SmartSDLC: AI-Enhanced SDLC Assistant Using IBM Watsonx

1. Introduction

Project Title: SmartSDLC: AI-Enhanced SDLC Automation Platform

Team ID : LTVIP2025TMID34234

Team Members:

Team Leader: Mohammad Luqman

Team member: Maradana Jnana Prasanna

Team member: Margani Kumar Arjun

Team member: Mangalapurapu Eswari

2. Project Overview

SmartSDLC utilizes IBM Watsonx's Generative AI to enhance and automate the Software Development Lifecycle. It transforms how developers handle requirements, code generation, bug fixing, testing, and documentation.

Key Features:

• Requirement Classification: Extracts and classifies requirements from PDF into SDLC phases.

• Code Generator: Converts user stories into production-ready Python code.

• Bug Fixer: Accepts buggy code and returns corrected versions.

• Test Case Generator: Generates unit test cases using natural language prompts.

• Code Summarizer: Provides clear explanations of complex code.

• AI Chatbot: A floating chatbot that answers SDLC-related queries.

3. Architecture

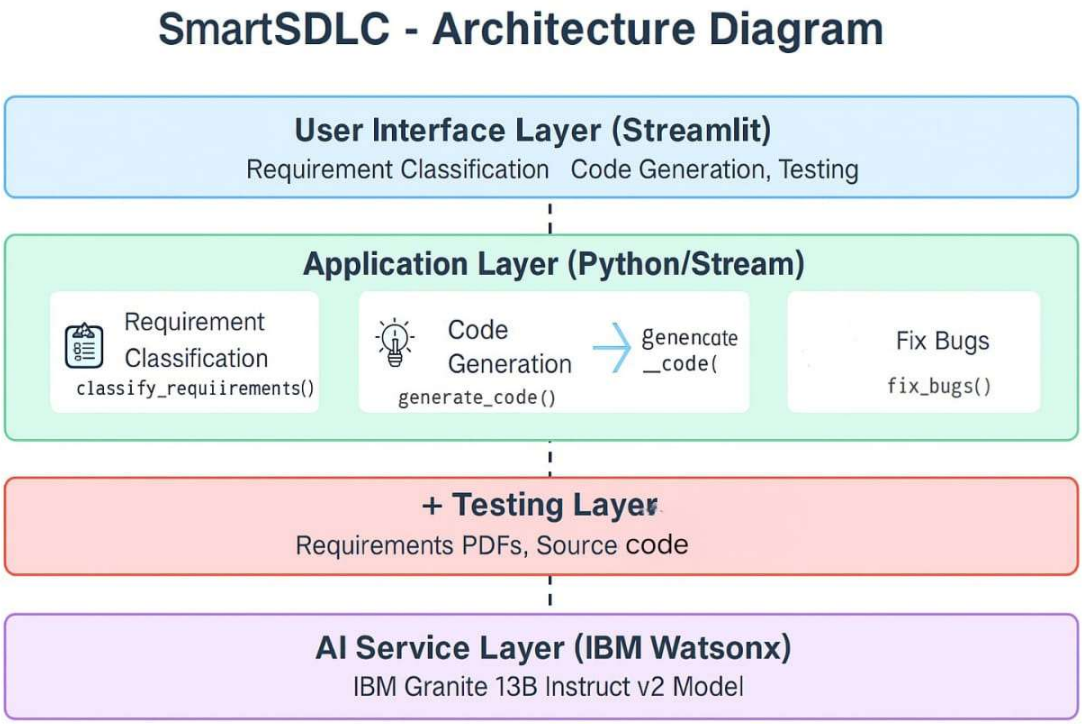
The architecture includes four main layers:

• UI Layer (Streamlit): User interaction via sidebar navigation and dynamic components.

• Application Layer: Handles PDF processing, prompt creation, and AI calls.

• AI Service Layer: Powered by IBM Watsonx Granite 13B Instruct v2.

• Session Layer: Manages stateful interactions across modules.



4. Setup Instructions

1. Install Python 3.8+ and pip.

2. Create and activate a virtual environment.

3. Install dependencies: streamlit, pymupdf, ibm-watsonx-ai, python-dotenv.

4. Create a .env file with IBM_WATSONX_API_KEY and URL.

5. Run the app with: streamlit run app.py

5. API Documentation

The system uses IBM Watsonx Granite 13B via the ibm-watsonx-ai SDK. The ask_watsonx() function manages prompts for all modules.

The backend "API" in this project is primarily the interface with the IBM Watson Machine Learning service. There are no traditional REST API endpoints exposed by a separate backend server (like Node.js/Express.js) as this is a Streamlit-based application directly interacting with the AI service. The primary "API" calls are made internally to the IBM Watson Machine Learning service: ¬ Service: IBM Watson Machine Learning ¬ ModelUsed: IBMGranite 13B Instruct v2 (ibm/granite-13b-instruct v2) ¬ Authentication: API Key and Endpoint URL, managed securely via .env file and loaded by python-dotenv.

6. Authentication

API credentials are securely stored in a .env file and accessed using python-dotenv. These are used to authenticate with Watsonx AI services.

Authentication with the IBM Watson Machine Learning service is handled via an API Key.

- TheAPIKeyisobtained from the IBM Cloud account associated with the Watson Machine Learning service instance.

- Itis securely stored as an environment variable in the .env file within the project directory.
- Thepython-dotenv library loads this API key at application startup.
- Theibm_watson_machine_learning.credentials.Credentials class uses this API key along with the service URL to authenticate with the IBM Watson Machine Learning API when initializing the Model object. This ensures secure communication without hardcoding sensitive information

7. User Interface

The user interface of SmartSDLC is designed using the Streamlit framework, emphasizing modularity, clarity, and ease of interaction for developers, testers, and project managers.

Main Application Layout

- Page Title & Layout:
  The application sets a custom page title ("SmartSDLC – AI Assistant") and uses a wide layout for enhanced code readability and output formatting.

- Sidebar Navigation:
  A prominent sidebar (st.sidebar.radio) allows users to switch between six major modules:

    - Requirement Classification
    - Code Generator
    - Bug Fixer
    - Test Case Generator
    - Code Summarizer
    - Floating AI Chatbot

- Session Management:
  st.session_state is used to retain chat history and form input across module switches, ensuring a consistent experience.

- UI Customization:
  Though Streamlit does not support native CSS styling out-of-the-box, optional custom CSS can be added using st.markdown with unsafe_allow_html=True for visual enhancement if needed.

Feature-Specific Interfaces

Each module in SmartSDLC is implemented as an isolated interface, optimizing user interaction for its specific task:

- Requirement Classification
    - Upload interface for PDF requirement documents.
    - Output is shown in a structured format grouped by SDLC phases (Requirement, Design, Development, Testing, Deployment).
- Code Generator
    - Input form for user stories or feature descriptions.

    - Output is shown using st.code() with Python syntax highlighting.
- Bug Fixer
    - Text area for pasting buggy code (Python or JavaScript).
    - AI-generated corrected code is returned and displayed with syntax highlighting.
- Test Case Generator
    - Input area for code or functional requirements.
    - Output includes unit tests using unittest or pytest format.
- Code Summarizer
    - Code input area.
    - Output is a plain-text functional explanation of the given code snippet.
- AI Chatbot Assistant
    - Chat interface using st.chat_input() and st.chat_message().
    - Maintains multi-turn interactions using session state.

Dynamic Visualizations (optional / extendable)

Currently, SmartSDLC focuses on text and code-based interfaces. However, the platform supports dynamic visualizations for future enhancements:

- Possible Future Enhancements:
    - Prompt Heatmap / Usage Graphs: Track which modules are used most frequently.
    - Error Type Distribution: Visual breakdown of common bug categories (e.g., syntax vs logical).
    - Test Coverage Indicators: Visual test case generation metrics.

8. Testing

The SmartSDLC project follows a structured testing strategy to ensure functionality, reliability, and a smooth user experience across all AI-assisted SDLC modules.

Unit Testing (Conceptual)

- Focuses on verifying the correctness of individual functions, such as:
    - ask_watsonx(): Ensures it sends well-structured prompts and returns valid AI responses.
    - PDF parsing using PyMuPDF: Tests that requirement text is accurately extracted from uploaded PDFs.
    - Session state handling in Streamlit: Confirms chat history and module input/output persist across user actions.

Integration Testing

- Validates the end-to-end workflow between frontend (Streamlit) and backend (IBM Watsonx AI):
    - Ensures correct prompt construction for each module (code generator, bug fixer, etc.).
    - Verifies successful API calls to the Watsonx Granite 13B model and appropriate output rendering.
    - Tests module switching and ensures that state is preserved without breaking the application.

User Acceptance Testing (UAT)

- Involves manual testing by actual users (developers, testers, or mentors) to ensure the tool:
    - Meets defined expectations for each SDLC task.
    - Generates relevant and context-aware responses from natural language inputs.
    - Provides a user-friendly experience in terms of navigation, output clarity, and functionality.

Error Handling Tests

- Verifies robustness of the application by simulating error scenarios:
    - API Errors: Broken Watsonx connection or invalid credentials handled via try-except blocks.
    - Input Validation: Checks for empty or invalid user input (e.g., no code entered before clicking “Fix Code”).
    - Session Failures: Confirms fallback behavior if session state is not initialized or corrupted.

9. Known Issues

• Session state lost on app refresh.
• No user authentication implemented.
• API limits may affect high-volume use.

10. Future Enhancements

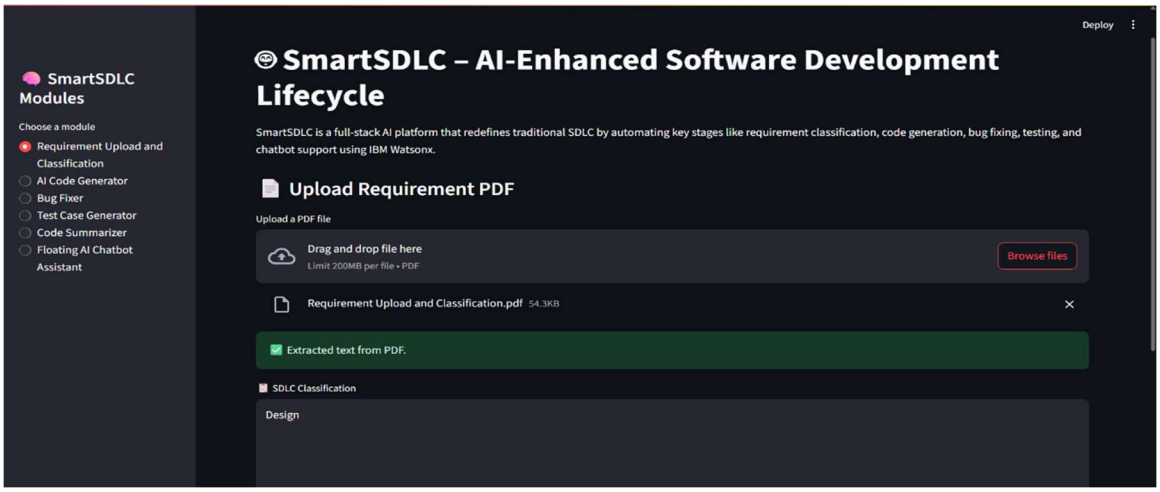
• User login system with profile persistence.

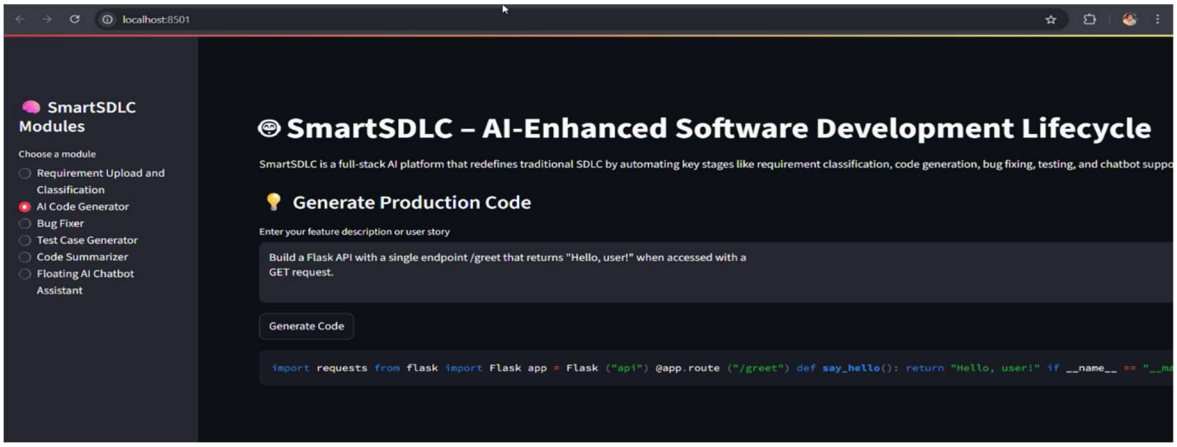
• Integration with GitHub for code suggestions.

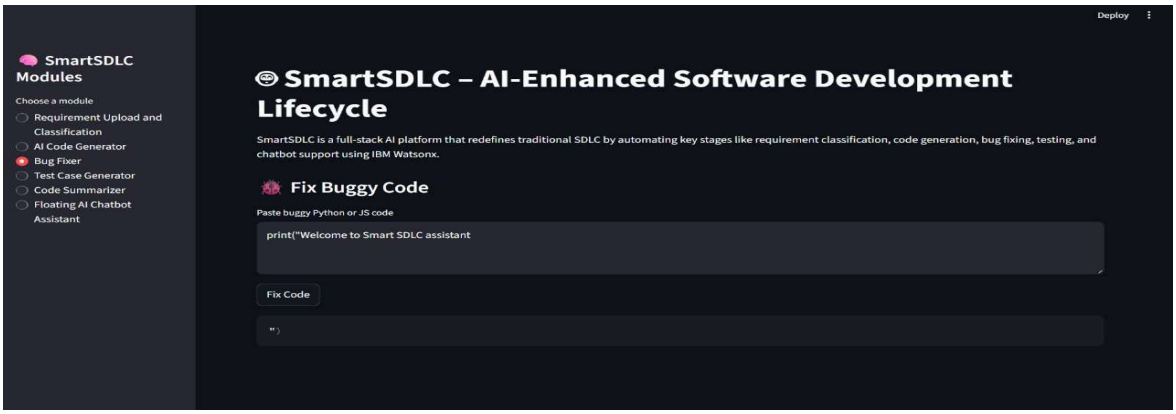
• Advanced analytics on test coverage and bug history.

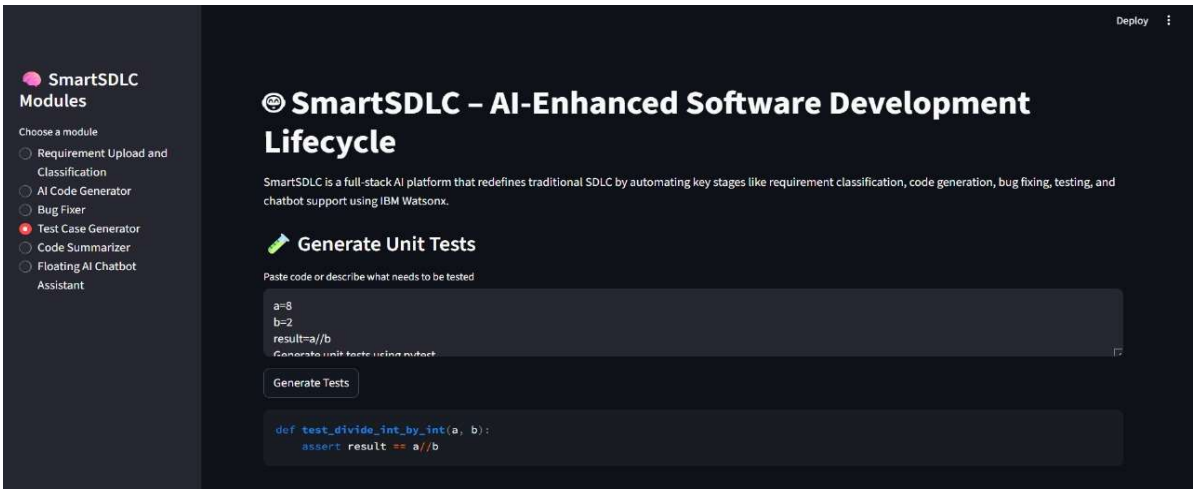
• Expand support for multiple programming languages.

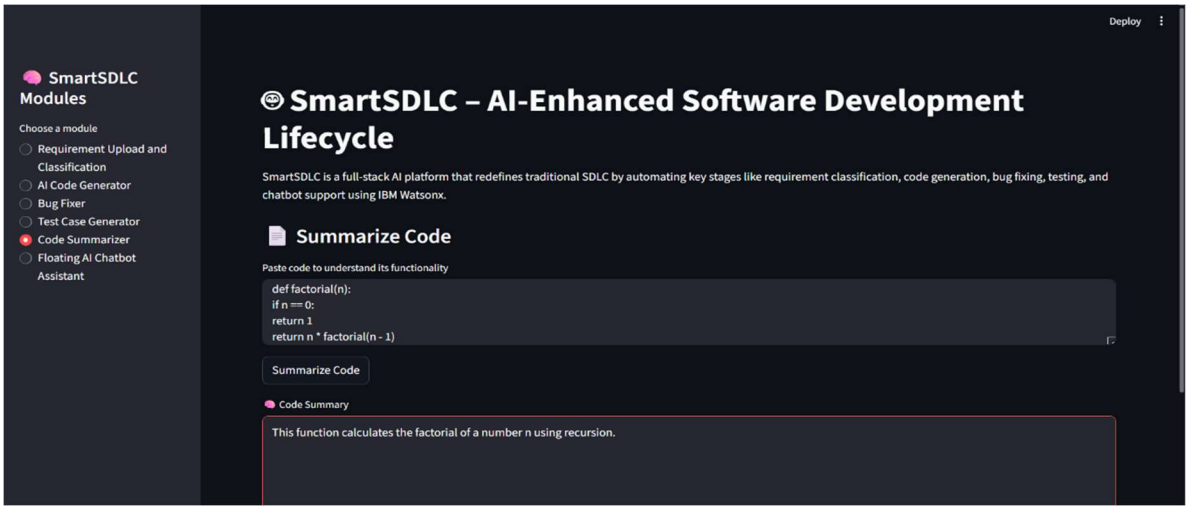
11. Screenshots

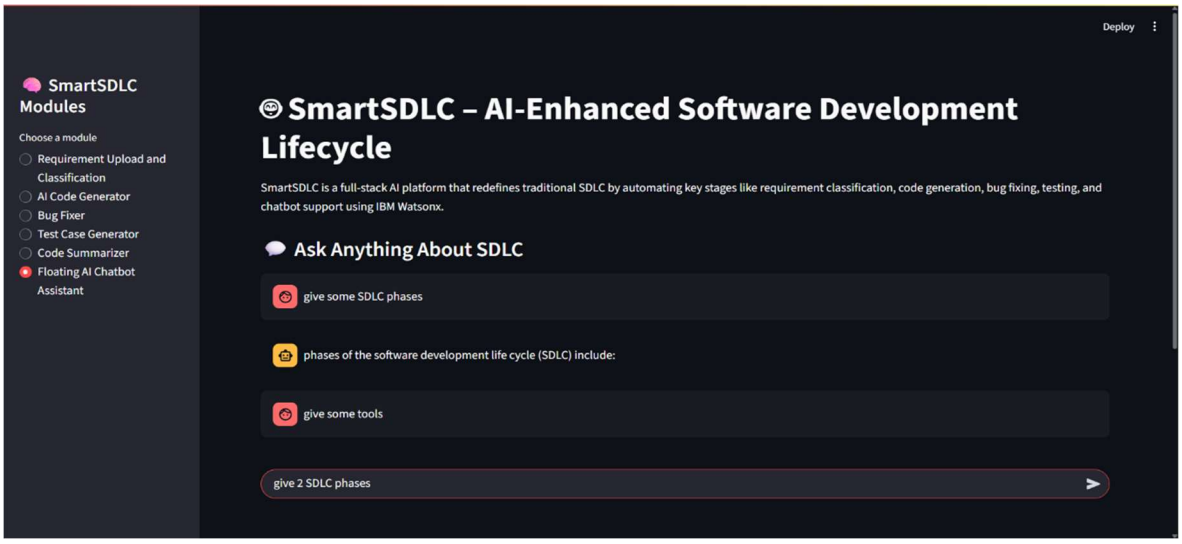







Deploy
SmartSDLC Modules
Choose a module
Requirement Upload and Classification
AI Code Generator
Bug Fixer
Test Case Generator
Code Summarizer
Floating AI Chatbot Assistant
SmartSDLC – AI-Enhanced Software Development Lifecycle
SmartSDLC is a full-stack AI platform that redefines traditional SDLC by automating key stages like requirement classification, code generation, bug fixing, testing, and chatbot support using IBM Watsonx.
Generate Unit Tests
Paste code or describe what needs to be tested
a=8
b=2
result=a//b
Generate Tests
def test_divide_int_by_int(a, b):
assert result == a//b


Deploy
SmartSDLC Modules
Choose a module
Requirement Upload and Classification
AI Code Generator
Bug Fixer
Test Case Generator
Code Summarizer
Floating AI Chatbot Assistant
SmartSDLC – AI-Enhanced Software Development Lifecycle
SmartSDLC is a full-stack AI platform that redefines traditional SDLC by automating key stages like requirement classification, code generation, bug fixing, testing, and chatbot support using IBM Watsonx.
Summarize Code
Paste code to understand its functionality
def factorial(n):
if n == 0:
return 1
return n * factorial(n - 1)
Summarize Code
Code Summary
This function calculates the factorial of a number n using recursion.


Deploy
SmartSDLC Modules
Choose a module
Requirement Upload and Classification
AI Code Generator
Bug Fixer
Test Case Generator
Code Summarizer
Floating AI Chatbot Assistant
SmartSDLC – AI-Enhanced Software Development Lifecycle
SmartSDLC is a full-stack AI platform that redefines traditional SDLC by automating key stages like requirement classification, code generation, bug fixing, testing, and chatbot support using IBM Watsonx.
Ask Anything About SDLC
give some SDLC phases
phases of the software development life cycle (SDLC) include:
give some tools
give 2 SDLC phases