Smart SDLC of Generative AI using IBM Cloud

1. Introduction

Project Title: Smart SDLC of Generative AI using IBM Cloud

Team Members:

• K. Sri Rama Krishna Veni – Streamlit Frontend Developer
• Pedireddi Lakshmi Saroja Sai Veni– Backend Logic & AI Integration
• Nagalla Devi– IBM Cloud Services & Deployment

2. Project Overview

Purpose:
To optimize the Software Development Life Cycle (SDLC) using Generative AI, leveraging the IBM Granite model on IBM Cloud. The goal is to automate tasks like code generation, documentation, and testing while enhancing development speed and quality.

Features:

- • AI-assisted code generation using ibm/granite-13b-instruct-v2
- • Intelligent documentation generation
- • Automated test case generation
- • Real-time UI with Streamlit
- • Cloud-based deployment with IBM Cloud
- • Simple UI for developers to interact with SDLC components

3. Architecture

Frontend:
Built with Streamlit, it provides a minimal and fast interface for developers to:
- Input requirements
- Generate code/documentation
- View outputs in real-time

Backend:
- Python-based backend
- Integrates with IBM's Granite model for AI tasks
- Handles all logic and processing through APIs

Model & Cloud:
- Uses IBM Granite (13B Instruct v2) hosted on IBM Cloud
- Model invoked via API calls for code/test/doc generation

4. Setup Instructions

Prerequisites:
- Python 3.10+
- Streamlit
- IBM Cloud CLI & API Key
- Requests, OpenAI-compatible API libraries

Installation:

```
git clone <your-repo-url>
cd smart-sdlc-ai
pip install -r requirements.txt
streamlit run app.py
```

5. Folder Structure

```
smart-sdlc-ai/
├── app.py             # Streamlit UI
├── services/
│   └── ai_engine.py     # Granite API Integration
├── assets/
│   └── diagrams/        # UI images and SDLC visuals
├── utils/
│   └── helpers.py
├── .env               # Environment variables
└── requirements.txt
```

6. Running the Application

```
streamlit run app.py
```

7. API Documentation

• /generate/code
  - Method: POST

- Input: Requirement text
- Output: Generated Python code

• /generate/test-cases
- Method: POST
- Input: Function description
- Output: Test case code

8. Authentication

Uses IBM API key-based access
- Environment variable setup via .env file
- Keys are securely loaded to authorize model access

9. User Interface

• Simple sidebar for input prompts
• Code & document output display area
• Buttons to clear, save, and export results

10. Testing

• Manual testing for each prompt-output flow
• Validated correctness of AI responses
• Planned: Add unit tests using pytest for AI handler

11. Screenshots or Demo

(Insert generated images here: architecture diagram, idea map, prioritization matrix, etc.)

12. Known Issues

• Occasional latency in model response
• AI outputs sometimes require manual refinement

13. Future Enhancements

- Add login-based user personalization
- Save session history
- Integrate multi-model support (e.g., Mistral, Gemini)
- Automate complete SDLC from input to deploy