language interaction.

2. Project Overview

The purpose of SmartSDLC is to minimize human effort in software planning, development, and quality assurance by offering AI-driven support at every major SDLC stage.

Key Features:

- | Requirement Upload & Classification
- AI Code Generator
- 🐛 Bug Fixer
- 📝 Test Case Generator
- Code Summarizer
- 🗨 Chat Assistant

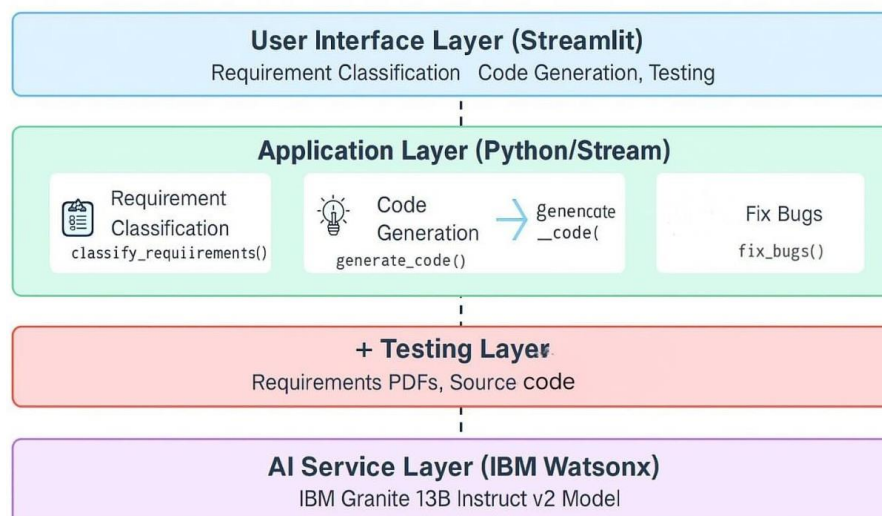
Each feature integrates IBM's Granite model to interpret user input and generate relevant code or insights.

3. Architecture

SmartSDLC follows a clean, modular architecture:

1. User Interface (Streamlit): Collects input, displays results, and manages interactions through a web-based UI.
2. Application Logic (Python): Handles user commands, generates prompts, processes AI outputs, and manages flow control.
3. AI Service Layer (IBM Watsonx): Uses IBM Granite 3.3 Instruct model to generate code, fix bugs, and provide test cases based on context-aware inputs.
4. Temporary State Memory (Session State): Stores user session data and intermediate results using Streamlit's in-memory session state management.
5. File Handling Layer (PyMuPDF): Supports PDF uploads and extracts relevant text using PyMuPDF (fitz) and local storage.
6. Security (Environment Variables): Protects credentials and API keys using .env and python-dotenv.
7. Deployment (Local/Cloud): The app is deployable on IBM Cloud Foundry or can be run locally using Uvicorn + FastAPI backend and Streamlit frontend.

SmartSDLC - Architecture Diagram



4. Setup Instructions

****Prerequisites**:**

- Python 3.8 or above
- IBM Cloud account with Watsonx access
- Required libraries: streamlit, fastapi, uvicorn, ibm-watsonx-ai, python-dotenv, PyMuPDF, pandas

****Installation Steps**:**

1. Clone the project and navigate to the folder
2. Create a virtual environment: `python -m venv venv`
3. Activate the virtual environment:
Windows: `.\venv\Scripts\activate`
Linux/macOS: `source venv/bin/activate`
4. Install dependencies: `pip install -r requirements.txt`
5. Create .env file with the following:
IBM_API_KEY="your_ibm_api_key"
PROJECT_ID="your_project_id"
BASE_URL="https://eu-de.ml.cloud.ibm.com"
6. Run the backend server: `uvicorn main:app --reload`

5. API Documentation

SmartSDLC does not expose traditional REST APIs but integrates directly with IBM Watsonx's `generate_text()` method using the Python SDK.

Example Prompt Usage:

- Code Generation: "Generate a Python function that sorts a list using merge sort."
- Bug Fixing: "Fix the following Python code: [insert buggy code]"
- Code Summarization: "Summarize what this Python script does: [insert code]"
- Requirement Analysis: "Generate use case scenarios for an e-commerce website"

AI Parameters:

- `max_new_tokens = 500`
- `temperature = 0.7`
- `top_p = 1.0`
- `decoding_method = "sample"`

6. Authentication

IBM Watsonx is accessed using secure API key authentication. Credentials are managed through a .env file and loaded using the python-dotenv package.

Security Practice:

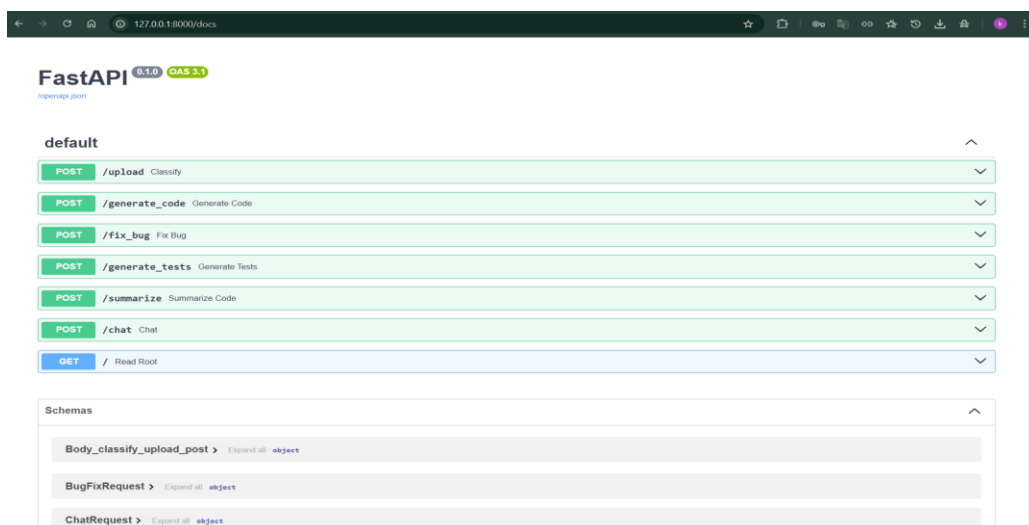
- Avoid hardcoding API keys directly in source files.
- Use a .env file to store sensitive variables securely.
- Ensure .env is listed in .gitignore to prevent leaks.
- Load environment variables dynamically at runtime using load_dotenv().

7. User Interface

The application provides an intuitive, web-based UI using Streamlit. A sidebar allows users to navigate between various SDLC modules. Each module is designed with tailored input and output widgets for seamless interaction.




Modules:

- **Requirement Upload** → Upload project requirements in PDF format
- **Code Generator** → Enter prompt in a text area to generate code
- **Bug Fixer** → Paste buggy code for automatic corrections
- **Test Generator** → Submit code or requirements for test case generation
- **Code Summarizer** → Analyze and condense code functionality
- **Chatbot** → Interact using natural language for development queries



8. Testing

SmartSDLC incorporates multiple layers of testing to ensure functionality, reliability, and output quality:

-  **Unit Testing:** Validates core utility functions like prompt formatting and output sanitization.
-  **Integration Testing:** Ensures seamless communication between the Streamlit UI and IBM Watsonx service layer.
-  **Manual Testing:** Comprehensive manual testing is performed on all six modules for input handling, user experience, and result accuracy.

Tests are embedded within the codebase and also generated dynamically using the Test Generator module provided within SmartSDLC.

9. Screenshots

POST

/upload

Classify

Parameters

No parameters

Request body

required

multipart/form-data

file

required

Choose File

SmartSDLC_Complete_Project_Report.pdf

Execute

Clear

Responses

200

Code

Details

Response body

```

{
  "text": "SmartSDLC: AI-Enhanced Software Development Lifecycle Project Documentation\n\nProject ID: LTP2023TMD31987\n\nTeam Size: 4\n\nTeam Lead: Jannalagadda Sai Surendra\n\nTeam Members: Gundathinne Easa Byan Prabhu, Vilem Mennery, Guruprathi Balaji\n\nVid: Introduction\n\nSmartSDLC is an AI-powered software application designed to automate the critical phases of the Software Development Lifecycle (SDLC). It utilizes Python and Streamlit, and integrated with IBM Watson's Watson Assistant 3.3 Instruct model. It transforms textual requirements into code, tests, summaries, and even fixes bugs through natural language interaction.\n\nVid: Project Overview\n\nThe purpose of SmartSDLC is to minimize human effort in software development, and ensure accuracy by offering AI-driven insights at every stage of the SDLC.\n\nVid: Features\n\n1. Requirement Elicitation & Classification\n2. AI Code Generation
          
```

POST

/generate_code

Generate Code

Parameters

No parameters

Request body

required

application/json

Edit Value

Schema

```

{
  "story": "sum of two numbers in python"
}
          
```

Execute

Clear

Responses

200

Code

Details

Response body

```

{
  "story": "sum of two numbers in python"
}
          
```

POST

/fix_bug

Fix Bug

Try it out

Parameters

No parameters

Request body required

application/json

Example Value | Schema

```
{
  "code": "string",
  "msg": "string"
}
```

Responses

Code	Description	Links
200	Successful Response	No links
Media type: application/json		
Controls Accept header.		
Example Value Schema		
<pre>"string"</pre>		
422	Validation Error	No links
Media type: application/json		
Example Value Schema		
<pre>{</pre>		

POST

/generate_tests

Generate Tests

Try it out

Parameters

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		
<pre>"string"</pre>		
422	Validation Error	No links
Media type: application/json		
Example Value Schema		
<pre>{</pre>		

POST

/summarize

Summarize Code

Try it out

Parameters

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		
<pre>"string"</pre>		
422	Validation Error	No links
Media type: application/json		
Example Value Schema		
<pre>{</pre>		



10. Known Issues

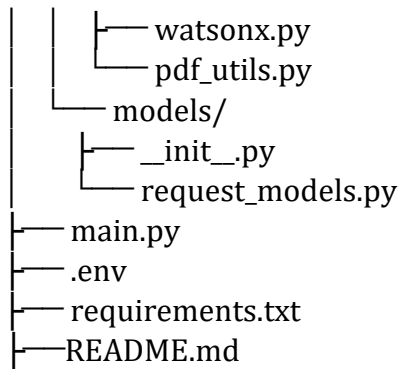
- No persistent user login system
- No database support (all session-based)
- No role-based access or advanced error handling
- IBM Watsonx API has rate limits depending on your cloud plan

11. Future Enhancements

- Add persistent database (MongoDB, PostgreSQL)
- Dockerize for CI/CD deployment
- Implement role-based login system
- Extend to support software architecture generation
- Add support for audio-based prompts or file-to-code generation

12. Folder Structure

```
smart_sdlc/  
├── app/  
│   ├── __init__.py  
│   ├── routes/  
│   │   ├── __init__.py  
│   │   ├── upload.py  
│   │   ├── codegen.py  
│   │   ├── bugfix.py  
│   │   ├── tests.py  
│   │   ├── summarize.py  
│   │   └── chat.py  
│   └── services/  
│       └── __init__.py
```



13. Modules Breakdown

Each module uses the `ask_watsonx(prompt)` function to interact with the IBM Granite model:

- Requirement Classifier → Converts uploaded PDFs into user stories
- Code Generator → Translates natural language prompts into executable code
- Bug Fixer → Accepts code with errors and returns corrected code
- Test Generator → Generates unit test cases for given functions or code blocks
- Summarizer → Describes the functionality of code snippets
- Chat Assistant → Provides conversational answers on SDLC and project-related queries

14. 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

15. Conclusion

SmartSDLC effectively leverages AI to streamline and accelerate software development processes such as requirements gathering, code generation, testing, and documentation. Its modular design and prompt-driven AI interaction make it adaptable for future growth.

Future versions aim to expand into DevOps automation, mobile-friendly interfaces, and GitHub integration for continuous development workflows.