SmartSDLC – AI-Enhanced Software Development Lifecycle

Project Documentation

1.Introduction

The SmartSDLC project is an AI-enhanced platform designed to automate and streamline the Software Development Lifecycle (SDLC) using advanced technologies like IBM Watsonx, FastAPI, LangChain, and Streamlit. It integrates generative AI to handle key SDLC phases, including requirement analysis, code generation, test case creation, bug fixing, and documentation. The platform features a user-friendly interface that allows users to upload PDFs, generate structured requirements, and transform natural language prompts into functional code. It also includes an AI-powered chatbot for real-time assistance and support. The backend, built with FastAPI, efficiently processes API requests, while the frontend,developed using Streamlit, offers a visually appealing and interactive dashboard. The system is modular, scalable, and secure, featuring a robust authentication mechanism and seamless integration between AI models and user inputs.

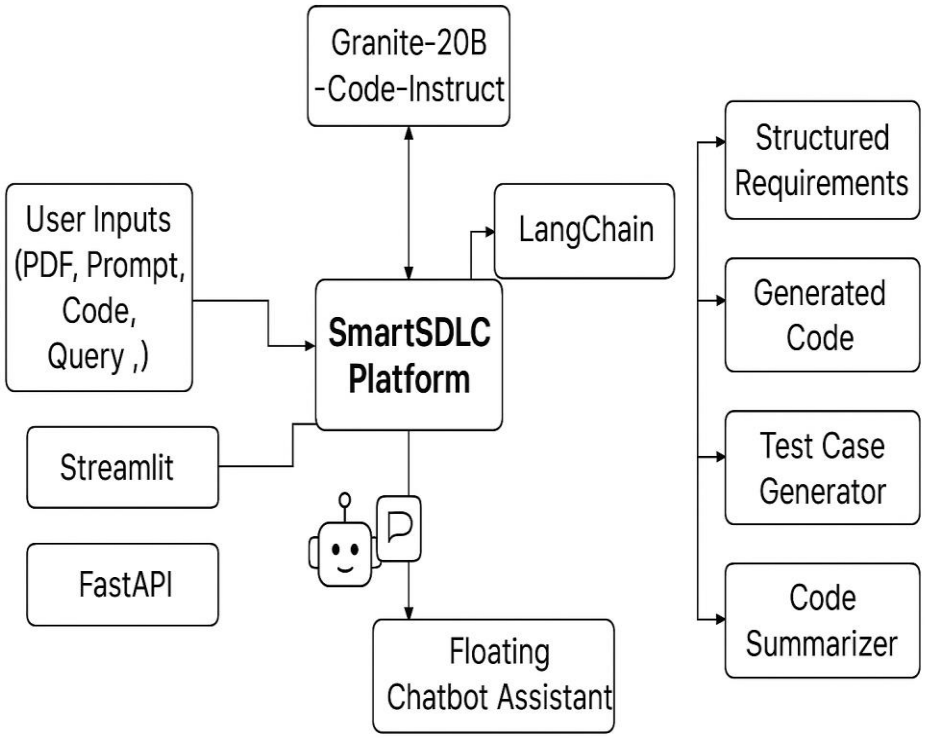
- **Project title : SmartSDLC – AI-Enhanced Software Development Lifecycle**
- **Team member : V KOWSALYA**

- **Team member : R AKASH**
- **Team member : K ASHWIN RAJ**
- **Team member : M BARANIKUMAR**

2.project overview

Requirement Upload and Classification, the platform simplifies the complex task ofrequirement gathering by allowing users to upload PDF documents containing raw, unstructured text. The backend extracts content using PyMuPDF and leverages IBM Watsonx's Granite-20B AI model to classify each sentence into specific SDLC phases such as Requirements, Design, Development, Testing, or Deployment. These classified inputs are then transformed into structured user stories, enabling clear planning and traceability. The frontend displays this output in an organized, readable format grouped by phase, significantly improving clarity and saving manual effort.

Architecture:-



Pre-requisites:

- Python 3.10 : https://www.python.org/downloads/release/python-3100/
- FastAPI : https://fastapi.tiangolo.com/
- Streamlit : https://docs.streamlit.io/
- IBM Watsonx AI & GraniteModels:https://www.ibm.com/products/watsonxai/foundation-model
- LangChain :https://www.langchain.com/

- Uvicorn: https://www.uvicorn.org/
- PyMuPDF (fitz): https://pymupdf.readthedocs.io/en/latest/
- Git & GitHub: https://www.w3schools.com/git/git_intro.asp?remote=github
- Frontend Libraries

Milestone 1: Model Selection and Architecture

Milestone 1 focuses on selecting the optimal AI models for SmartSDLC, choosing granite-13b-chat-v1 for natural language tasks and granite-20b-code-instruct for code generation. The architecture integrates the backend (FastAPI), frontend (Streamlit), AI layer (Watsonx), and modules (LangChain, GitHub). The development environment is set up with necessary dependencies, API keys, and a modular structure, enabling efficient integration and development in subsequent phases.

Activity 1: Research and select the appropriate generative AI model

EXAMPLE:-

When selecting a generative AI model, you must first define the specific task, such as generating text, images, or code. The best model for the job depends on balancing your use case, data needs, resource constraints, and the level of control required.

How to select the right model

1. Define your use case

Start by clearly defining the specific problem you need to solve or the content you need to create.

The type of content will narrow down your model options significantly.

- **Text generation:** For writing articles, summarization, or chatbots, use a Large Language Model (LLM) based on a transformer architecture.
- **Image generation:** For creating realistic or stylistic images from text, a diffusion or GAN-based model is appropriate.
- **Code generation:** For writing, debugging, or completing code, use a transformer model specifically trained on code.
- **Video generation:** For animating still images or creating short video clips from text, a diffusion-based model is suitable.

2. Assess data availability

The amount of data you have available for customization or fine-tuning will influence your choice.

- **Limited data:** If you have a small, proprietary dataset, consider fine-tuning a pre-trained, off-the-shelf model. This is more cost-effective and faster than training a new model from scratch.
- **Large, custom data:** If you have a significant amount of your own labeled, domain-specific data, building a custom model may offer higher accuracy and better specialization, though it is more resource-intensive.

3. Evaluate performance and resources

Consider the trade-offs between a model's performance and the resources it requires.

- **racSpeed vs. accuy:** A smaller model might be faster and cheaper to run, which is ideal for real-time applications, but may be less accurate than a larger, more complex model.
- **Cloud vs. on-device:** For applications that need low latency and work offline, like those on mobile phones, a lightweight, on-device model such as Google's Gemini Nano is a good fit. For more powerful tasks, a larger model run on cloud infrastructure is necessary.
- **Cost:** Compare the total cost of ownership, including training, inference, maintenance, and licensing.

4. Address governance and control

Your industry and use case may have strict requirements around data privacy, security, and explainability.

- **High transparency (open-source):** Open-source models offer greater control and transparency, allowing modification and fine-tuning of the code. This is a good option when you need to understand how the model makes decisions.
- **Security (proprietary):** Proprietary, or "closed," models are managed by a third party and often include robust, built-in security features and legal indemnification for outputs.

Milestone 2: Core Functionalities Development

Milestone 2 involves building the core AI-driven features of SmartSDLC, including requirement analysis, code generation, test case creation, bug fixing, code summarization, chatbot assistance, and GitHub integration. These functionalities are implemented using Watsonx and LangChain via modular service scripts. The FastAPI backend handles routing, authentication, and smooth API interactions, connecting frontend inputs with AI processing and generating structured outputs.

Milestone 3: main.py Development

Milestone 3 focuses on creating and organizing the main control center of the application the main.py file, which powers the Fast API backend in SmartSDLC. This milestone involves linking each core SDLC functionality through clearly defined API routes, ensuring input/output handling is reliable, and preparing the system for frontend interaction. This modular setup will allow seamless integration of Watsonx and LangChain responses while maintaining a clean API surface for the frontend.

Activity 3: Writing the Main Application Logic in main.py

1: Define the Core Routes in main.py

- Import and include routers for each of the SmartSDLC's functional areas: requirement analysis, code generation, testing, summarization, bug fixing, authentication, chatbot, and feedback.
- Use FastAPI's include_router() method to modularize the route

definitions and separate concerns across files.

🕐 **Example router mounts:**

- /ai: Handles AI-based endpoints like code generation, test creation, bug fixing.
- /auth: Manages login, registration, and user validation.
- /chat: Powers the floating chatbot via LangChain.
- /feedback: Handles submission and storage of feedback.

2: Set Up Route Handling and Middleware

FastAPI middleware is configured using CORSMiddleware to ensure the fronter (e.g., Streamlit) can interact with the backend without cross-origin issues. This is especially useful during development and should be fine-tuned in production.

```
# CORS configuration (adjust in production)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # Replace with frontend domain like ["http://localhost:3000"]
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

3: Implement Application Metadata and Root Route

The FastAPI app is created with custom metadata such as a project title, version, and description to help identify the API on interactive Swagger docs (/docs).

```
# Local imports
from app.core.config import settings
from app.db.init_db import init_db
from app.routes import  chat_routes, ai_routes, auth_routes  , feedback_routes

app = FastAPI(
    title="SmartSDLC - AI-Enhanced Software Development Lifecycle",
    description="AI-powered microservice platform for accelerating SDLC using Watsonx, FastAPI, and GitHub in
    version="1.0.0"
)
```

Milestone 4: Frontend Development

Milestone 4 focuses on creating a visually appealing and user-friendly interface for SmartSDLC using Streamlit. The UI is designed to provide smooth navigation across SDLC functionalities like requirement classification, code generation, bug fixing, and more. This milestone ensures a responsive, consistent experience while leveraging AI outputs effectively, and includes a built-in chatbot for dynamic user interaction.

Activity 4: Designing and Developing the User Interface

Step 1: Set up the project structure

First, create a new project directory and set up the file structure, which should look like this:

The directory is a special folder that Streamlit automatically recognizes to create a

multi-page app and add navigation to the sidebar.

Step 2: Install necessary libraries

Open your terminal or command prompt, navigate to your project

directory, and install the required Python packages:

pip install streamlit streamlit-lottie requests

Step 3: Create the Home.py entry point

This file will contain the welcome hero section. It loads a Lottie animation and arranges the

feature links in a grid layout using st.columns.

EXAMPLE:-

Code for Home.py

```
python

import json

import requests

import streamlit as st

from streamlit_lottie import st_lottie

st.set_page_config(

page_title="My Streamlit App",

page_icon="",

layout="wide"

)

def load_lottieurl(url: str):

r = requests.get(url)
```

```
if r.status_code != 200:

return None

return r.json()

lottie_home =

load_lottieurl("https://lottiefiles.com/animations/data-analysis-G0N6hT4qC4.json")

with st.container():

st.header("Welcome to My Streamlit Dashboard")

st.title("A Powerful Application for Your Data")

st.write("Navigate the app using the sidebar to explore different features.")

st.write("---")

hero_col, lottie_col = st.columns([2, 1])

with hero_col:

st.subheader("Interactive Features at Your Fingertips")

st.write("""

Our application provides a clean and intuitive interface for analyzing your data.

Features are modular and easy to navigate. Get started by clicking the links below!

""")
```

with lottie_col:

st_lottie(lottie_home, height=200, key="home_animation")

st.write("---")

st.header("App Features")

st.write("Click on a feature to get started.")

col1, col2, col3 = st.columns(3)

with col1:

st.subheader("🏠Dashboard")

st.write("An overview of key metrics and visualizations.")

st.page_link("pages/1_Dashboard.py", label="Go to Dashboard", icon="🏠")

with col2:

st.subheader("📄Report")

st.write("Generate detailed reports with custom parameters.")

st.page_link("pages/2_Report.py", label="Go to Report", icon="📄")

with col3:

st.subheader("📞Contact")

st.write("Get in touch with our team for support.")

st.page_link("pages/3_Contact.py", label="Go to Contact", icon="📞")

Step 4: Create the modular pages

Now, create the pages that will appear in the sidebar and be accessible from the main page links.

Code for Dashboard.py

python

```
import streamlit as st

st.set_page_config(

page_title="Dashboard",

page_icon="",

layout="wide"

)

st.title(" My Interactive Dashboard")

st.write("This page will contain your main data visualizations and analytics.")

# Add placeholder content

st.info("Dashboard content goes here.")
```

Code for _Report.py

python

```
import streamlit as st

st.set_page_config(

page_title="Report",
```

page_icon="📊",

layout="wide"

)

st.title(" Report Generation")

st.write("This page is for generating and viewing custom reports.")

Add placeholder content

st.info("Report generation interface goes here.")

Code Contact.py

python

import streamlit as st

st.set_page_config(

page_title="Contact",

page_icon="📞",

layout="wide"

)

st.title(" Contact Us")

st.write("Please fill out the form below to get in touch.")

st.info("Contact form and details go here.")

Step 5: Run the Streamlit app From your project's root directory, execute the following command in your terminal to start the

application:

2: Design a Responsive Layout Using Streamlit Components

- Use st.columns(), st.container(), st.markdown() for layout control and consistency.
- Apply custom CSS styling for better fonts, backgrounds, and card shadows.
- Design a flexible layout that works well across different screen sizes and resolutions.

3: Create Separate Pages for Each Core Functionality Build Upload_and_Classify.py: Upload PDF and classify requirements. **Code_Generator.py:** Convert user prompts into working code. Test_Generato**feature-specific modules under the pages/ directory: r.py:** Generate test cases.

Bug_Fixer.py: Automatically fix buggy code. Code_Summarizer.py: Summarize code into documentation.

Feedback.py: Collect user feedback. Connect each page to corresponding FastAPI endpoints via api_client.py.

Milestone 5: Deployment

In Milestone 5, the focus is on deploying the SmartSDLC application locally using FastAPI for the backend and Streamlit for the frontend. This involves setting up a secure environment, installing dependencies, connecting API keys for Watsonx, and launching the services to simulate a real-world workflow. This milestone helps ensure that the app can run smoothly in a local setup before shifting to cloud platforms or CI/CD

pipelines.

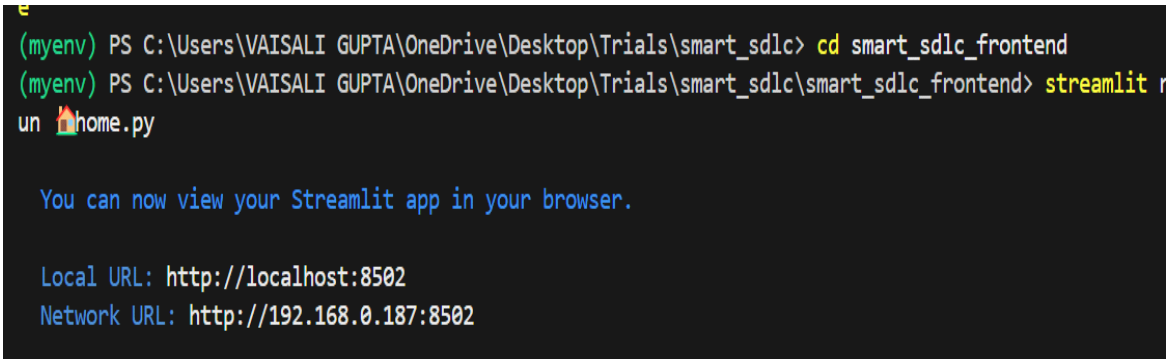
Activity 5: Testing and Verifying Local Deployment

1: Launch and Access the Application Locally

- Start the FastAPI backend using Uvicorn: uvicorn app.main:app --reload
- Run thfrontende Streamlit : streamlit run frontend/Home.py

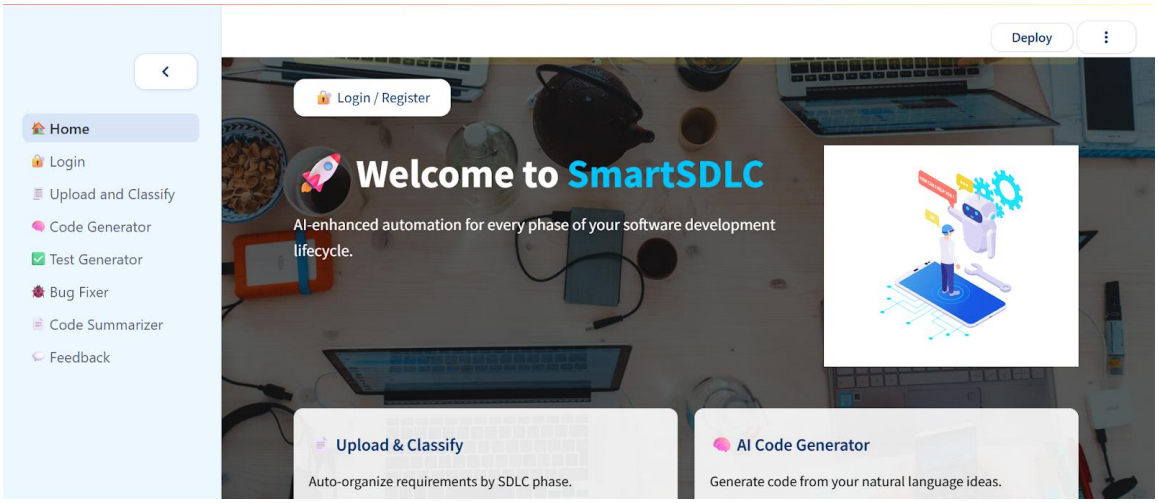
Open your browser and navigate to:

- **Streamlit UI:** http://localhost:8502
- **FastAPI Swagger Docs:** http://127.0.0.1:8000/docs Test each SmartSDLC feature (requirement upload, code generation, bug fixing, chatbot, feedback) to ensure everything is connected and functioning correctly.
- **Run the Web Application**
- Now type —streamlit run ????home.py‖ command
- Navigate to the localhost where you can view your web page

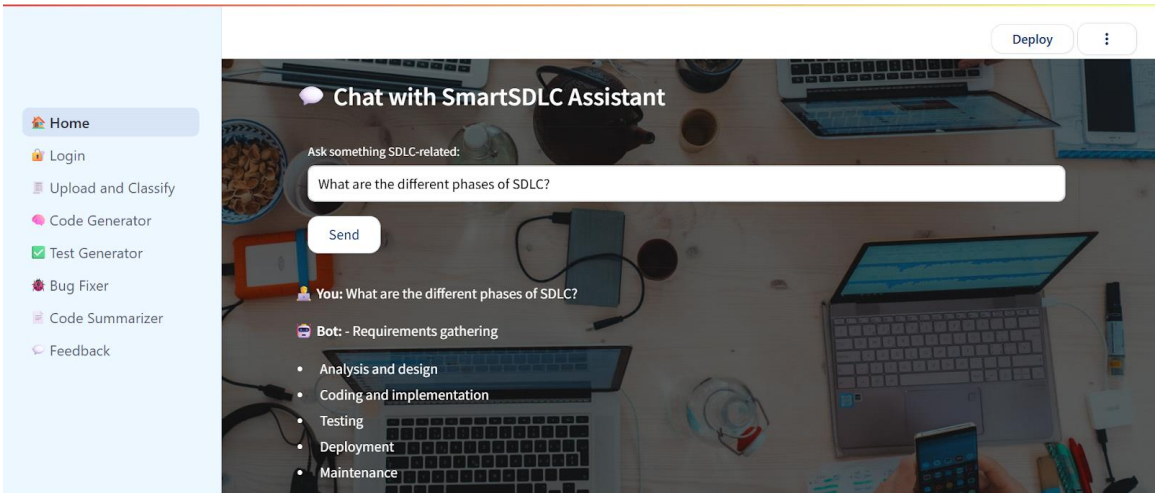


Milestone 6:

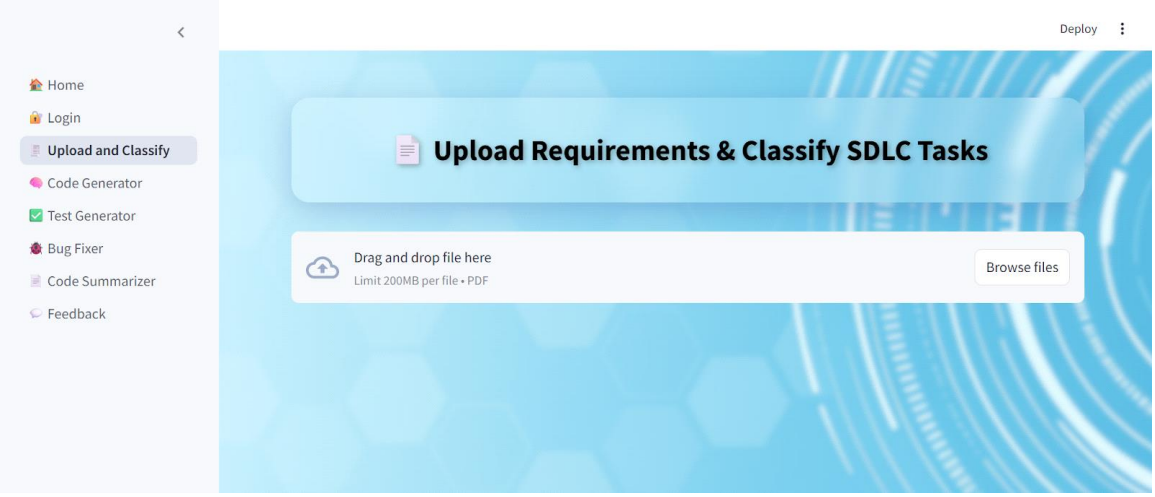
The web application will open in the web browser:



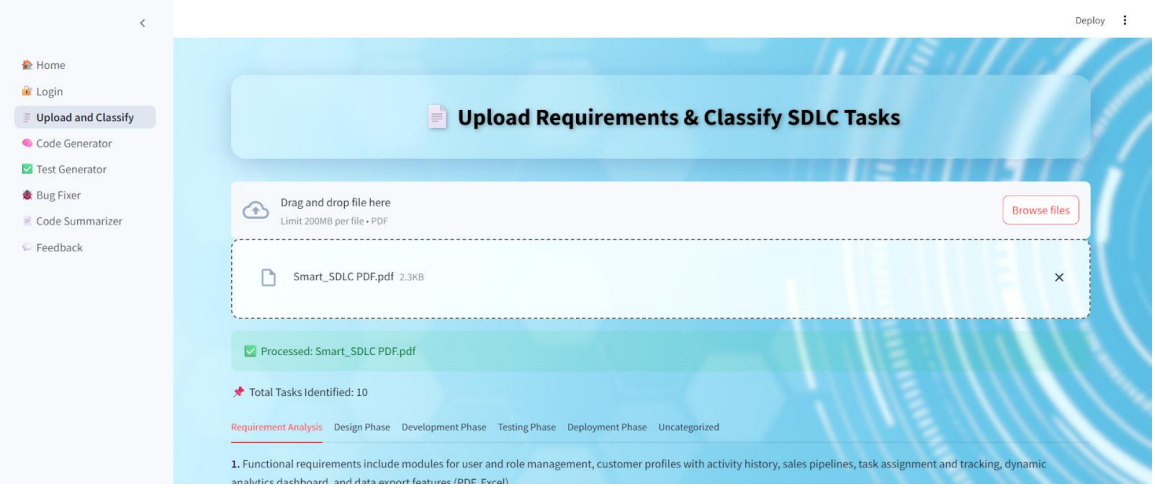
"SmartSDLC Dashboard – Your AI-powered command center for accelerating every phase of the software development lifecycle."



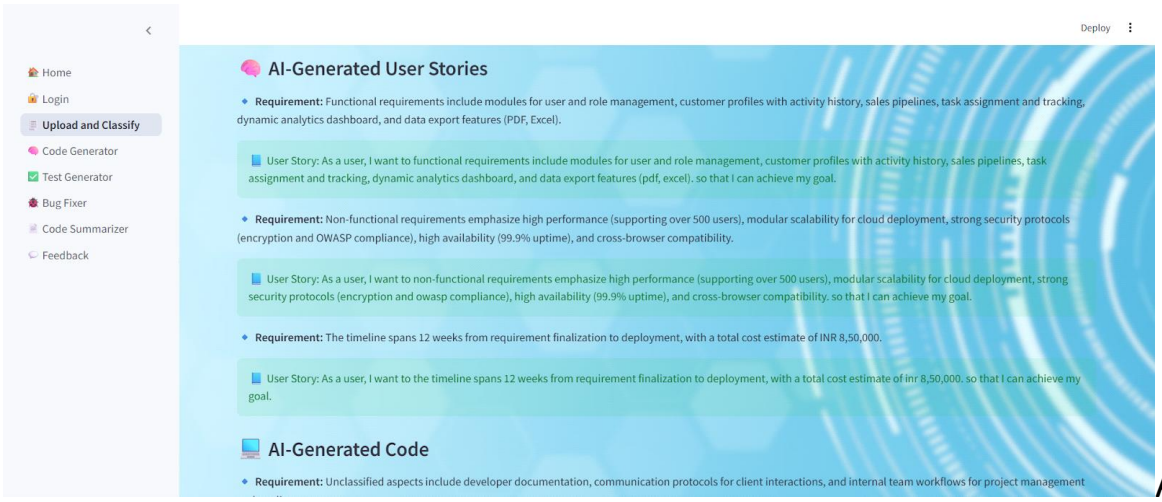
Interactive Chat with SmartSDLC Assistant – Instantly learn SDLC concepts through AI-powered conversations.



Input: "Upload and Classify – Effortlessly convert raw PDF requirements into structured SDLC tasks using AI."

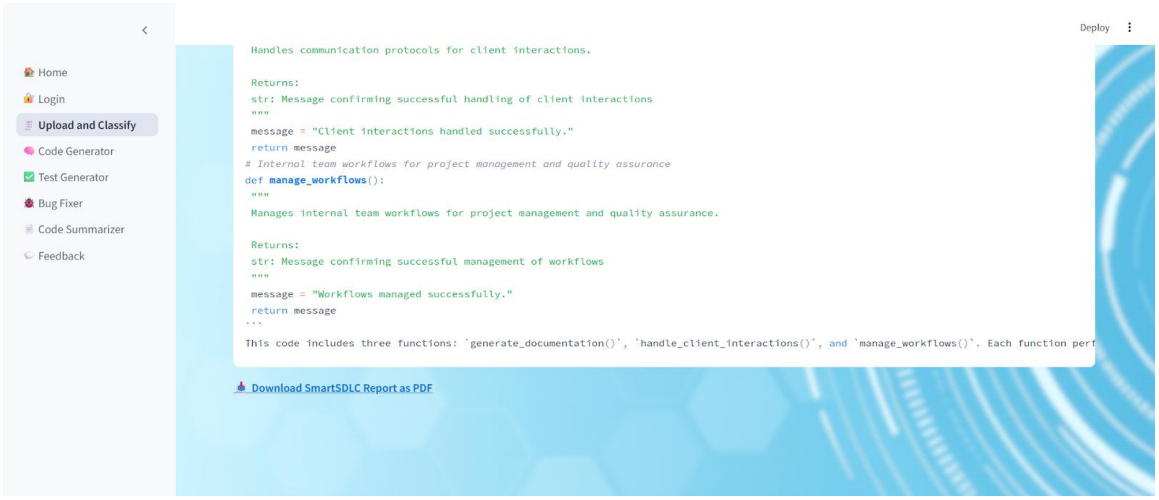


Output: "Smart Requirement Classifier – Instantly extract and categorize software tasks into SDLC phases from your uploaded PDF."

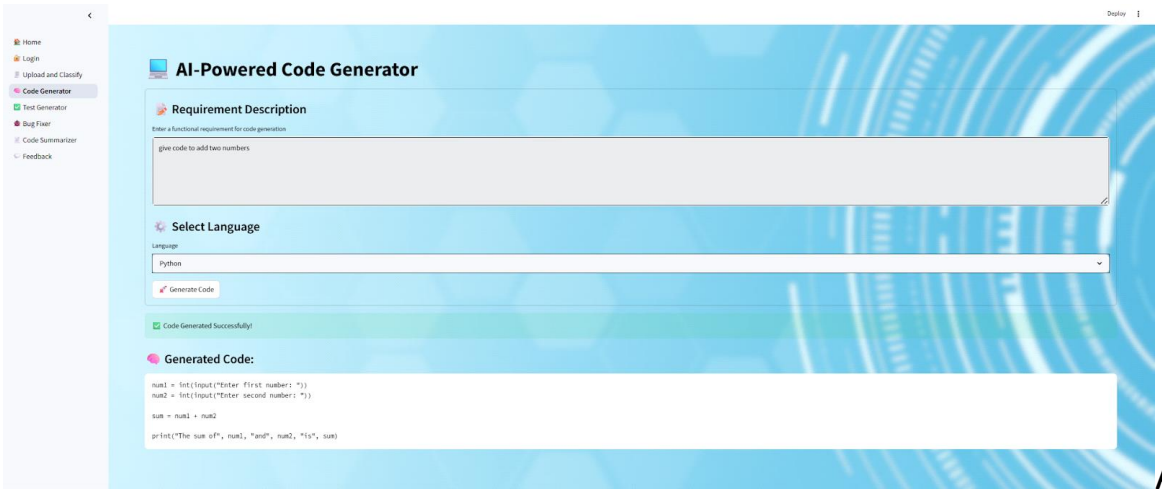


AI-Generated User Stories – Transform raw requirements into structured user-centric stories for agile development."

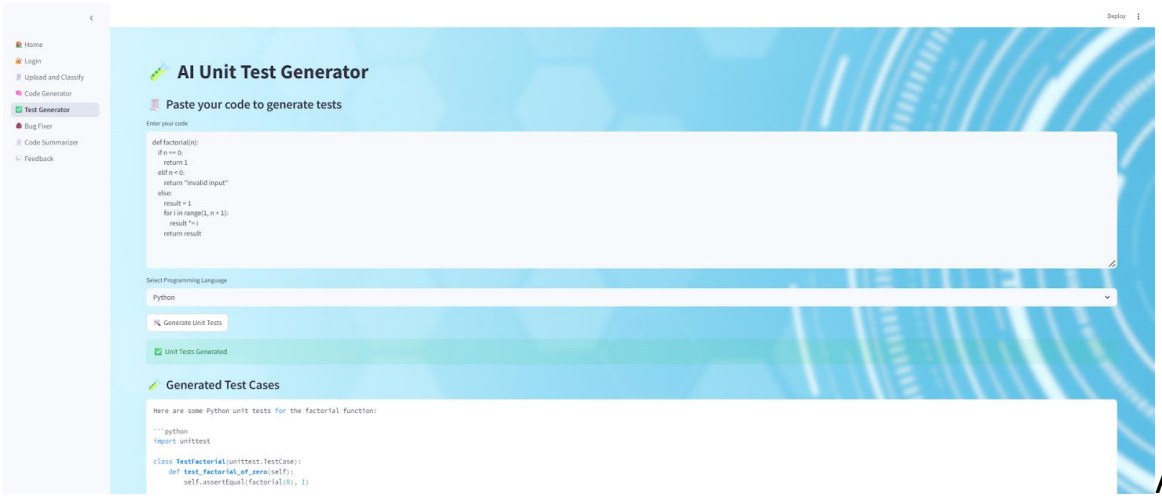
Auto-Generated Summary with Downloadable Insights – Review AI-explained code and export the SmartSDLC report as a professional PDF."



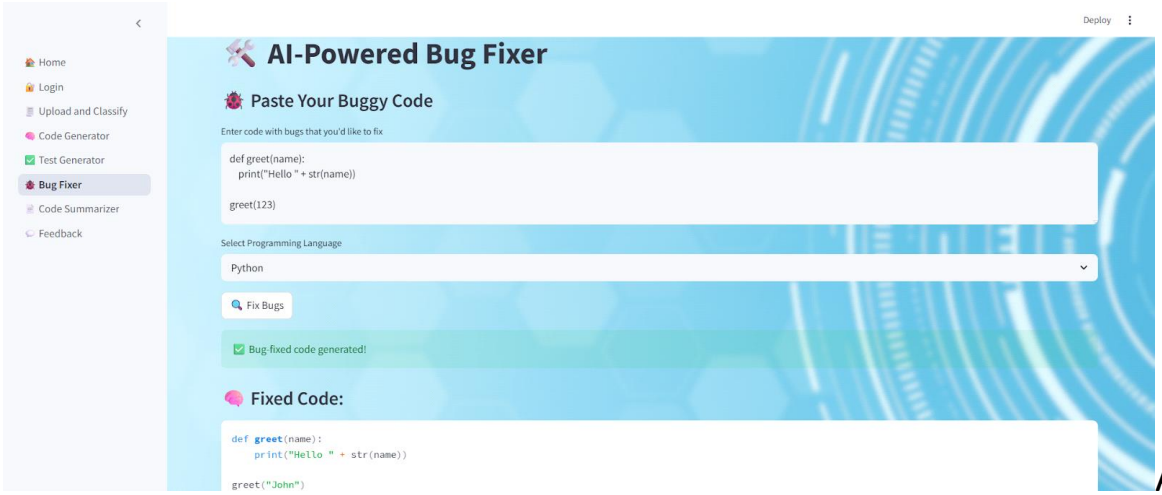
Auto-Generated Summary with Downloadable Insights – Review AI-explained code and export the SmartSDLC report as a professional PDF."



AI Code Generator – Instantly convert natural language requirements into clean, executable code with just one click."

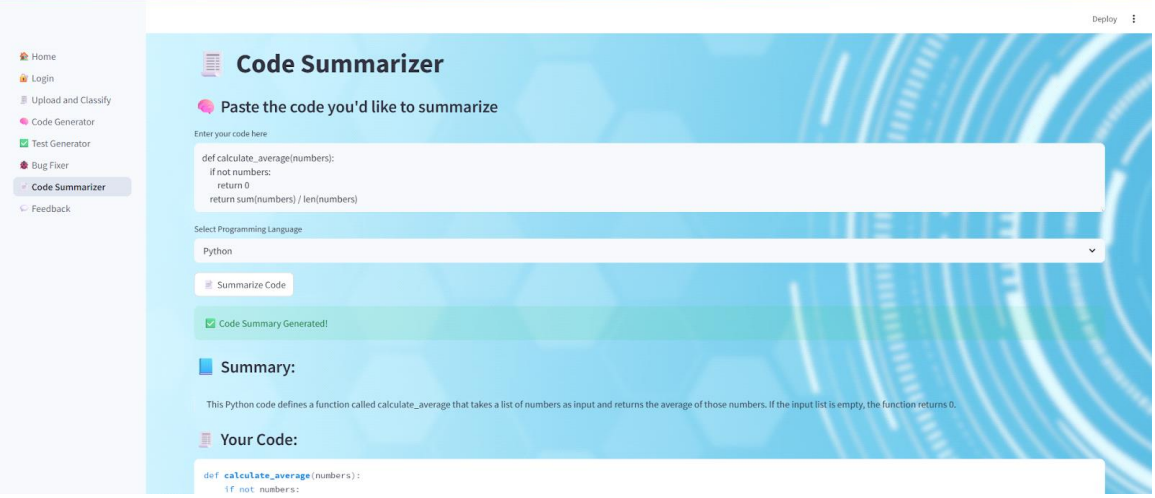


AI-powered tool to auto-generate unit tests from user-provided code

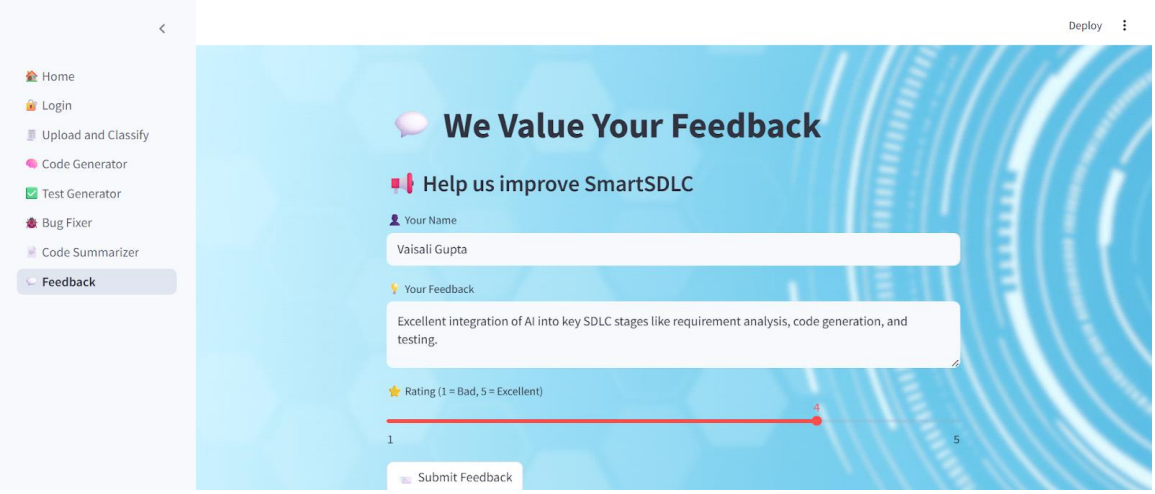


AI-

Powered Bug Fixer – Instantly detect and correct code issues for cleaner, error-free execution."



“AI-powered Code Summarizer interface displaying a generated summary for the provided code snippet.”



"User submitting feedback on the SmartSDLC system, highlighting effective AI integration across SDLC phases"

Conclusion:-

The SmartSDLC platform represents a significant advancement in the automation of the Software Development Lifecycle by integrating AI-powered intelligence into each phase—from requirement analysis to

code generation, testing, bug fixing, and documentation. By leveraging cutting-edge technologies like IBM Watsonx, FastAPI, LangChain, and Streamlit, the system demonstrates how generative AI can streamline traditional software engineering tasks, reduce manual errors, and accelerate development timelines.

The platform's modular architecture and intuitive interface empower both technical and non-technical users to interact with SDLC tasks efficiently. Features such as requirement classification from PDFs, AI-generated user stories, code generation from natural language, auto test case generation, smart bug fixing, and integrated chat assistance illustrate the power of AI when applied thoughtfully within a development framework.Overall, SmartSDLC not only improves productivity and accuracy but also the foundation for future enhancements like CI/CD integration, team collaboration, version control, and cloud deployment. It is a step toward building intelligent, developer-friendly ecosystems that support modern agile development needs with smart automation at its core.