SmartSDLC - AI-Powered Software Development Lifecycle Platform

Project Description

SmartSDLC is a full-stack AI-powered platform that automates key stages of the Software Development Lifecycle using IBM Watsonx, FastAPI, and Streamlit. It transforms unstructured PDF requirements into structured user stories, generates production-ready code, and creates test cases automatically.
The platform includes modules for bug fixing, code summarization, and real-time chatbot support for SDLC guidance.
Its modular design and intuitive UI enable both technical and non-technical users to interact seamlessly with AI-driven development tools.
SmartSDLC accelerates development, enhances accuracy, and reduces manual effort across the entire SDLC process.

☑ Use Case Scenarios

Scenario 1: Requirement Upload and Classification

➞ User Steps:

- **Uploads a PDF containing unstructured requirements via the SmartSDLC interface.**
- **The backend extracts text using PyMuPDF and sends it to Watsonx Granite-20B for classification.**
- **Results are organized into SDLC phases and converted into structured user stories.**

Outcome:
Users receive clearly categorized user stories grouped by SDLC phase, streamlining project planning and traceability.

Scenario 2: AI Code Generator

➞ User Steps:

- **Inputs a natural language prompt or user story in the development module.**
- **Prompt is forwarded to Watsonx Granite-20B-Code-Instruct to generate relevant code.**
- **The frontend displays syntax-highlighted, production-ready code.**

Outcome:
Users obtain clean, executable code instantly, reducing development time and effort.

Scenario 3: Bug Fixer

➞ **User Steps:**

- **Pastes buggy Python or JavaScript code into the Bug Fixer interface.**
- **The code is analyzed and corrected by Watsonx for syntax and logic errors.**
- **Corrected code is shown alongside the original for comparison.**

Outcome:
Users quickly receive optimized code, minimizing manual debugging and enhancing code reliability.

Scenario 4: Test Case Generator

➞ **User Steps:**

- **Submits functional code or a user requirement.**
- **Watsonx generates test cases using standard frameworks like unittest or pytest.**
- **Test cases are presented for review or integration.**

Outcome:
Automated test case generation ensures consistency and accelerates quality assurance processes.

Scenario 5: Code Summarizer

➞ **User Steps:**

- **Enters source code into the Code Summarizer module.**
- **Watsonx processes the logic and returns a plain-language summary.**
- **Summary is displayed in an easily readable format.**

Outcome:
Users gain quick understanding of complex code, aiding onboarding and documentation.

Scenario 6: Floating AI Chatbot Assistant

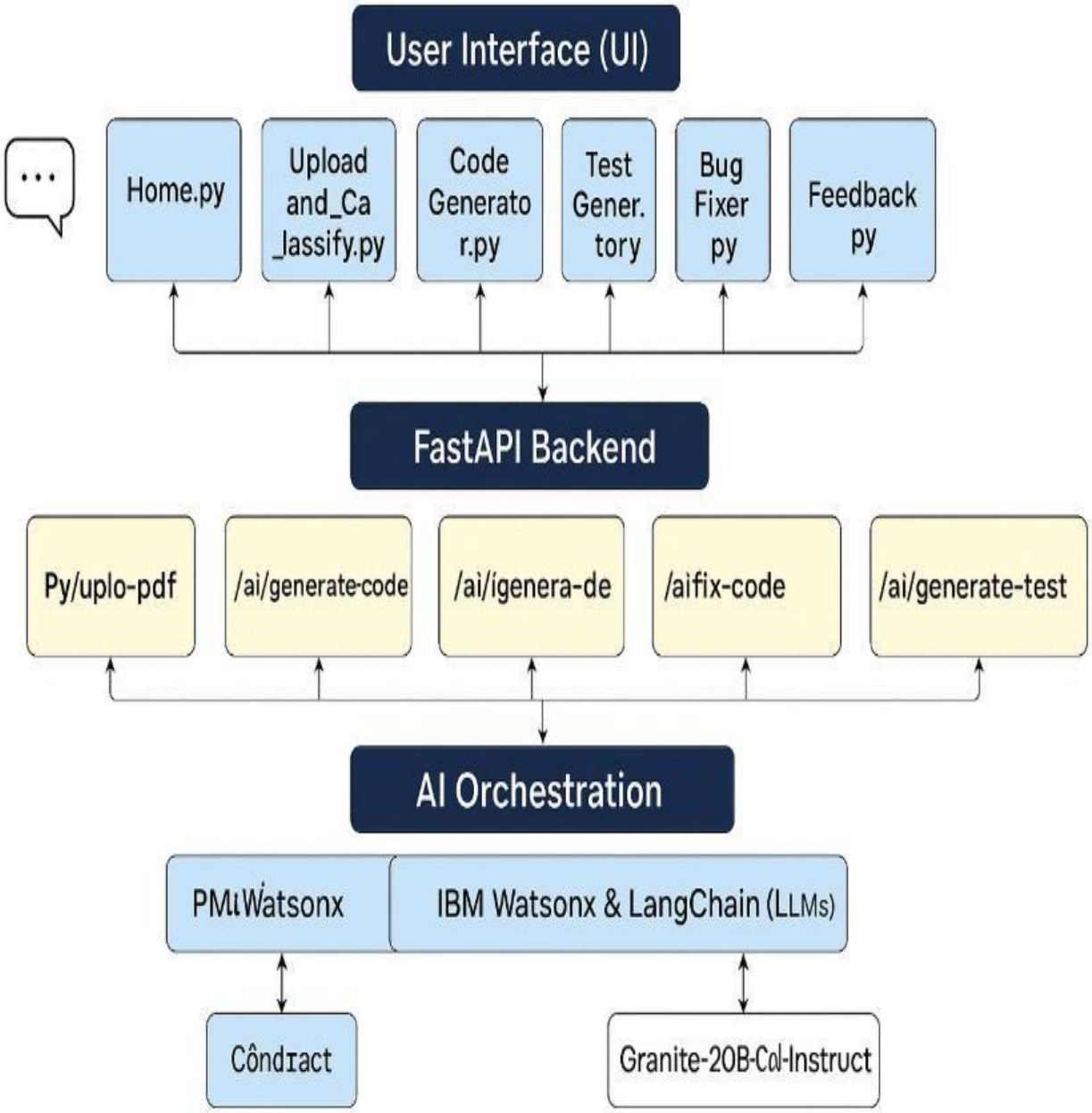
➞ **User Steps:**

- **Opens the integrated chatbot from the homepage.**
- **Submits SDLC-related queries like “How do I write a unit test?”.**
- **Receives intelligent, contextual answers powered by LangChain and Watsonx.**

Outcome:

Users benefit from real-time guidance and support, enhancing learning and platform usability.

Technical Architecture:



Activity: Designing and Developing the User Interface

Activity 1: Set Up the Base Streamlit Structure

➞ Add a hero section with:

- **Lottie animation for visual appeal.**
- **Platform title and tagline for branding.**

- **Organize feature links in a grid layout using st.columns() for a clean user experience.**

Activity 2: Design a Responsive Layout

➞ Use layout components:

- **st.columns() for multi-column design.**
- **st.container() to group elements.** o **st.markdown() for HTML/CSS styling.**

➞ Apply custom CSS to enhance UI with better fonts, background colors, and shadow effects.

➞ Ensure design is mobile and desktop responsive.

Activity 3: Build Modular Feature Pages

➞ Create individual feature modules inside the pages/ directory for easy navigation and separation of concerns:

- **Test_Generator.py: Generate relevant test cases.**
- **Bug_Fixer.py: Detect and fix code issues.**
- **Code_Summarizer.py: Summarize code into documentation.**

Milestones and Development Phases

Milestone 1: Model Selection and Architecture

Objective:

Establish the technical backbone of SmartSDLC with AI model integration and system design.

Key Actions:

- **Research and select IBM Watsonx Granite models for:**
    - **Natural Language Processing (NLP)** o

      Code generation
- **Define complete system architecture:**
    - **Frontend: Streamlit** o **Backend:**

      FastAPI
- **Integrate AI workflow using:** o **Watsonx APIs** o **LangChain for orchestration**
- **Set up project structure and Python virtual environments**

Sample Code: Configuring Watsonx

```
# config.py  import os
from dotenv import load_dotenv
load_dotenv()
WATSONX_API_KEY = os.getenv("WATSONX_API_KEY")  WATSONX_MODEL_ID =
os.getenv("WATSONX_MODEL_ID")
```

Milestone 2: Core Functionalities Development Objective:

Build intelligent modules covering the core SDLC operations. Features to Implement:

- **Requirement Analysis**
- **Code Generation**

- **Test Case Generation**
- **Bug Fixing**
- **Documentation Generation**
- **Chatbot Assistance**
- **Feedback Collection**
- **GitHub Integration**

Backend Components:

- **FastAPI backend with: o Routing o Authentication o Modular service layers Sample Code: Bug Fixer Module**

```
# bug_resolver.py def fix_code(buggy_code:
str) -> str:
    prompt = f"Fix the following code:\n{buggy_code}"
return generate_response(prompt)
```

Milestone 3: User Interface Design and Integration

Objective:

Create an intuitive and visually clean frontend interface using Streamlit. Tasks:

- **Develop modular pages in Streamlit for each functionality**
- **Apply responsive design with custom CSS**
- **Embed a floating chatbot for assistance**
- **Implement real-time communication with backend via requests**

Sample Code: Frontend Integration

```
# Home.py

import streamlit as st

st.title("SmartSDLC") st.markdown("Automate your SDLC
with AI")

col1, col2 = st.columns(2) with col1:  st.page_link("pages/Code_Generator.py", label="Code
Generator")

# Embedding chatbot

if user_input:

reply = get_chatbot_response(user_input)

st.chat_message("assistant").write(reply)
```

Milestone 4: Deployment and Local Testing Objective:

- Create and activate Python virtual environment
- Install dependencies from requirements.txt
- Launch backend (FastAPI + Uvicorn) and frontend (Streamlit)
- Perform end-to-end testing of all modules

Sample Code & Commands

```
# Start backend uvicorn app.main:app --reload

# Start frontend

streamlit run frontend/Home.py

WATSONX_API_KEY=your_api_key

WATSONX_MODEL_ID=granite-20b-code-instruct
```

Conclusion:

SmartSDLC is a full-stack, AI-powered platform that automates key stages of the Software Development Lifecycle using IBM Watsonx and LangChain. It transforms unstructured inputs like PDFs and prompts into structured user stories, clean code, test cases, bug fixes, and documentation—significantly reducing manual effort and accelerating development.

With a modular FastAPI backend and an intuitive Streamlit frontend, SmartSDLC delivers features such as:

- **AI-Powered Requirement Classification**
- **Code Generation & Auto Bug Fixing**
- **Test Case & Documentation Generation**
- **Interactive Chatbot for SDLC Assistance**

The platform is deployable locally and provides real-time API interactions, making it practical for both developers and project teams. Its design sets the stage for future enhancements like CI/CD integration and cloud deployment.

SmartSDLC streamlines modern software development, making it faster, smarter, and more accessible.