**Smart SDLC - AI Enhanced Software Development Lifecycle- Project Documentation**

**Introduction**

Project Title:

Smart SDLC - AI Enhanced Software Development Lifecycle

Team Members:

1. Niveditha.S

2. Pavithra.S

3. Pooja.L

4. Pooja.S

**Contents:**

* Project overview
* Architecture
* Setup instructions
* Running the application
* API documentation
* Authentication
* User interface
* Testing
* Known issue

**Project overview**

The Smart Software Development Lifecycle (SDLC) is an

approach that leverages Artificial Intelligence (AI) to

optimize various stages of software development, from

requirement analysis to code generation. The integration

of AI models, such as Granite-3.2-2b-instruct by IBM, helps

in automating manual tasks, enhancing productivity, and

reducing human errors. The system allows software

developers to seamlessly transition between different

stages of the SDLC, enabling more effective collaboration

and faster development cycles.

Key Benefits of Smart SDLC:

Automated Requirement Analysis: AI tools automatically

parse and extract relevant requirements from documents

and descriptions.

Code Generation: Based on analyzed requirements, code

snippets are automatically generated in various

programming languages.

Increased Efficiency: Reduces the need for repetitive,

manual work, saving time and resources for developers.

Multi-Language Code Generation: The AI can generate

code in languages like Python, JavaScript, Java, C++, C#

The Smart SDLC AI Enhanced Software Development

Lifecycle project aims to revolutionize the way software is

developed by leveraging the power of artificial intelligence. This

project integrates AI-powered code analysis and

capabilities into the software development lifecycle, enabling

developers to write better code, faster. The Smart SDLC

Enhanced Software Development Lifecycle project includes

following key features: AI-powered code analysis, code

generation, and integration with Gradio for user interface. The

project utilizes the following technologies: PyTorch

Transformers, PyPDF2, and Gradio. PyTorch is a popular

learning framework for building and training machine learning

models. Transformers are a type of neural network architecture

that's particularly well-suited for natural language processing

tasks. PyPDF2 is a library for reading and writing PDF files

Gradio is a library for building user-friendly interfaces for

machine learning models

**Architecture**

The architecture of Smart SDLC follows a **modular**

**layered design**, ensuring scalability, flexibility, and

maintainability.

**User Interface Layer (Frontend)**

* **Technology:** Gradio (Python-based UI library).
* **Role:**
  + Provides an accessible web dashboard.
  + Allows users to upload PDF requirement documents

or enter free-text requirements.

* + Provides interactive tabs for requirement analysis and

code generation.

**Application Layer (Backend Services)**

* **Technology:** Python with Hugging Face Transformers.
* **Role:**
  + Manages interactions between the frontend and the

AI model.

* + Implements functions for requirement extraction,

analysis, and code generation.

* + Handles user prompts and organizes outputs into

structured formats.

**AI Processing Layer**

* **Technology:** IBM Granite 3.2 2B Instruct LLM via Hugging Face Transformers.
* **Role:**
  + Performs **Natural Language Understanding (NLU)** for

requirement classification.

* + Generates **programming code snippets** based

natural language requirements.

* + Ensures contextual relevance and accuracy.**Data**

**Processing Layer**

* **Technology:** PyPDF2.
* **Role:**
  + Extracts and preprocesses text from uploaded PDF d
  + Ensures clean, structured data is passed to the

layer for analysis.

**System Deployment Layer**

* **Technology:** Python runtime with optional GPU

acceleration (CUDA-enabled).

* **Deployment:**
  + Runs locally or on cloud-based environments.
  + Accessible via browser through Gradio’s hosted

interface (supports share=True for public access).

**Setup Instructions**

**Prerequisites:**

Python 3.9 or later (3.10+ recommended) - pip (Python

package manager) - Virtual environment tool (venv or conda) -

Git (to clone the repository) - GPU with CUDA (optional, for

faster performance) 2. Environment Setup

Step 1: Clone the

Repository git clone https://github.com/niviniv710/IBM-Project.git

Step 2: Create Virtual Environment

python -m venv venv source venv/bin/activate # Linux / macOS

venv\Scripts\activate # Windows (Or using Conda) conda create

-n smartsdlc python=3.10 -y conda activate smartsdlc

Step 3:

Install Dependencies pip install -r requirements.txt (If missing)

pip install transformers torch gradio PyPDF2

**Running the Application**

Step 1: Start the Application python sdlc1.py Step 2: Access the

Gradio Interface Running on local URL: http://127.0.0.1:7860 (If

share=True is set, a public URL will also be generated)

4. Optional: GPU Acceleration - Install GPU-enabled PyTorch:

pip install torch --index-url

https://download.pytorch.org/whl/cu121 - Verify GPU: import

torch print(torch.cuda.is\_available()) 5. Project Folder Structure

Smart-SDLC/ ■■■ sdlc1.py # Main application file ■■■

requirements.txt # Python dependencies ■■■ outputs/ #

(Optional) Generated analysis/code results ■■■ docs/ #

Documentation files 6. Troubleshooting –

ModuleNotFoundError: Reinstall dependencies.

**API Documentation**

**Overview:**

The Smart SDLC backend exposes key functions as APIs

(internally via Python) and externally via the **Gradio interface**.

These APIs allow requirement analysis, text extraction from

PDFs, and code generation.

**Functions:**

1. **generate\_response(prompt, max\_length=1024)** –

Generates text/code from input prompts.

1. **extract\_text\_from\_pdf(pdf\_file)** – Reads and extracts text

from PDF documents.

1. **requirement\_analysis(pdf\_file, prompt\_text)** – Extracts

and classifies requirements.

1. **code\_generation(prompt, language)** – Produces code

snippets in the specified language.

**Supported Code Languages:** Python, JavaScript, Java, C++,

C#, PHP, Go, Rust.

**Authentication**

Currently, the Smart SDLC prototype does **not implement**

**user-level authentication**. The application is accessible locally

or through a shared Gradio link.

**Planned Authentication Features (Future Scope):**

* **User Login System:** Secure login with username/password

or Single Sign-On (SSO).

* **Role-Based Access:** Restrict access to analysis vs. code-

generation modules.

* **API Tokens:** For secure programmatic access, especially if

exposed as a REST API.

* **Data Privacy:** Ensure uploaded PDFs and requirements are

stored securely and deleted after processing.

**User Interface**

**Technology:** Gradio Web UI

**Features:**

* **Tab 1: Requirement Analysis**
  + Inputs: Upload PDF file or type requirements.
  + Output: Structured classification (Functional, Non-

functional, Technical).

* **Tab 2: Code Generation**
  + Inputs: Requirement text + programming language

dropdown.

* + Output: Generated code snippet in the selected

language.

**Design Characteristics:**

* Simple, text-based interface suitable for technical and non-

technical users.

* Immediate response generation powered by IBM Granite

LLM.

* Supports file handling (PDF) and free-text input.

**Future UI Enhancements:**

* Dark mode for developer-friendly usage
* Save/export results directly as files.
* Dashboard for managing past analyses and code snippets.

**Testing**

**Testing Approach:**The project requires both **functional** and

**non-functional** testing to ensure reliability and performance.

**1. Unit Testing**

* **Functions Tested:**
  + extract\_text\_from\_pdf() → Ensure correct text

extraction.

* + generate\_response() → Validate model output

structure.

* + requirement\_analysis() → Check correct

classification.

* + code\_generation() → Verify valid code is produced.

**2. Integration Testing**

* Test PDF upload → requirement analysis → code

generation end-to-end workflow.

* Validate UI integration with backend functions.

**3. User Interface Testing**

* Check upload, textbox, and dropdown components.
* Ensure smooth switching between tabs.
* Validate error messages (e.g., invalid PDF).

**4. Performance Testing**

* Measure response times with/without GPU acceleration.
* Validate memory usage for large PDF inputs.

**5. Future Testing Enhancements**

* **Automated Test Suite:** Using pytest for backend

functions.

* **UI Testing:** Using Selenium or Playwright for Gradio

interface automation.

* **Load Testing:** Simulate multiple users to check system

scalability.

**Known Issues**

1. **Limited Authentication**
   * The current system does not implement user login or

role-based authentication.

* + Gradio’s share=True option generates a public link

that may expose the system to unauthorized access if

shared.

1. **Dependency Management**
   * Requires manual installation of Python libraries

(transformers, torch, gradio, PyPDF2).

* + Version conflicts may occur if users install

incompatible Torch or Transformers versions.

1. **GPU Dependency for Performance**
   * While the application runs on CPU, response times are

slower compared to GPU-enabled systems.

* + Large requirement documents or complex prompts

may take significant time on CPU.

1. **PDF Extraction Limitations**
   * PyPDF2 does not perfectly extract text from scanned

PDFs or image-based documents.

* + Poorly formatted PDFs may result in missing or

garbled requirement text.

1. **AI Model Limitations**
   * Generated requirements or code may contain

inaccuracies, especially for highly domain-specific

tasks.

* + AI sometimes produces verbose or redundant

responses.

* + Code snippets may require manual debugging before

production use.

1. **User Interface Constraints**
   * Current Gradio UI is minimal and not customizable
   * for enterprise use.
   * No built-in export (e.g., download analysis as a file).
2. **Error Handling**

Limited error messages for invalid input

(e.g., corrupted PDF, empty prompt).

* + No logging system to track application errors.

1. **Scalability Issues**
   * Application is designed for single-user/local use.
   * Multi-user load handling (concurrent sessions) not

tested or supported.

**Future Enhancements**

1. **Authentication & Security**
   * Implement a **user login system** with role-based access

(e.g., Admin, Developer, Viewer).

* + Introduce **API key/token authentication** for

programmatic access.

* + Encrypt and securely store uploaded documents;

enable auto-deletion after processing.

1. **Advanced Requirement Analysis**
   * Improve classification with **domain-specific AI**

**models** (e.g., healthcare, finance).

* + Support **natural language queries** like “List all

performance requirements.”

* + Add requirement traceability features to map

requirements to generated code.

1. **Enhanced PDF/Text Processing**
   * Support **OCR (Optical Character Recognition)** for

scanned PDFs using Tesseract or PyMuPDF.

* + Handle multi-column layouts and tables for more

accurate requirement extraction.

* + Enable direct upload of **Word documents (.docx)** in

addition to PDFs.

1. **Improved Code Generation**
   * Expand support to more programming languages and

frameworks.

* + Generate **unit test cases** alongside functional code.
  + Provide **explanations with code** to improve developer

understanding.

1. **User Interface Enhancements**
   * Introduce a **dashboard** to manage past analyses and

generated code.

* + Add export options (PDF, DOCX, Markdown) for

analysis and code outputs.

* + Provide a **dark mode** and customizable themes.

1. **Collaboration Features**
   * Multi-user support with shared workspaces.
   * Real-time collaboration on requirement analysis.
   * Commenting and versioning system for requirements

and generated code.

1. **Performance & Scalability**
   * Deploy system on **cloud platforms** (AWS, Azure,

GCP) for scalability.

* + Containerize the application using **Docker** for

portability.

* + Implement load balancing to support

concurrent users.

1. **Testing & Quality Assurance**
   * Automated test suite using pytest for backend

functions.

* + **Continuous Integration/Continuous Deployment**

**(CI/CD)** pipeline.

* + End-to-end UI testing with Selenium or Playwright.

1. **Analytics & Reporting**
   * Track requirement analysis trends (e.g., percentage of

functional vs. non-functional requirements).

* + Provide **usage reports** for teams and project

managers.AI-based quality score for generated code.