**SMART SDLC – AI –ENHANCED SOFTWARE DEVELOPMENT LIFECYCLE**

# 

# **1.INTRODUCTION**

* + **Project title : Smart SDLC – AI –Enhanced Software Development Lifecycle**
  + **Team member : PINKI B**
  + **Team member : GOWRI KUMARI P**
  + **Team member : PRASANNA E**
  + **Team member : ROOPASREE M**

# **2.PROJECT OVERVIEW**

* **Purpose :**

The purpose of **Smart SDLC** is to transform the traditional software development process into a more intelligent, efficient, and adaptive framework by integrating modern technologies such as artificial intelligence, automation, and predictive analytics. It aims to streamline every phase of development—right from requirement gathering to deployment—by automating repetitive tasks, improving accuracy, and enabling data-driven decision-making. Through continuous testing, real-time monitoring, and predictive risk analysis, Smart SDLC ensures higher software quality, faster delivery, and reduced development costs.

* **Features :**

**Conversational Interface**

***Key Point****:* Natural language interaction

***Functionality:***It enhances **collaboration and decision-making** by offering quick insights, automated suggestions, and seamless communication across teams.

**Policy Summarization**

***Key Point:*** Simplified policy understanding

***Functionality:*** It ensures **quick decision-making and adherence** to standards by highlighting key rules and guidelines without overwhelming details.

**Resource Forecasting**

***Key Point*:** Predictive analytics

***Functionality:*** It helps in **optimizing resource allocation and reducing project delays** by anticipating future needs accurately.

**Eco – Tip Generator**

***Key Point:*** Personalized sustainability advice***.***

***Functionality:*** It promotes **sustainable software engineering practices** by encouraging eco-friendly decisions at every stage of the lifecycle.

**Citizen Feedback Loop**

***Key Point:*** Community engagement

***Functionality:*** It ensures **continuous enhancement and higher user satisfaction** by integrating feedback directly into the development cycle.

**KPI Forecasting**

***Key Point:*** Strategic planning support.

***Functionality:*** It helps in **tracking progress and making proactive adjustments** to ensure project goals are consistently met.

**Anomaly Detection**

***Key Point:*** Early warning system

***Functionality:*** It enhances **system reliability and security** by detecting errors or threats early before they impact the project.

## **Multimodal Input Support**

***Key Point:*** Flexible data handling.

***Functionality:*** Multimodal input support in Smart SDLC allows **interaction through text, voice, images, and other formats** for seamless communication with development tools.

**Streamlit or Gradio UI**

***Key Point:*** User-friendly interface.

***Functionality:*** It enables **easy experimentation and collaboration** by allowing teams to interact with AI models and tools without complex coding.

**3.ARCHITECTURE**

* **Frontend (stream lit):**

In the **Smart SDLC architecture**, the **frontend built with Stream lit** acts as the interactive layer that connects users with the intelligent backend services. Stream lit provides a lightweight yet powerful framework to create **real-time dashboards, visualizations, and input forms** without requiring extensive frontend development skills.

* **Backend (Fast API):**

In the **Smart SDLC architecture**, the **backend powered by Fast API** serves as the core engine that manages communication between the frontend, AI/ML modules, and databases. Fast API is a modern, high-performance Python web framework designed for building APIs quickly and efficiently. Additionally, Fast API provides automatic documentation, scalability, and seamless integration with machine learning models, making the backend of Smart SDLC both **robust and developer-friendly.**

## **LLM Integration (IBM Watsonx Granite):**

## In the **Smart SDLC architecture, LLM integration with IBM Watsonx Granite** plays a vital role in enhancing intelligence and automation throughout the development lifecycle. By embedding Watsonx Granite, the system gains advanced **natural language processing (NLP)** and **large-scale reasoning capabilities** to support tasks like **policy summarization, citizen feedback analysis, anomaly detection, and conversational interfaces**.

* **Vector Search (Pinecone):**

In the **Smart SDLC architecture, vector search using Pinecone** plays a crucial role in handling large volumes of unstructured data like documentation, logs, policies, and user feedback. Pinecone enables efficient **semantic search and retrieval**, allowing the system to find the most relevant information based on meaning rather than just keywords. This is especially useful for modules like **policy summarization, anomaly detection, and citizen feedback analysis,** where quick access to context-rich data is essential.

* **ML Modules (Forecasting and Anomaly Detection):**

In the **Smart SDLC architecture**, the **Machine Learning (ML) modules** play a crucial role in bringing intelligence and automation into the software development process. These modules focus mainly on **Forecasting** and **Anomaly Detection**.

* **Forecasting**: This module uses predictive analytics to estimate future trends such as project timelines, resource requirements, costs, and key performance indicators (KPIs).
* **Anomaly Detection**: This module continuously monitors project activities, system performance, and code quality to detect unusual patterns, errors, or security threats.

**4.SETUP INSTRUCTONS**

**Prerequisites:**

* Python 3.9 or later installed
* pip and virtual environment tools (venv or virtualenv)
* Hugging Face Transformers
* PyTorch (with CUDA if GPU available)
* Gradio for interface

**Installation Process:**

* Clone the Repository
* Create and Activate Virtual Environment
* Install Dependencies
* Download/Configure Model
* Run the Application
* Access the Gradio Interface
* Interact with Modules

**5.FOLDER STRUCTURE**

app/ – Contains the main Gradio application logic, including response functions and interface setup.

modules/ – Subdirectory for modular Python scripts like city analysis, citizen interaction, and helper functions.

granite\_llm.py – Handles all communication with the IBM Granite model, including loading, tokenization, and text generation.

ui/ – Holds frontend assets such as CSS stylesheets, images, or future UI customizations.

citizen\_ai.py – Entry script for launching the Gradio interface with City Analysis and Citizen Services tabs.

city\_analysis.py – Provides detailed safety and accident analysis for a given city.

citizen\_services.py – Manages AI-powered responses for public service and policy-related citizen queries.

requirements.txt – Lists all dependencies (torch, transformers, gradio) for easy installation.

README.md – Documentation file explaining setup, usage, and project overview.

docs/ – Extra documentation files (architecture, API usage, and future enhancements).

**6.RUNNING THE APPLICATION**

**To start the project:**

* Run the Gradio app script:
* The Gradio server will launch automatically and display a local URL
* Open the URL in your web browser to access the Smart SDLC interface.

**Navigating the Application:**

* Requirement Analysis Tab → Enter project requirements to generate AI-assisted requirement documentation.
* Design Phase Tab → Get system design suggestions and architectural diagrams.
* Coding Phase Tab → AI generates starter code snippets or optimizes existing code.
* Testing Phase Tab → Create automated test cases, detect bugs, and suggest fixes.
* Deployment Phase Tab → AI provides deployment strategies and monitoring insights.

**Frontend (Gradio):**

* Built using Gradio Blocks and Tabs for an interactive interface.
* Organized into multiple pages (tabs) corresponding to SDLC phases.
* Enter text inputs (e.g., project requirements, code queries).
* Submit queries via buttons.
* View AI-generated responses in real-time.
* The layout uses rows and columns for clarity, ensuring a clean and user-friendly experience.

**Backend (Python + Hugging Face Transformers):**

* Implemented in Python, with Gradio also handling the web server logic.Input is forwarded to the IBM Granite model, where:
* The model is loaded and tokenized.
* AI generates structured outputs.
* Results are returned in a formatted way to the UI.
* Ensures smooth integration between frontend (UI) and backend (AI engine), delivering instant feedback.

**7.AUTHENTICATION**

The project runs in an open environment for demonstration purposes. All users can directly access the Smart SDLC interface without restrictions.For secure deployments, authentication can be integrated at multiple levels.

* simple username/password login.
* Token based authentication Implement JWT (JSON Web Tokens) or API keys for API-style programmatic access.

Ensures secure communication between clients and backend services.

* OAuth2 Integration

Connect with IBM Cloud IAM, Google, Azure AD, or other third-party identity providers.

* Role-Based Access Control (RBAC)

Define user roles such as:

1.Admin → Full system control

2.Analyst → Access to SDLC analysis and reporting tools

3.Citizen/User → Restricted interaction with public-facing modules

**8.USER INTERFACE**

The Smart SDLC AI-Enhanced Software Development Life Cycle interface is designed to be minimalist, functional, and accessible, ensuring that even non-technical users can interact with the system seamlessly.

Tabbed Layout

Organized into multiple tabs.Enables easy navigation between modules without clutter.

Textboxes with placeholder guidance to help users enter valid queries or city names.

Buttons for quick submission of queries.

Intuitive field labels for clarity.

Real-time AI responses shown in dedicated output boxes.

Built-in hints and examples in input fields.

Step-by-step feedback to reduce user confusion.

**9.TESTING**

**1. Unit Testing:**

Core functions such as generate\_response(), city\_analysis(), and citizen\_interaction() were tested individually to verify correct output generation and ensure logical accuracy.

**2. Manual Testing:**

Different scenarios were manually tested, including entering city names, submitting public service queries, and reviewing AI-generated responses for accuracy, clarity, and usability.

**3. Interface Testing:**

Gradio input and output components were checked thoroughly. Button clicks, text input handling, and smooth navigation between tabs were validated to confirm proper UI functionality.

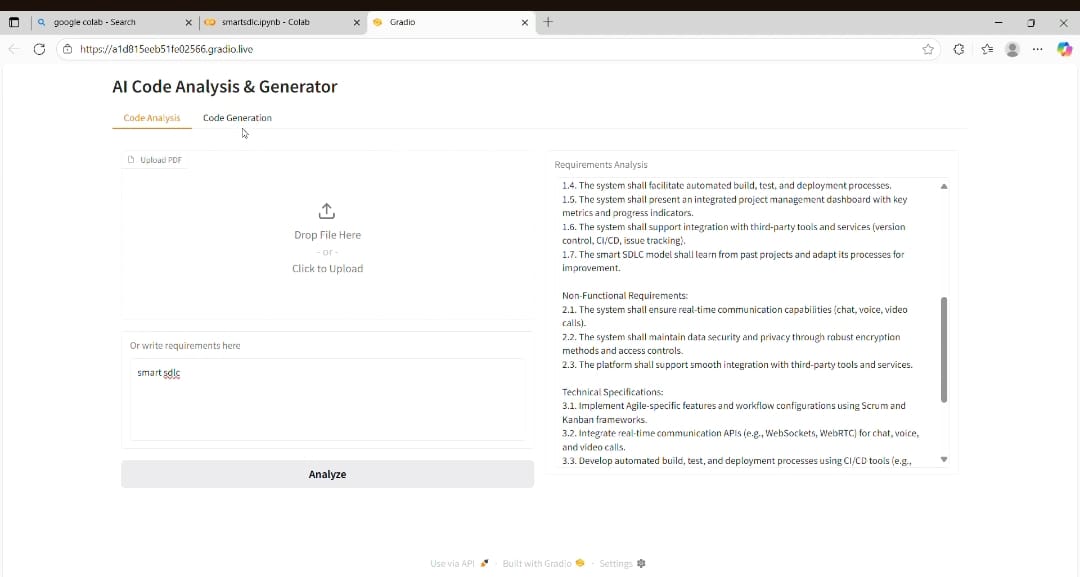
**4. Edge Case Handling:**

Special cases such as malformed queries, empty inputs, unusually long text, and unexpected characters were tested to confirm the system handles errors gracefully without breaking the interface.

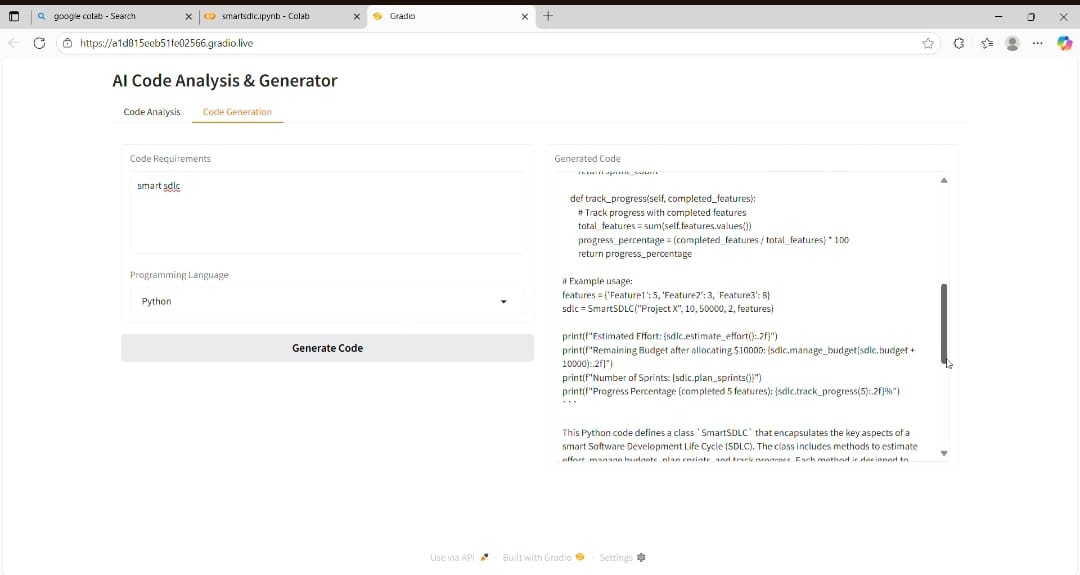
All functions and interactions were validated to ensure reliability, responsiveness, and consistency in real-time usage, providing a stable and smooth experience for end users.

**10.SCREEN SHOTS**

* **CODE ANALYSIS**

****

* **CODE GENERATION**

****

**11.KNOWN ISSUES**

* Performance Limitations: Running large models like IBM Granite on CPU can result in slower response times compared to GPU-accelerated environments.
* Model Dependency: The system depends on internet access to download and load Hugging Face models during setup.
* Basic Error Handling: While edge cases such as empty inputs and malformed queries are handled.
* Authentication Gaps: Current implementation runs in an open demo environment.

**12.FUTURE ENHANCEMENT**

Add user authentication and role-based access for secure usage.

Implement persistent chat history and session tracking.

Integrate vector search with FAISS or Pinecone for fact-grounded responses.

Improve UI/UX, including dashboards, summary cards, and visual analytics.

Add file upload support for documents or datasets to enhance city analysis.

Optimize model inference speed for large inputs.

Include analytics modules like sentiment analysis and citizen query categorization.