SmartSDLC – AI-Enhenced Software Development Lifecycle

Project Document

1. Introduction

- Project Title: **AI Code Analysis & Generator**
- Team Member : KANAGA K
- Team Member: KASTHURI R
- Team Member: KALAIVANI E
- Team Member: DEVADHARSHINI D

2. Project Overview

Purpose:

Empower developers and analysts by simplifying the process of analyzing requirements and generating code automatically, thus increasing productivity and accuracy.

Key Features:

Requirements Extraction & Analysis

Extracts key software requirements from PDF documents or direct user input.

Code Generation

Generates code snippets based on the analyzed requirements in multiple programming languages such as Python, Java, C++, JavaScript, PHP, Go, Rust, etc.

Gradio Web UI

Easy-to-use interface with clear layout for both analysis and code generation workflows.

3. Architecture

Frontend (Gradio)

Provides an interactive web UI.

Contains two main functional tabs:

1. Code Analysis Tab

- Upload PDF or write text input.
- Button triggers requirement analysis.
- Outputs analyzed requirements.

2. Code Generation Tab

- User inputs code prompt.
- Dropdown to select programming language.
- Button generates code based on prompt.
- Outputs are displayed in scrollable text areas.

Backend (Python, Transformers, PyPDF2)

Model Integration:

- IBM Granite model (ibm-granite/granite-3.2-2b-instruct) for language tasks.
- Hugging Face Transformers used for tokenizer and model loading.

Core Functionalities:

- Generate_response(prompt, max_length): Generates text/code responses using the model.
- Extract_text_from_pdf(pdf_file): Reads and extracts text from uploaded PDF file.
- Requirement_analysis(pdf_file, prompt_text): Combines PDF extraction with prompt-based analysis.
- Code_generation(prompt, language): Generates code snippets from given prompts.

Error Handling:

- Gracefully manages cases of missing PDF, empty input, and model errors

4. Setup Instructions

Install Dependencies:

```
!pip install transformers torch gradio PyPDF2 –q
```

Launch Application:

```
app.launch(share=True)
```

5. Folder Structure

project-root

```
├── app.py              # Main Gradio interface
├── model_utils.py      # Tokenizer & model handling
├── pdf_utils.py        # PDF text extraction utilities
├── code_analysis.py    # Requirement analysis logic
├── code_generation.py  # Code generation logic
├── requirements.txt    # Dependencies list
├── .env                # Environment variables (if needed)
```

6. API Documentation

No explicit API endpoints—uses Gradio blocks for direct user interaction in the web app.

7. Authentication

Currently runs in open mode for demonstration purposes.

Planned enhancements:

- Token-based authentication (e.g., Hugging Face HF_TOKEN)
- OAuth2 integrations for secure deployment.

8. User Interface

Tabs:

- Requirements Analysis
- PDF Upload + Text input
- Analysis button
- Output: Requirements extracted

Code Generation

- Textbox for code prompt
- Dropdown for language selection
- Generate Code button
- Output: Generated code snippet

9. Testing

Manual Testing:

- PDF upload functionality
- Chat interface responses
- Code generation accuracy

Edge Cases Handled:

- Missing file
- Large file uploads
- Empty text input

10.Known Issues

- Missing HF_TOKEN for private Hugging Face models.
- Deprecation warnings for torch_dtype usage.
- Not production-grade; designed for prototyping.

11. Future Enhancements

- Add secure user authentication.
- Extend support to DOCX, TXT inputs.
- Implement session management and history tracking.
- Improve error handling and performance on large files