SmartSDLC – AI Code Analysis & Generator

# 1. Introduction

Project title : SmartSDLC – AI Code Analysis & Generator

Team member : M.R.Harini

Team member : V.Dharshini

Team member : T.Fareen Tawfeeka

Team member : Elavarasi.S

# 2. Project Overview

The SmartSDLC project is an AI-powered assistant that helps in software development life cycle (SDLC) stages. It focuses on two main functionalities:  
  
1. Requirement Analysis – Extracting and organizing requirements into functional, non-functional, and technical categories from either uploaded PDF documents or user-provided text.  
  
2. Code Generation – Automatically generating code snippets in multiple programming languages based on user requirements.  
  
This tool uses Gradio for user interaction, Hugging Face Transformers with IBM Granite LLM for AI, and PyPDF2 for PDF processing.

# 3. Architecture

Frontend (Gradio):  
Provides a web-based user interface with tabs for Requirement Analysis and Code Generation. Users can upload PDFs, input text, select programming languages, and view outputs.  
  
Backend (Python, Hugging Face Transformers, PyTorch):  
Handles AI model loading, requirement analysis, code generation, and PDF text extraction. Uses IBM Granite instruct model for natural language understanding.  
  
Model Integration (Hugging Face LLMs):  
The AI model processes user prompts to generate structured requirement analysis and code.  
  
PDF Processing (PyPDF2):  
Used for extracting text content from uploaded PDF requirement documents.

# 4. Setup Instructions

Prerequisites:  
- Python 3.8 or later  
- pip and virtual environment tools  
- Libraries: torch, transformers, gradio, PyPDF2  
  
Installation Process:  
1. Clone the repository or download smartsdlc.py  
2. Install dependencies: pip install -r requirements.txt (or install manually)  
3. Run the script: python smartsdlc.py  
4. Access the Gradio interface via local browser or shareable link.

# 5. Folder Structure

SmartSDLC/  
│── smartsdlc.py # Main application script  
│── requirements.txt (optional) # List of dependencies

# 6. Running the Application

To run the application:  
1. Execute python smartsdlc.py  
2. The Gradio interface will launch in your browser  
3. Use the 'Code Analysis' tab to upload requirements (PDF/text)  
4. Use the 'Code Generation' tab to generate code snippets in chosen programming language

# 7. API Documentation

The application exposes the following functional APIs through the Gradio interface:  
  
1. Requirement Analysis API  
 - Input: pdf\_file (optional, PDF format), prompt\_text (string)  
 - Output: Structured requirements categorized as Functional, Non-functional, and Technical.  
 - Trigger: Analyze button in Gradio UI  
  
2. Code Generation API  
 - Input: prompt (string), language (string: Python, JavaScript, Java, C++, C#, PHP, Go, Rust)  
 - Output: Auto-generated code snippet in the chosen language  
 - Trigger: Generate Code button in Gradio UI  
  
Utility Functions (internal):  
- generate\_response(prompt, max\_length): Generates AI-powered responses using Granite LLM  
- extract\_text\_from\_pdf(pdf\_file): Extracts raw text from PDF documents

# 8. User Interface

The interface is built with Gradio and includes:  
- Tabs for Code Analysis and Code Generation  
- File upload component for PDFs  
- Textboxes for requirement entry and code prompts  
- Dropdown for selecting programming languages  
- Output areas for displaying analyzed requirements and generated code  
The design is user-friendly and requires no technical expertise to operate.

# 9. Testing

Testing was carried out in multiple stages:  
- Unit Testing: Individual functions like generate\_response, extract\_text\_from\_pdf  
- Functional Testing: Verified requirement analysis and code generation outputs  
- Integration Testing: Confirmed Gradio interface correctly triggers backend functions  
- Manual Testing: Tested with sample requirement PDFs and text inputs  
- Edge Case Handling: Large PDF files, incomplete prompts, unsupported inputs

# 10. Screenshots

Screenshots of the running application (Gradio UI with Requirement Analysis and Code Generation) can be attached here.

# 11. Known Issues

- Generated code may need minor corrections for syntax or logic  
- Requirement analysis accuracy depends on clarity of input  
- Large PDF documents may take longer to process  
- Limited to predefined set of programming languages for code generation

# 12. Future Enhancements

- Expand supported programming languages (e.g., Kotlin, Swift)  
- Allow exporting results in multiple formats (Word, Excel, PDF)  
- Provide advanced configuration options for AI model parameters  
- Add collaboration features (multi-user, version tracking)  
- Improve UI with visual requirement mapping and diagrams