SmartSDLC – AI-Enhanced Software Development Lifecycle

Project Documentation:

1. Introduction:

Project Title: SmartSDLC – AI-Enhanced Software Development Lifecycle

Team Members:

Member 1:P.Monisha

Member 2:K.preethi

Member 3:P.Pragatheswari

Member 4:K.Sivaranjani

2. Project Overview:

Purpose:

SmartSDLC accelerates and enhances the software development lifecycle by leveraging AI models from IBM’s Granite family. It automates requirement gathering, code generation, test creation, bug fixing, documentation, and provides an AI chat assistant to support developers, ultimately reducing time-to-market and improving software quality.

Features:

Requirement Extraction

Key Point: Automatically generate clear requirements.

Functionality: Upload PDFs or prompts to extract and organize software requirements.

Code Generation from Prompts

Key Point: AI-assisted coding.

Functionality: Converts natural language prompts into functional code snippets.

Automated Test Creation

Key Point: Faster quality assurance.

Functionality: Creates test cases and scripts automatically for generated code.

Bug Fixing Assistant

Key Point: Smart debugging.

Functionality: Identifies issues in code and suggests fixes.

Documentation Writer

Key Point: Simplified knowledge sharing.

Functionality: Generates clear and structured documentation for projects.

AI Chat Helper

Key Point: Interactive developer assistant.

Functionality: Provides real-time help and suggestions within the development environment.

Deployment in Google Colab

Key Point: Easy setup and performance.

Functionality: Runs the entire application in GPU-enabled Google Colab notebooks.

Gradio UI

Key Point: Developer-friendly dashboard.

Functionality: Offers an intuitive interface to interact with all SmartSDLC features.

3. Architecture:

Frontend (Gradio):Interactive web UI where developers can upload files, generate code, run tests, and chat with the AI

assistant.

Backend (IBM Granite Models):

Granite models from Hugging Face handle natural language tasks such as requirement extraction,

code generation, and bug fixing.

Workflow Integration:

Inputs (PDFs, prompts) are processed by the backend model and displayed as structured outputs in

the Gradio UI.

4. Setup Instructions

Prerequisites:

Python 3.9 or later

Gradio framework (pip install gradio)

Transformers & Torch libraries

PyPDF2 library for PDF processing

IBM Granite model from Hugging Face

Google Colab T4 GPU (optional for acceleration)

GitHub account for code hosting

Installation Process:

1. Open Google Colab and create a new notebook named “SmartSDLC”.

2. Change runtime to T4 GPU under Runtime > Change runtime type.

3. Run:

!pip install transformers torch gradio PyPDF2 -q

4. Load the IBM Granite model from Hugging Face.

5. Run the rest of the code to start the application.

6. Click the generated URL to open the Gradio app.

5. Folder Structure:

app/ – Core logic for SmartSDLC AI modules.

ui/ – Gradio-based UI components.

smart_sdlc.py – Main entry script for running the app in Colab or locally.

model_integration.py – Handles communication with IBM Granite models.

requirement_extractor.py – Extracts and organizes requirements from PDFs.

code_generator.py – Generates code snippets from prompts.

test_creator.py – Creates test cases automatically.

bug_fixer.py – Detects and fixes bugs in code.

doc_writer.py – Generates documentation for codebases.

6. Running the Application:

Launch Google Colab and open your SmartSDLC notebook.

Install prerequisites.

Run the cells to start the backend and Gradio frontend.

Click the link to open the live Gradio app.

Upload PDFs, enter prompts, and interact with the AI assistant to generate requirements, code, tests, and documentation.

7. API Documentation:

(If you expose backend endpoints using FastAPI)

POST /extract-requirements – Uploads PDFs and extracts requirements.

POST /generate-code – Converts prompts to code snippets.

POST /create-tests – Generates test cases for provided code.

POST /fix-bugs – Identifies and fixes code issues.

POST /write-docs – Produces documentation for the project.

8. Authentication

For demo purposes, the app runs openly in Google Colab. For production:

Token-based authentication for API calls.

OAuth2 integration with developer credentials.

Role-based access for team members (developer, QA, admin).

9. User Interface:The interface is designed for software teams. It includes:

Sidebar with navigation between requirements, code generation, tests, bug fixing, and documentation modules.

Real-time output display.

Easy export of generated code, tests, and docs.

10. Testing

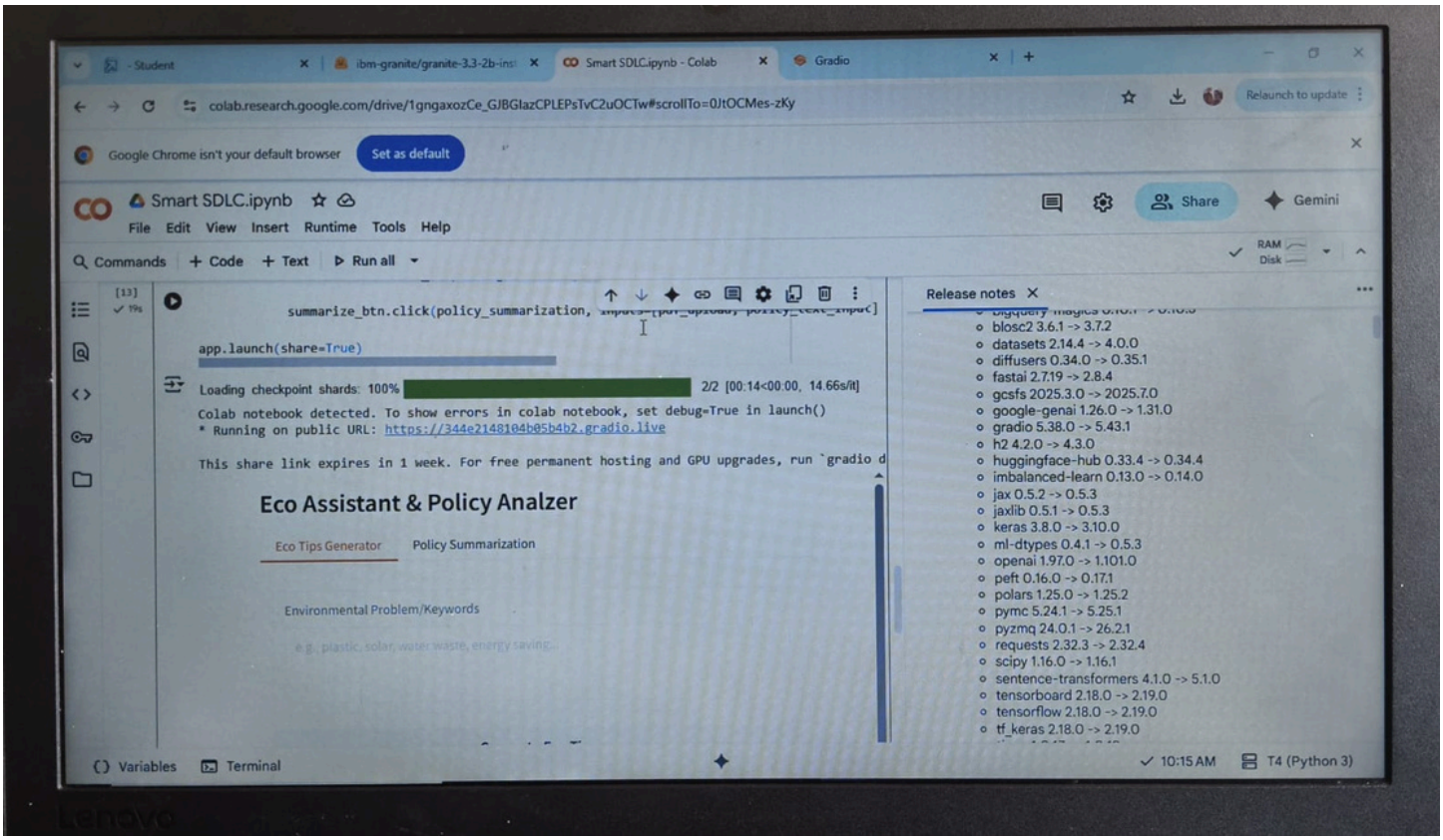
Unit Testing: Validate requirement extraction, code generation, and bug fixing modules.

Integration Testing: Test Gradio UI with backend models.

Manual Testing: Ensure smooth workflow in Google Colab.

Edge Cases: Handle large PDF files or ambiguous prompts.

11. Screenshots:




12. Known Issues:

Limited integration with external development tools.

Requires stable internet for Hugging Face model loading.

13. Future Enhancements:

Integrate with GitHub Actions for CI/CD automation.

Add multilingual support for requirement extraction.

Include code quality scoring and metrics.

Implement secure deployment on cloud platforms.