***CodeGenie: AI for Smarter SDLC***

***SmartSDLC – AI-Enhanced Software Development Lifecycle***

**Project Documentation**

**1. Introduction**

* Project title **: CodeGenie: AI for Smarter SDLC**
* Team ID : NM2025TMID11077
* Team member : ELAMATHI E
* Team member : VINUPRITHA G
* Team member : SARANYA S

**2. Project Overview**

* **Purpose**:

The purpose of CodeGenie is to streamline the Software Development Life Cycle (SDLC) by providing an interactive AI-powered assistant for requirement analysis and code generation. Using IBM Granite models and Gradio, CodeGenie enables both developers and project managers to quickly analyze software requirements from text or PDF files, generate organized documentation, and produce code snippets in various languages instantly.

* **Features**:
* **Conversational Interface:**
* Natural language interaction via Gradio UI
* Ask questions, get instant code and analysis
* **Requirement Analysis Module:**
* Extracts and organizes functional/non-functional requirements from text or PDF files
* **Code Generation Module:**
* Multi-language code generation using IBM Granite models (Python, JavaScript, etc.)
* **PDF Processing:**
* Extracts and analyzes data from uploaded PDFs
* **User-Friendly Interface:**
  + - * Dashboard with tabbed layout for both code analysis and generation
      * Real-time input/output with Gradio Blocks

**3. Architecture**

* **Frontend (Gradio):**

Built with Gradio (Python library) for an interactive web-based UI

* **Code Analysis Tab**
* Upload PDF or write text input.
* Button triggers requirement analysis.
* Outputs analyzed requirements.
* **Code Generation Tab**
  + - User inputs code prompt.
    - Dropdown to select programming language.
    - Button generates code based in prompt.
* **Backend (Local Model Serving):**
  + - Utilizes Hugging Face Transformers to load IBM Granite LLM for inference
    - Handles prompt encoding, model interaction, and output decoding
* **Integration**:
  + - PDF handling via PyPDF2
    - Language modeling via IBM Granite LLM
    - Deployment with Gradio's share feature for remote accessibility
* **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**

* **Prerequisites:**
  + Python 3.9 or later
  + pip and virtual environment tools
  + Internet access for model download (Hugging Face)
  + GPU device recommended (optional, for faster inference)
* **Installation Process:**
* Clone the project repository
* Install dependencies via requirements.txt (gradio, transformers, torch, PyPDF2)
* Launch app with python app.py
* Interact via local browser or via the remote public link

**5. Folder Structure**

* **Text**

app.py # Main Gradio app source code

requirements.txt # Dependency list for Python environment[13]

README.md # Optional project overview for Hugging Face viewers

* app.py: Contains all UI logic, code analysis/generation modules
* requirements.txt: Lists required libraries (gradio, transformers, torch, PyPDF2)
* README.md: Brief description of project purpose and setup

**6. Running the Application**

**To start the project:**

* Launch the app via python app.py
* Gradio will open an interactive UI in the browser
* Navigate between "Code Analysis" and "Code Generation" via tabs
* Upload documents, enter prompts, select programming language, generate outputs in real-time

**7. API Documentation**

* **Core UI Logic:**
  + Requirement Analysis: requirement\_analysis(pdf\_file, prompt\_text)
  + Code Generation: code\_generation(prompt, language)
* **Inputs:**
  + PDF file (.pdf)
  + Text prompt
  + Programming language selection
* **Outputs:**
  + Organized requirements (functional, non-functional)
  + Generated code snippets

**8. Authentication**

* The demo runs in an open environment for local or trial use
* For production, recommend integration with user authentication (API keys, OAuth, etc.)
* Planned enhancements: Permission controls/role-based access, session tracking

**9. User Interface**

**UI is minimalist and functional:**

* + - **Sidebar**: Navigation between modules
    - **Tabbed layout:** Code Analysis, Code Generation
    - Real-time result display in textboxes
    - PDF upload and text input options
    - Programming language dropdown for code output

**10. Testing**

**Testing performed via:**

* **Unit Testing:**

Core functions for prompt handling, PDF extraction

* **Manual Testing:**

UI flows for file uploads, requirement extraction, code generation

* **Edge Case Handling:**

Large files, unsupported formats, prompt validation

**11. Screenshots**

Add screenshots of UI displaying requirement extraction, code generation, PDF upload, etc.

**12. Known Issues**

Long or complex PDFs may cause slow extraction or incomplete text parsing

Code generation limited by model context window and prompt length

Model license and deployment restrictions may apply for IBM Granite

**13. Future Enhancement**

* Add support for additional languages and frameworks
* Integrate more advanced requirement parsing with domain models
* Production-grade authentication and logging features
* Dockerization, cloud deployment options
* More robust PDF/minor file format support
* Functionality to download generated reports or code as files