SmartSDLC – AI-Enhanced Software Development Lifecycle

Project Documentation

1.Introduction

Project title : SmartSDLC – AI-Enhanced Software Development Lifecycle

Team ID :NM2025TMID02154

Team Leader : KARTHIKEYAN S

Team member : PRANESH S P

Team member : SIDDHARTH M C

Team member : SANJAI S

2.Project Overview ■

Purpose:

This project is an AI Code Analysis & Generator web application built using Gradio and Hugging Face Transformers. It leverages the ibm-

granite/granite-3.2-2b-instruct model to perform two primary functions: analyzing software requirements and generating code snippets. Users can either upload a PDF document or type in their requirements for analysis. The application categorizes these requirements into functional, non-functional, and technical specifications. Additionally, it can generate code in various programming languages based on user prompts, streamlining the development process. The application is designed to be accessible via a user-friendly Gradio interface.

3.Architecture ■■

The application's architecture is a simple monolithic structure with a clear separation of concerns, all running within a single Python script.

■ Frontend (UI): The user interface is built using the Gradio library. It provides a web-based, interactive dashboard with separate tabs for "Code Analysis" and "Code Generation". Gradio handles all user input (text boxes, file uploads, dropdowns) and output display.

Backend (Core Logic): The backend is written in Python. It's responsible for:

Model Management: Loading the ibm-granite/granite-3.2-2binstruct model and its tokenizer from Hugging Face. It uses torch for managing the model and utilizes GPU acceleration (device_map="auto") if available.

Requirement Analysis: The requirement_analysis function processes text from either a PDF (via PyPDF2) or a text input field. It then crafts a detailed prompt for the AI model to categorize the requirements.

Code Generation: The code_generation function takes a user prompt and a selected programming language to formulate a specific prompt for the AI model to generate code.

AI Interaction: The generate_response function is the core of the backend. It prepares inputs for the transformer model, runs the inference, and decodes the model's output into a humanreadable response. It manages tokenization, input tensor creation, and GPU acceleration.

4.Setup Instructions ■■

To get the application up and running on your local machine, follow these steps.

Clone the Repository (if applicable): Bash

git clone cd

Create and Activate a Virtual Environment: Bash

python -m venv venv # On Windows

venv\Scripts\activate # On macOS/Linux source venv/bin/activate

Install Dependencies: The provided script requires a few key libraries. Install them using pip. Bash

pip install gradio torch transformers PyPDF2

Note: If you have a CUDA-enabled GPU, you should also install the appropriate version of torch for GPU acceleration. Refer to the PyTorch website for instructions.

5.Folder Structure ■

The project is a single-file application, meaning all the code is contained within one script.

/ai-code-app/

■ README.md

■ app.py # The main application script with all the code

■ app.py: Contains all the logic for the Gradio interface, model loading, requirement analysis, and code generation. This is the only file you need to run.

6.Running the Application ■

Once you have completed the setup, running the application is a single command.

Navigate to the project directory if you are not already there.

Run the Python script: Bash

python app.py

Access the web interface: The script will start a local web server and provide a URL in the console, typically http://127.0.0.1:7860. Open this URL in your web browser. The share=True parameter in the app.launch() command also provides a public URL for sharing, which will be valid for a limited time.

7.API Documentation ■

This application does not have a traditional REST API. It is an interactive web application that uses Gradio's backend to handle function calls from the UI. The

"API" is effectively the direct invocation of the Python functions requirement_analysis and code_generation via the Gradio interface.

Endpoint for Analysis: The Gradio UI button analyze_btn triggers the requirement_analysis function.

Inputs: pdf_file (a Gradio gr.File object) and prompt_text (a gr.Textbox string).

Outputs: A string of text containing the analyzed requirements.

Endpoint for Code Generation: The generate_btn button triggers the code_generation function.

Inputs: code_prompt (a gr.Textbox string) and language_dropdown (a gr.Dropdown string).

Outputs: A string of generated code.

8.Authentication ■

The current application does not implement any authentication or authorization. It is designed as a standalone, public-facing tool for demonstration and personal use. Any user who can access the URL can use its full functionality. For a production environment, this would need to be integrated with an authentication system like OAuth, JWTs, or a simple username/password model.

9.User Interface ■

The user interface is built with Gradio, which automatically creates a clean and responsive design.

Tabs: The interface is organized into two distinct tabs:

Code Analysis Tab: Contains a file upload component for PDFs, a text area for manual input, and a button to trigger the analysis. The output is displayed in a large, scrollable text box.

Code Generation Tab: Includes a text area for describing the desired code, a dropdown menu to select the programming language, and a button to generate the code. The generated code is shown in a separate text box.

Aesthetics: Gradio's default theme is used, providing a modern, minimalist design that is intuitive to navigate.

10.Testing

Given the nature of this project, testing can be approached from several angles.

Functional Testing: Manually test the two main functions of the application:

Analysis: Upload a PDF with requirements and see if the output is correctly categorized. Type requirements directly into the text box and check if the analysis is accurate.

Generation: Provide a clear prompt for a code snippet (e.g., "Write a Python function to reverse a string") and verify that the generated code is correct and in the selected language.

Integration Testing: Ensure that the Gradio UI correctly communicates with the backend Python functions. Verify that button clicks trigger the right functions and that the outputs are displayed correctly on the web page.

Performance Testing: Monitor the application's response time, especially when generating long responses or processing large PDF files.

Check if the GPU acceleration is being utilized as expected.

Error Handling: Test the application's behavior with invalid inputs, such as uploading a non-PDF file, leaving the input fields blank, or providing ambiguous prompts. The current implementation includes a try-except block for PDF reading, which is a good starting point.

Screen shots 1.Input

2.Output

This project successfully demonstrates how a powerful AI model can be used to streamline key parts of the software development process. By using Gradio, we were able to quickly build a user-friendly application that can analyze software requirements and generate code snippets.

This application is a strong proof-of-concept for how AI can act as an intelligent assistant for developers, boosting productivity and making the coding process more efficient. While it's currently a simple tool, the project lays the groundwork for more advanced features in the future.