

SmartSDLC: AI-Enhanced Software Development Lifecycle

Project Documentation

Team ID: LTVIP2025TMID31232

Team Size: 4

Team Leader : Raga Loknath

Team Member: Mohammad Ismail Shaik

Team Member: M Vardhan

Team member: Kuruba Roopavathi



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

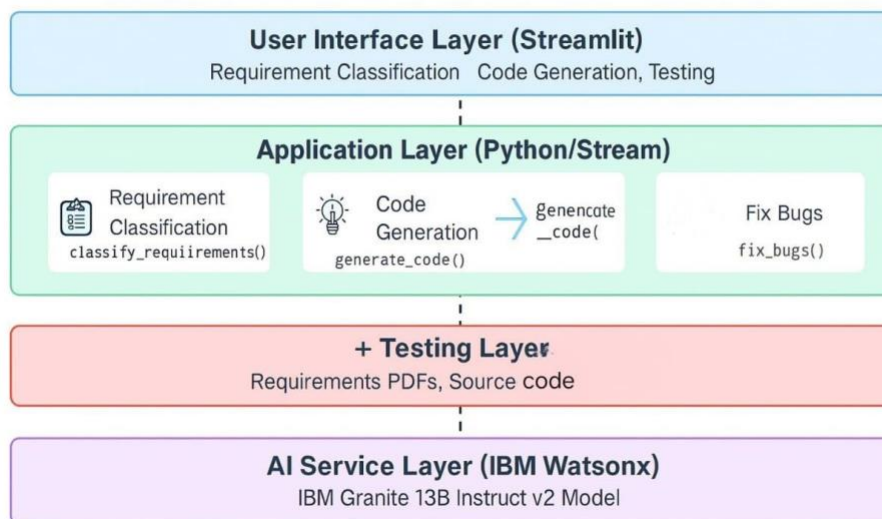
Each feature integrates IBM's Granite model to interpret user input and generate relevant code or insights.

3. Architecture

SmartSDLC follows a clean, modular architecture:

1. ****User Interface (Streamlit)****: Collects input, displays results, and manages interactions.
2. ****Application Logic (Python)****: Processes user commands, forms AI prompts, and handles session state.
3. ****AI Service Layer (IBM Watsonx)****: IBM Granite 3.3 Instruct model generates context-aware outputs.
4. ****Temporary State Memory****: User session and intermediate data stored in-memory using Streamlit session state.

SmartSDLC - Architecture Diagram



4. Setup Instructions ****Prerequisites****:

- Python 3.8 or above
- IBM Cloud account with Watsonx access

- Streamlit, pandas, python-dotenv, ibm-watsonx-ai

****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``
4. Install dependencies: ``pip install -r requirements.txt``
5. Create ``.env`` file with the following:
IBM_API_KEY="your_key"
PROJECT_ID="your_project_id"
BASE_URL="https://eu-de.ml.cloud.ibm.com"
6. Run the app: ``streamlit run SMART_SDLC.py``

5. API Documentation

SmartSDLC does not expose traditional REST APIs but uses IBM Watsonx's ``generate_text()`` method via the Python SDK.

****Example Prompt Usage**:**

- Code Generation: "Generate Python code that implements a login system."
- Bug Fixing: "Fix the following code:
[BUGGY CODE]"
- Testing: "Write pytest unit tests for the given function."

****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 stored in a ``.env`` file and loaded using ``python-dotenv``.

****Security Practice**:**

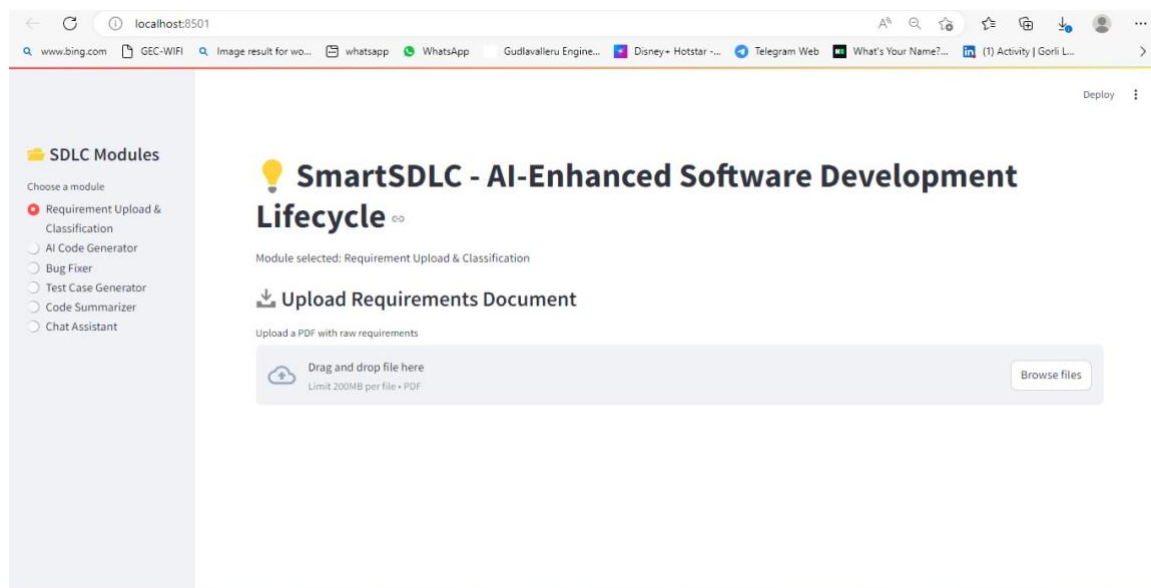
- Do not hardcode API keys.
- Use `.env` and `.gitignore` to prevent accidental exposure.
- Only load variables at runtime.

7. User Interface

The app features a sidebar for navigation and module selection. Each module accepts different input types and shows results using `st.code`, `st.text_area`, or `st.chat_input`.

Modules:

- Requirement Upload → file uploader
- Code Generator → text area prompt
- Bug Fixer → code input
- Test Generator → requirement/code input
- Code Summarizer → code input
- Chatbot → natural language Q&A



8. Testing

SmartSDLC includes several test mechanisms:

- Unit Testing: For prompt creation, output cleaning functions.
- Integration Testing: Streamlit frontend with IBM Watsonx API.
- Manual Testing: For all 6 features (input validation, output quality).

Tests are either in-code validation or handled by test cases generated using SmartSDLC's own test generator module.

9. Screenshots

The screenshots show the SmartSDLC - AI-Enhanced Software Development Lifecycle interface. The left sidebar lists SDLC Modules: Requirement Upload & Classification (selected), AI Code Generator, Bug Fixer, Test Case Generator, Code Summarizer, and Chat Assistant. The main content area shows the 'Upload Requirements Document' section, where a PDF file named 'requirements.pdf' (7.6KB) has been uploaded. Below this, the 'Structured User Stories by SDLC Phase' section displays a list of user stories categorized by phase (Requirements, Design, Development, Testing, Deployment). The interface includes a 'Deploy' button in the top right corner.

SmartSDLC - AI-Enhanced Software Development Lifecycle

Module selected: Requirement Upload & Classification

Upload Requirements Document

Upload a PDF with raw requirements

Drag and drop file here
Limit 200MB per file • PDF

Browse files

requirements.pdf 7.6KB

Text extracted from PDF.

Structured User Stories by SDLC Phase

Ensure the application is responsive and accessible.
Integrate a payment gateway for secure transactions.

1. Requirements:

As a developer, I want to implement unit testing for all API endpoints to ensure their functionality and reliability.
As a deployment team, I want to deploy the application on AWS Lambda to ensure scalability and cost-effectiveness.

2. Design:

As a designer, I want to ensure the application's user interface is responsive and accessible to accommodate various devices and users with disabilities.
As an architect, I want to design a scalable and secure REST API for product listing and cart management.

3. Development:

As a developer, I want to integrate a payment gateway to facilitate secure transactions within the application.

4. Testing:

As a tester, I want to validate the API endpoints for functionality, performance, and security to ensure a smooth user experience.

5. Deployment:

As a deployment team, I want to ensure a seamless deployment process with minimal downtime on AWS Lambda.

This structured user story format helps categorize the requirements into the respective SDLC phases and makes it easier to track progress and manage tasks.

📁 SDLC Modules

Choose a module

☐ Requirement Upload & Classification

☒ AI Code Generator

☐ Bug Fixer

☐ Test Case Generator

☐ Code Summarizer

☐ Chat Assistant

Deploy

💡 SmartSDLC - AI-Enhanced Software Development Lifecycle

Module selected: AI Code Generator

📄 Generate Code from Prompt

Describe what the code should do

Build a Python function that takes a list of integers and returns only the even numbers.

Generate Code

Here is the implementation of the function:

```
def get_even_numbers(numbers):  
    return [num for num in numbers if num % 2 == 0]
```

📁 SDLC Modules

Choose a module

☐ Requirement Upload & Classification

☐ AI Code Generator

☒ Bug Fixer

☐ Test Case Generator

☐ Code Summarizer

☐ Chat Assistant

Deploy

💡 SmartSDLC - AI-Enhanced Software Development Lifecycle

Module selected: Bug Fixer

🔧 Fix Buggy Code

Paste your buggy code here

```
def add_numbers(a, b)  
    return a + b
```

Fix Code

Here is the corrected code:

```
def add_numbers(a, b):  
    return a + b
```

The corrected code is:
The colon (:) was missing at the end of the function definition line in the erroneous code. This colon indicates the start of the

📁 SDLC Modules

Choose a module

☐ Requirement Upload & Classification

☐ AI Code Generator

☐ Bug Fixer

☒ Test Case Generator

☐ Code Summarizer

☐ Chat Assistant

Deploy

Lifecycle

Module selected: Test Case Generator

📄 Generate Test Cases

Paste the function or requirement for test generation

```
def multiply(x, y):  
    return x * y
```

Generate Test Cases

```
1. Test case for multiplying positive integers  
2. Test case for multiplying negative integers  
3. Test case for multiplying a positive integer by a negative integer  
4. Test case for multiplying zero by any integer  
5. Test case for multiplying a large integer by another large integer  
6. Test case for multiplying by zero (edge case)
```

Here are the test cases using Python's unittest:

```
'''Author
```

<

SDLC Modules

Choose a module

☐ Requirement Upload & Classification

☐ AI Code Generator

☐ Bug Fixer

☒ Test Case Generator

☐ Code Summarizer

☐ Chat Assistant

Deploy

Here are the test cases using Python's unittest:

```
'''python
import unittest

def multiply(x, y):
    return x * y

class TestMultiply(unittest.TestCase):

    def test_multiply_positive(self):
        self.assertEqual(multiply(5, 3), 15)

    def test_multiply_negative(self):
        self.assertEqual(multiply(-5, -3), 15)

    def test_mixed_signs(self):
        self.assertEqual(multiply(-5, 3), -15)

    def test_multiply_zero(self):
        self.assertEqual(multiply(0, 10), 0)

    def test_large_integers(self):
        self.assertEqual(multiply(123456789, 987654321), 121932631112635269)

    def test_multiply_by_zero(self):
        self.assertEqual(multiply(5, 0), 0)
```

<

SDLC Modules

Choose a module

☐ Requirement Upload & Classification

☐ AI Code Generator

☐ Bug Fixer

☒ Test Case Generator

☐ Code Summarizer

☐ Chat Assistant

Deploy

self.assertEqual(multiply(5, 0), 0)

```
if __name__ == '__main__':
    unittest.main()
'''

And here are the test cases using pytest:

'''python
def multiply(x, y):
    return x * y

def test_multiply_positive():
    assert multiply(5, 3) == 15

def test_multiply_negative():
    assert multiply(-5, -3) == 15

def test_mixed_signs():
    assert multiply(-5, 3) == -15

def test_multiply_zero():
    assert multiply(0, 10) == 0

def test_large_integers():
    assert multiply(123456789, 987654321) == 121932631112635269
```

<

SDLC Modules

Choose a module

☐ Requirement Upload & Classification

☐ AI Code Generator


☐ Bug Fixer

☐ Test Case Generator


☒ Code Summarizer

☐ Chat Assistant

Deploy

 **SmartSDLC - AI-Enhanced Software Development Lifecycle**


Module selected: Code Summarizer

 **Summarize Code**

Paste the code to summarize

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n-1)
```

Summarize

 Summary

The provided Python function, 'factorial(n)', is a recursive implementation of the factorial calculation. It takes an integer 'n' as input and computes the factorial of that number. The factorial of a non-negative integer 'n' is the product of all positive integers less than or equal to 'n'.

Here's a breakdown of the function:

- The base case checks if 'n' is equal to 0. In that situation, the function returns 1 because 0! (0 factorial) is defined as 1.

SDLC Modules

Choose a module

- ☐ Requirement Upload & Classification
- ☐ AI Code Generator
- ☐ Bug Fixer
- ☐ Test Case Generator
- ☒ Code Summarizer
- ☐ Chat Assistant

```
def factorial(n):
    return n * factorial(n-1)
```

Summarize

Summary

Here's a breakdown of the function:

1. The base case checks if 'n' is equal to 0. In that situation, the function returns 1 because 0! (0 factorial) is defined as 1.
2. If 'n' is not 0, the function calls itself with the argument 'n-1' and multiplies the result by 'n'. This recursive call continues until the base case is reached.

Use case: The factorial function is useful in various mathematical and statistical applications, such as combinatorics, permutations, and probability. For instance, in calculating the number of ways to arrange 'n' distinct items, or the probability of a specific sequence of events occurring.

Example:
factorial(5) # Returns 120, as 5! = 5 * 4 * 3 * 2 * 1 = 120

In summary, the given Python code snippet presents a recursive implementation of the factorial function, which computes the factorial of a given non-negative integer using the mathematical definition of factorial. This function is particularly useful in solving combinatorial problems and other mathematical

Deploy

SDLC Modules

Choose a module

- ☐ Requirement Upload & Classification
- ☐ AI Code Generator
- ☐ Bug Fixer
- ☐ Test Case Generator
- ☐ Code Summarizer
- ☒ Chat Assistant

Module selected: Chat Assistant

Ask Your SDLC Assistant

What is the difference between system testing and integration testing?

System testing and integration testing are both crucial stages in software testing, but they focus on different aspects.

System testing is a high-level testing technique that evaluates the complete and fully integrated software product. It tests the system's compliance with its specified requirements. The goal is to identify defects in the software that may affect the complete system or application. System testing is typically performed after integration testing, when all the individual software modules or components have been successfully integrated.

Integration testing, on the other hand, is a level of software testing where individual software modules are combined and tested as a group. The purpose is to reveal faults in the interaction between these modules. In other words, integration testing verifies that different modules or components of a system work together correctly. It's performed after unit testing and before system testing.

In summary, system testing checks the software product as a whole, while integration testing checks the interfaces and interaction between integrated modules.

Ask anything about SDLC, testing, coding, etc.

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

SmartSDLC/

- |
- |— SMART_SDL.py → Main Streamlit app file
- |— .env → Environment file for secrets
- |— requirements.txt → Dependencies
- |— /data → Optional sample input files
- |— /venv → Python virtual environment

13. Modules Breakdown

Each module calls `ask_watsonx(prompt)` to send instructions to the model.

- Requirement Classifier → PDF-to-user stories
- Code Generator → Prompt-to-code - Bug Fixer → Debug raw code input
- Test Generator → Create unit test cases
- Summarizer → Explain what code does
- Chat Assistant → Open Q&A on SDLC topics

14. Technology Stack

Frontend: Streamlit

Backend: Python

AI Model: IBM Watsonx Granite 3.3 Instruct

PDF Reader: PyMuPDF (fitz)

Authentication: python-dotenv + .env

Deployment Target: IBM Cloud Foundry / Localhost

15. Conclusion

SmartSDLC successfully demonstrates how AI can accelerate software development by automating key stages like planning, coding, testing, and documentation.

Future versions can extend its capabilities into DevOps, mobile responsiveness, and integration with GitHub workflows.