Project Title

Project Documentation

1. Introduction

• Project title: SmartSDLC – AI-Enhanced Software Development Lifecycle

Team member: Abinaya E

Team member: Iswariya I

Team member: Devadharshini S

Team member: Aarthi R

2. Project Overview

• Purpose:

The purpose of SmartSDLC is to accelerate and enhance the software development lifecycle using IBM Granite models. It supports requirements generation, code creation, bug fixing, test case generation, documentation, and conversational AI guidance.

The project integrates generative AI into SDLC workflows to increase productivity, reduce errors, and simplify software engineering processes.

• Features:

Requirement Analysis

- Key Point: Automatic requirement extraction

- Functionality: Converts uploaded PDFs into clear, structured requirements.

Code Generation

- Key Point: AI-powered coding

- Functionality: Generates code snippets from natural language prompts.

Test Case Creation

- Key Point: Automation in QA

- Functionality: Generates test cases automatically from requirements or code.

Bug Fixing

- Key Point: Debugging assistance

- Functionality: Analyzes code, detects issues, and suggests fixes.

Documentation Generator

- Key Point: Auto-documentation

- Functionality: Creates software documentation from project details.

AI Chat Assistant

- Key Point: Conversational support

- Functionality: Provides real-time help for developers throughout the SDLC.

3. Architecture

Frontend (Gradio):

The frontend is built with Gradio, providing an easy-to-use interface for uploading files, interacting with the AI assistant, generating code, and reviewing outputs.

Backend (Python + FastAPI):

FastAPI powers backend APIs for requirement extraction, code generation, test case generation,

and debugging assistance.

LLM Integration (IBM Granite):

Granite models from IBM Watsonx (via Hugging Face) are used for requirement summarization,

code generation, bug fixing, and intelligent responses.

Deployment (Google Colab):

The system is deployed on Google Colab using T4 GPUs, ensuring fast execution and easy setup.

4. Setup Instructions

Prerequisites:

o Python 3.9 or later

o Gradio framework installed

o IBM Granite models from Hugging Face

o Git for version control

o Google Colab environment with GPU access

Installation Process:

o Clone the project repository

o Install dependencies from requirements.txt

o Configure Hugging Face and IBM Granite credentials

o Launch the backend with FastAPI

o Run the Gradio interface on Google Colab

o Upload project files and interact with the assistant

5. Folder Structure

app/ – Backend FastAPI logic for requirements, code generation, and bug fixing

ui/ – Gradio components for project interface

smartsdlc.py – Main entry script for the application

granite_llm.py – Handles communication with IBM Granite models

requirement_extractor.py – Extracts requirements from uploaded PDFs

code_generator.py – Generates code from prompts

test_generator.py – Creates automated test cases

bug_fixer.py – Analyzes and fixes code issues

doc_generator.py – Creates auto-generated documentation

6. Running the Application

➢ Launch the FastAPI backend to enable API endpoints

➢ Start the Gradio interface in Google Colab

➢ Upload software requirements or code files

➢ Use the AI assistant for requirement analysis, coding, bug fixing, and test creation

➢ View outputs like generated code, test cases, and documentation

7. API Documentation

Available API Endpoints include:

POST /chat/ask – AI assistant for SDLC queries

POST /upload-req – Uploads PDFs for requirement extraction

POST /generate-code – Generates code from prompts

POST /generate-tests – Creates test cases from inputs

POST /fix-bugs – Analyzes and fixes uploaded code

GET /get-docs – Generates documentation from project files

8. Authentication

For demonstration, the system runs in open mode.

For secure deployment, options include:

• Token-based authentication (JWT or API keys)

• Role-based access (developer, tester, manager)

• OAuth2 integration with IBM Cloud credentials

9. User Interface

The interface is built in Gradio with:

- Requirement upload section

- Chat panel for developer queries

- Code generation and test output tabs

- Debugging results and documentation download options

10. Testing

Testing phases included:

- Unit testing for code and requirement extraction modules

- API testing using Postman and Colab notebooks

- Manual testing for Gradio interface and SDLC workflows

- Edge case handling for incomplete or erroneous inputs

11. Screenshots

[Add screenshots of Gradio interface, requirement extraction, code generation, and bug fixing]

12. Known Issues

- Limited dataset may affect edge-case requirement extraction

- Deployment currently restricted to Google Colab

13. Future Enhancements

- Integration with CI/CD pipelines

- Support for more programming languages

- Expanded bug detection and fixing capabilities

- Role-based dashboards for managers and testers