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

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. Deployment is streamlined with local hosting via **Uvicorn** and **Streamlit**, with comprehensive API documentation available through **Swagger UI**. By leveraging AI in a practical and efficient manner, SmartSDLC significantly reduces manual workload, accelerates development processes, and enhances software quality, making it an invaluable tool for developers and project teams.

**Scenario 1: Requirement Upload and Classification**

**Requirement Upload and Classification**, the platform simplifies the complex task of requirement 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.

**Scenario 2: AI Code Generator**

**AI Code Generator**, addresses the development phase, where developers can input natural language prompts or structured user stories. These prompts are sent to the Watsonx model, which generates contextually relevant, production-ready code. This reduces the time needed for boilerplate or prototype creation and enhances coding efficiency. The code is presented in a clean, syntax-highlighted format on the frontend, ready for use or further enhancement.

**Scenario 3: Bug Fixer**

**Bug Fixer**, the platform supports debugging by accepting code snippets in languages such as Python or JavaScript. Upon receiving the buggy code, the Watsonx AI analyzes it for both syntactical and logical errors and returns an optimized version. This not only assists developers in identifying mistakes without extensive manual reviews but also provides immediate, corrected code directly in the frontend for comparison.

**Architecture**



**Pre-requisites**

* **Python 3.10 :** [**https://www.python.org/downloads/release/python-3100/**](https://www.python.org/downloads/release/python-3100/)
* **FastAPI :** [**https://fastapi.tiangolo.com/**](https://fastapi.tiangolo.com/)
* **Streamlit :** [**https://docs.streamlit.io/**](https://docs.streamlit.io/)
* **IBM Watsonx AI & Granite Models:**[**https://www.ibm.com/products/watsonx-ai/foundation-model**](https://www.ibm.com/products/watsonx-ai/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**](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 1.1: Research and select the appropriate generative AI model**

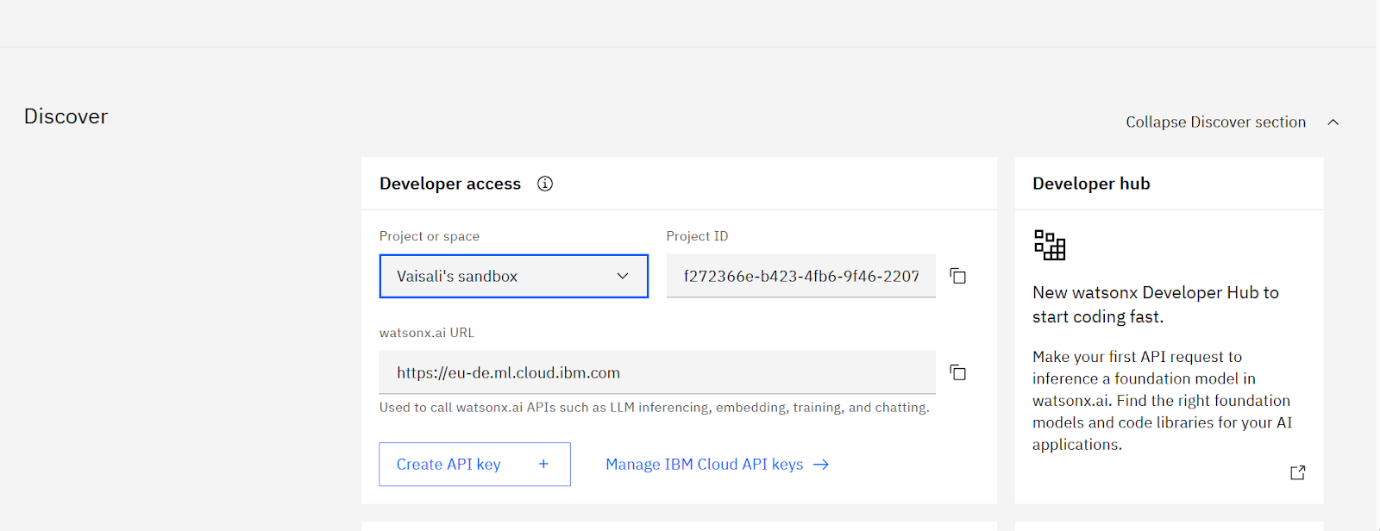
1. **Understand the Project Requirements:** Analyze the goals of the SmartSDLC platform, particularly the need for automating SDLC phases like requirement analysis, code generation, test creation, bug fixing, and documentation. This understanding helps narrow down which generative AI models best suit each task.

1. **Explore Available Models:** Study IBM’s Watsonx Granite model documentation to compare available foundation models. Focus on those capable of handling natural language processing and code synthesis, such as granite-13b-chat-v1 for language understanding and granite-20b-code-instruct for structured code generation and summarization.

1. **Evaluate Model Performance:** Evaluate models based on metrics like accuracy, latency, and quality of generated responses across different SDLC tasks. Use prior benchmarks or run pilot queries (e.g., “generate a function to validate email input”) to assess model suitability.

1. **Select Optimal Models:**Finalize the two best-fit Granite models—granite-13b-chat-v1 for interacting with natural language prompts and granite-20b-code-instruct for multilingual and functional code outputs. These models will power the core logic of SmartSDLC’s automation system.

1. **Generating Watsonx ai API key**



1. **Selecting the model**

**Activity 1.2: Define the architecture of the application.**

1. **Draft a System Architecture Diagram:**

**Create a high-level visual representation that includes the backend (FastAPI), the frontend (Streamlit), the AI layer (Watsonx APIs), and service modules (LangChain, GitHub integration). Include flow direction between components such as file uploads, AI interactions, and response rendering.**

1. **Define Frontend-Backend Interaction:**

**Detail how the Streamlit UI captures user inputs (PDF uploads, text prompts, buggy code) and sends them to FastAPI endpoints. Also, document how the backend routes handle logic and return structured outputs like user stories or code blocks.**

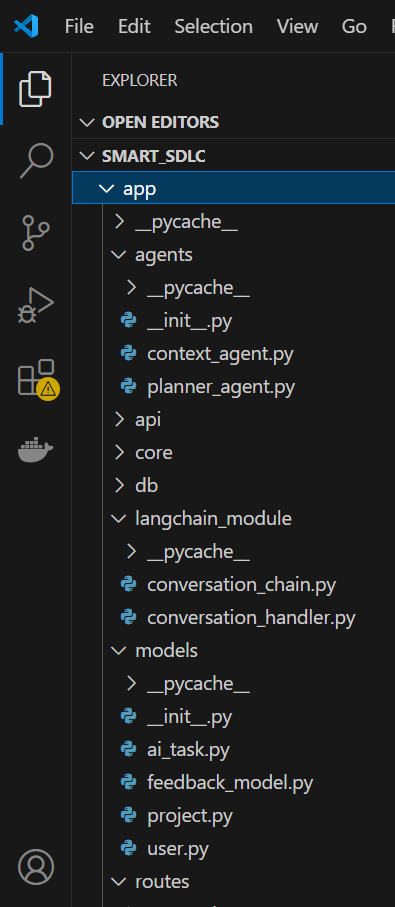
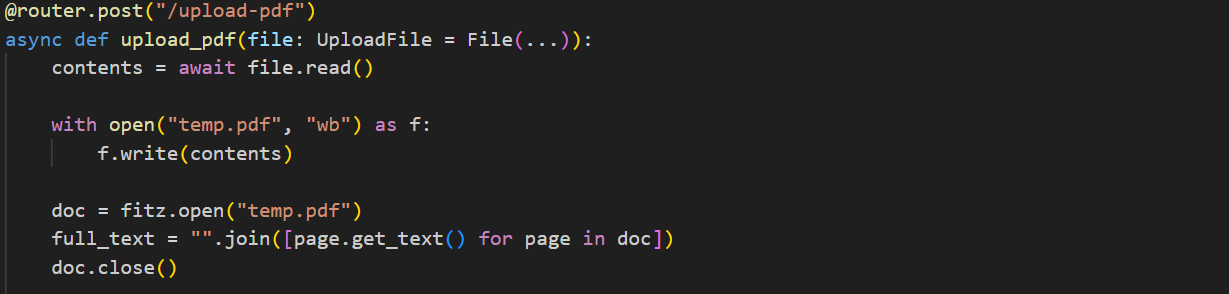
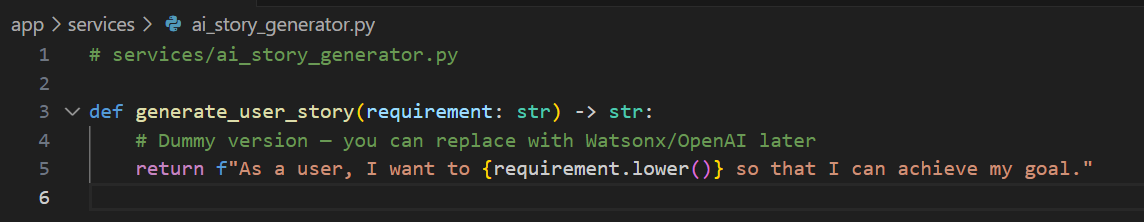
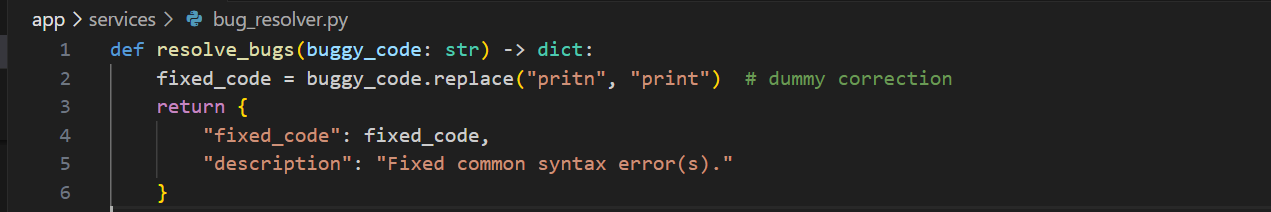
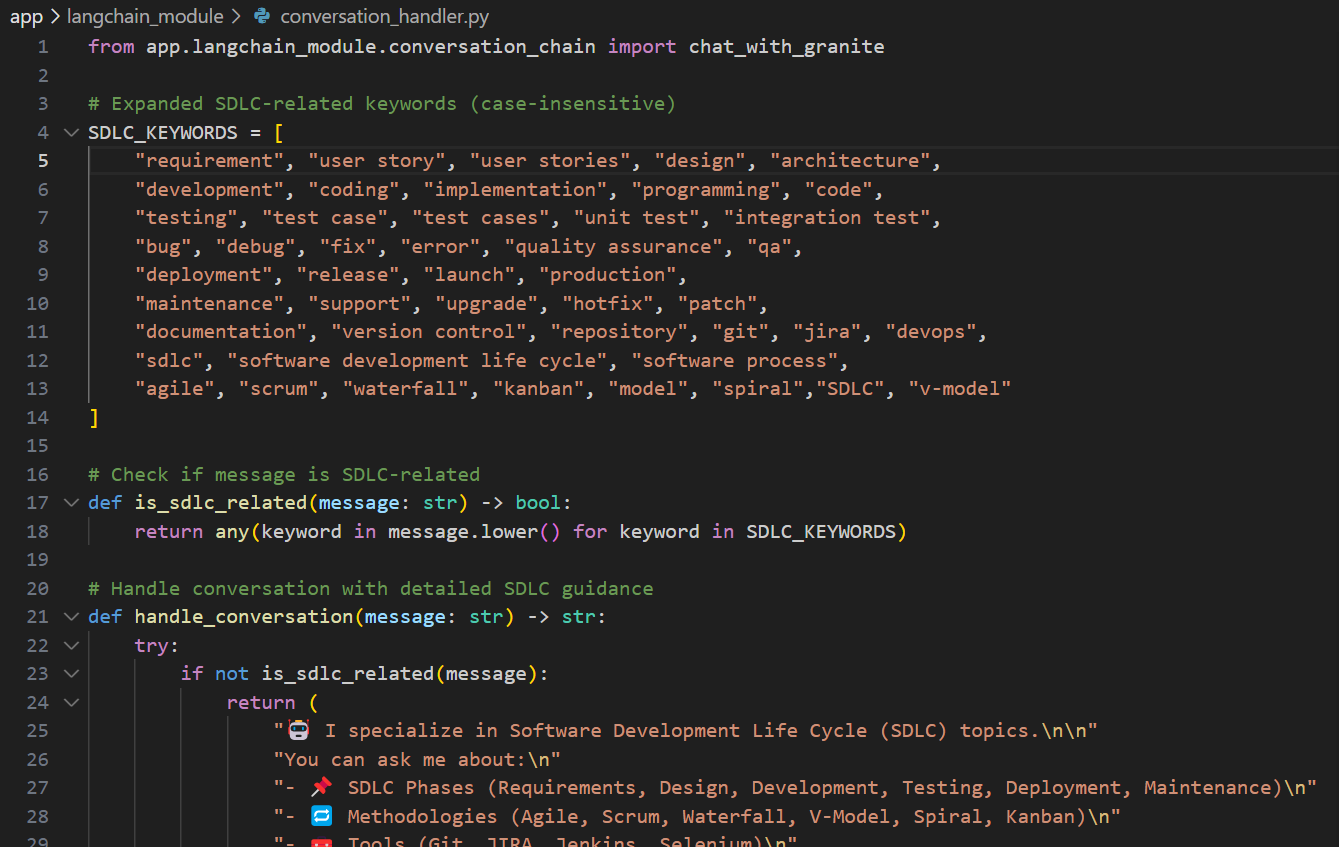
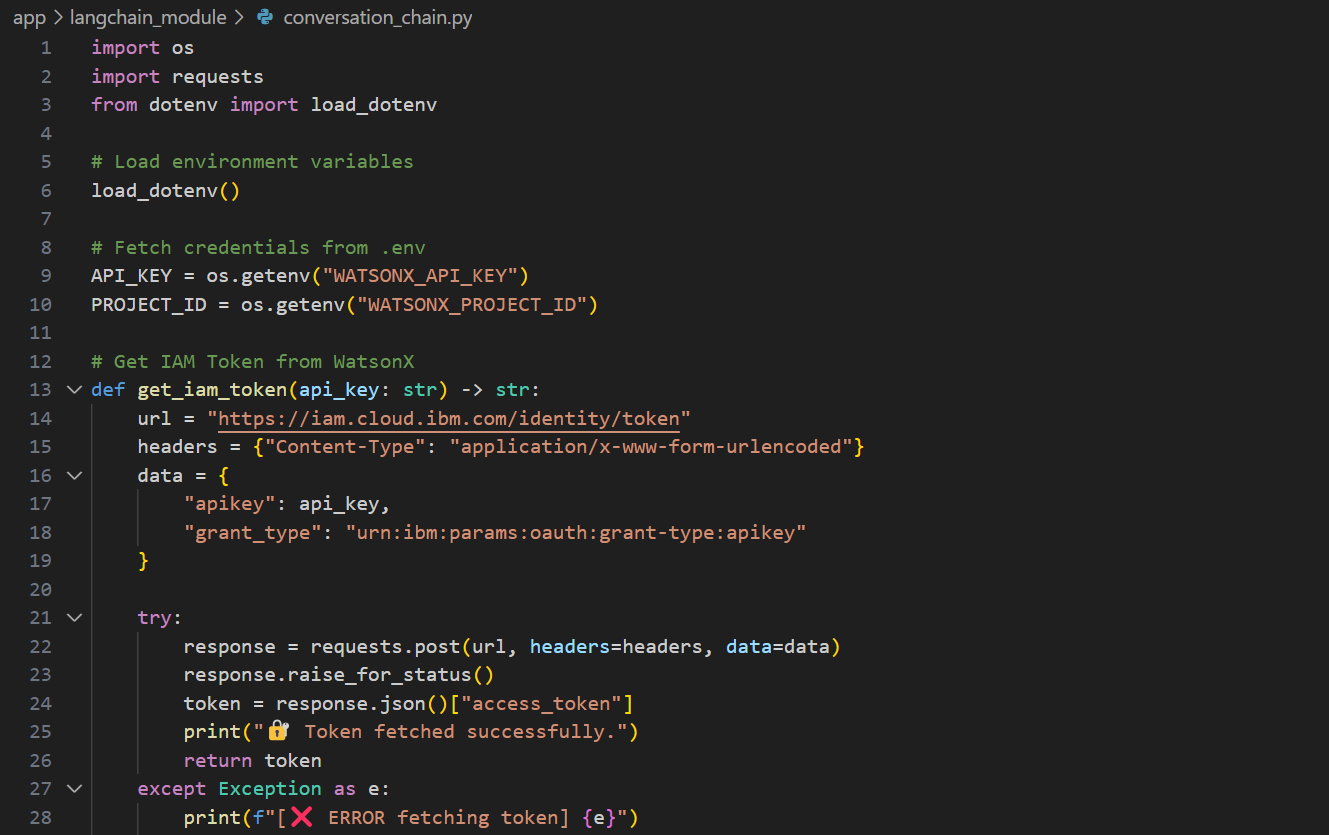
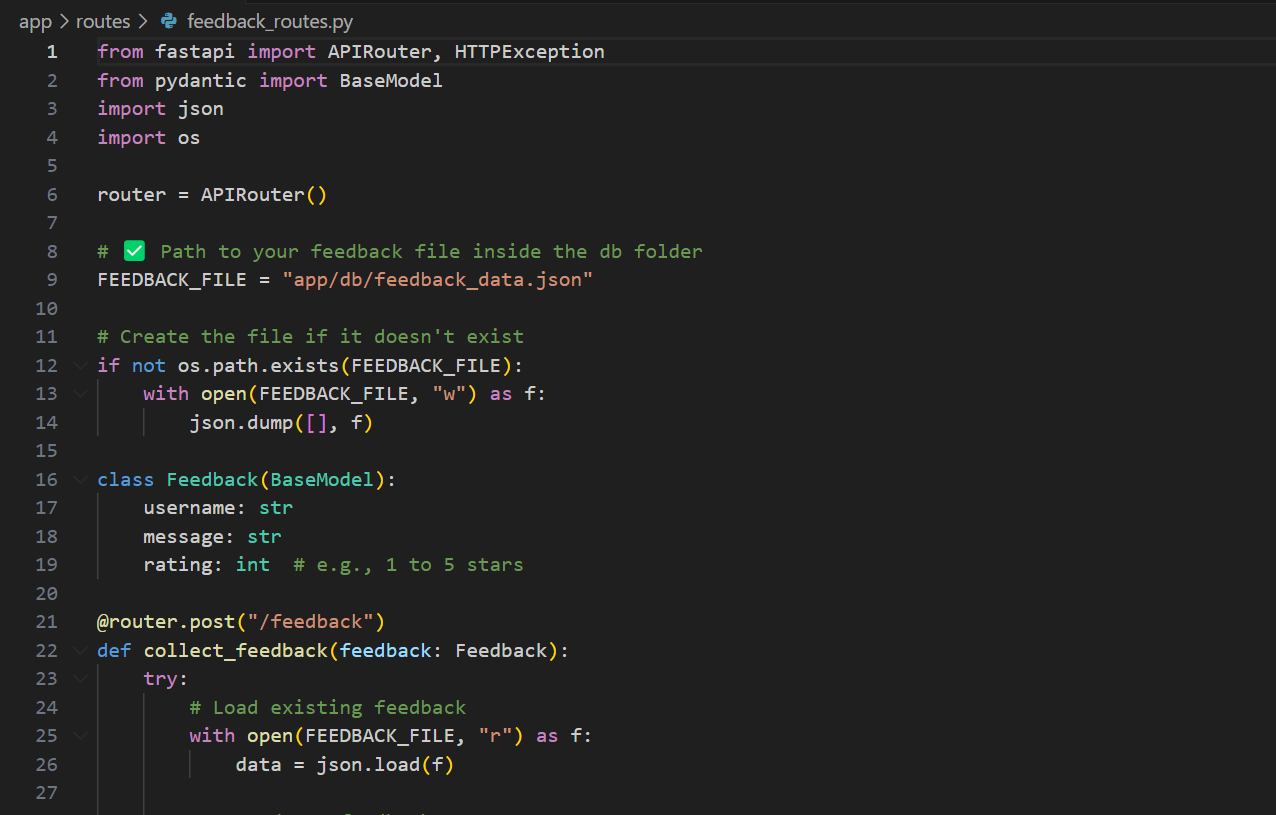
1. **Backend Service Responsibilities:  
    Define roles of each backend module:**
   * **PDF processing and classification logic (using PyMuPDF).**
   * **Prompt generation and API communication with Watsonx.**
   * **LangChain agent for orchestration.**
   * **Auth module for user sessions and hashed logins.**
2. **AI Integration Flow:  
    Outline how API calls are made to Watsonx (or optionally LangChain), and how responses are parsed and displayed back to the user. Specify if fallback models or retries will be incorporated.**

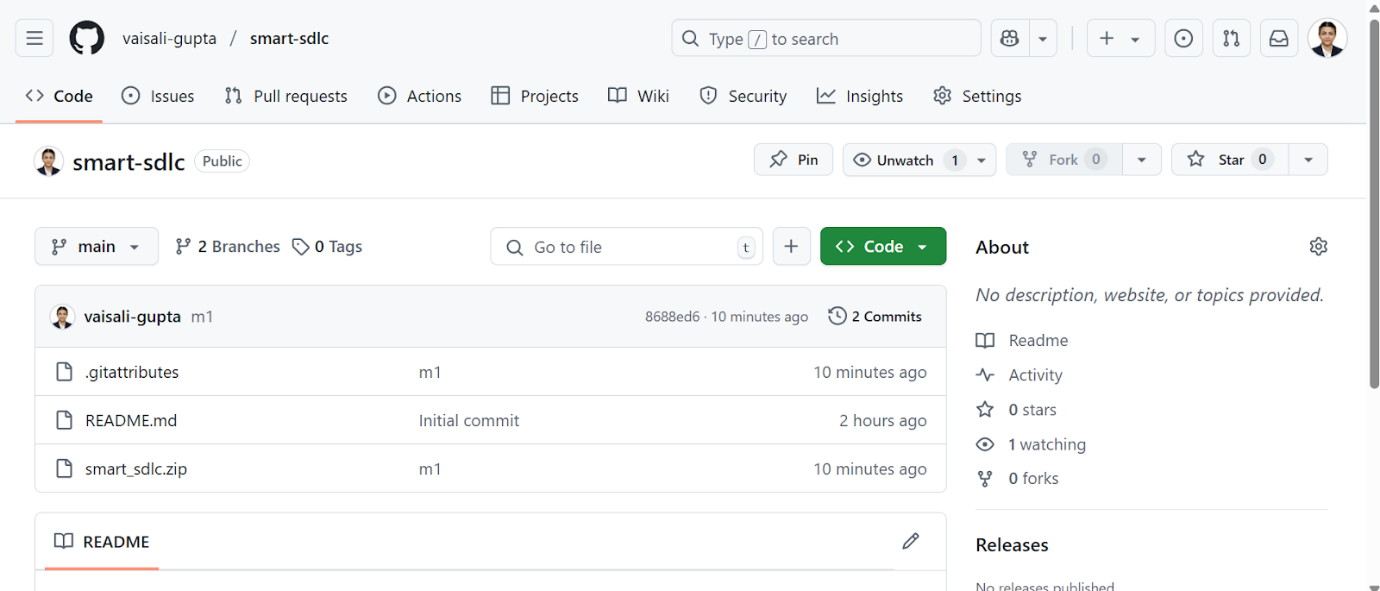
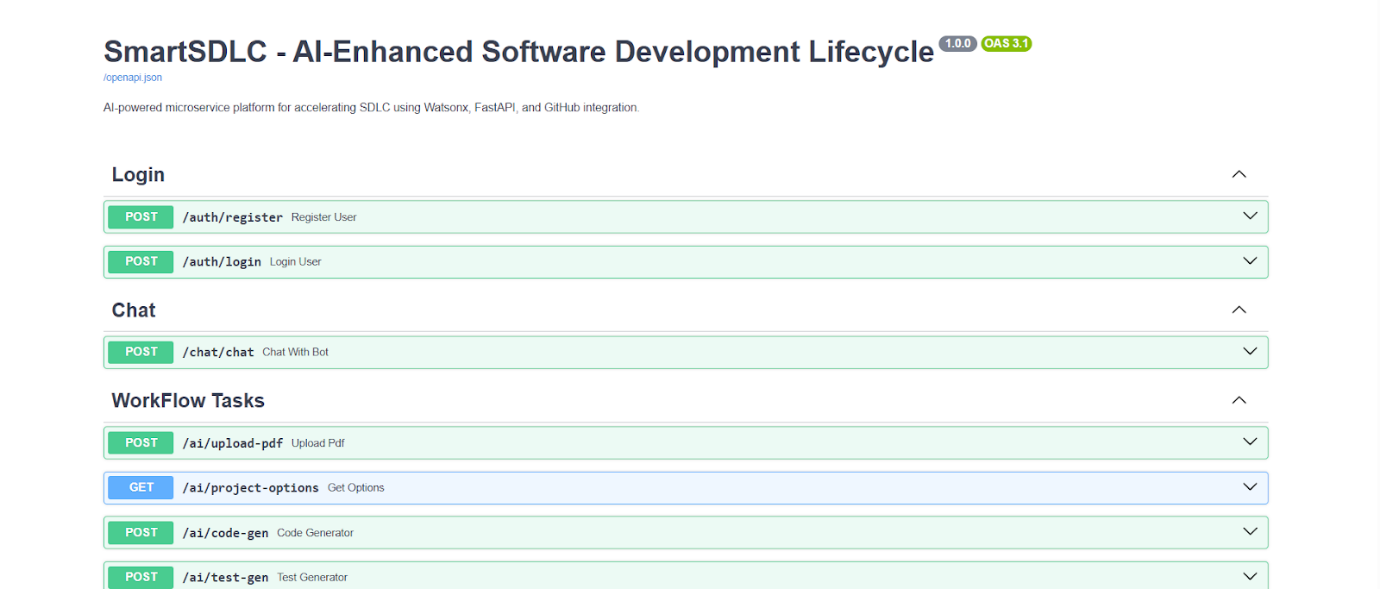
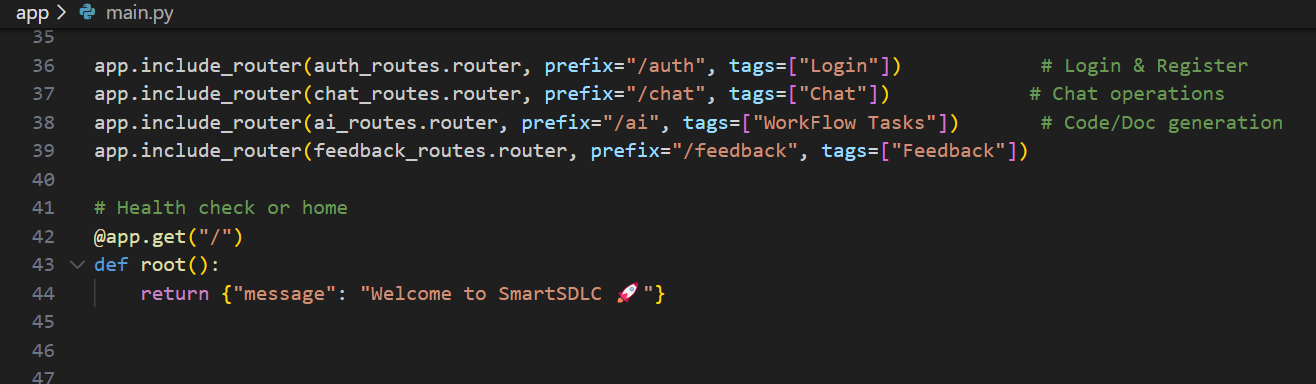
**Activity 1.3: Set up the development environment:**

1. **Install Python and Pip:** Ensure Python 3.10 is installed for compatibility with Watsonx SDK. Install pip to manage dependencies like FastAPI, Streamlit, and PyMuPDF.
2. **Create a Virtual Environment:** Use venv to create isolated environments for backend and frontend:

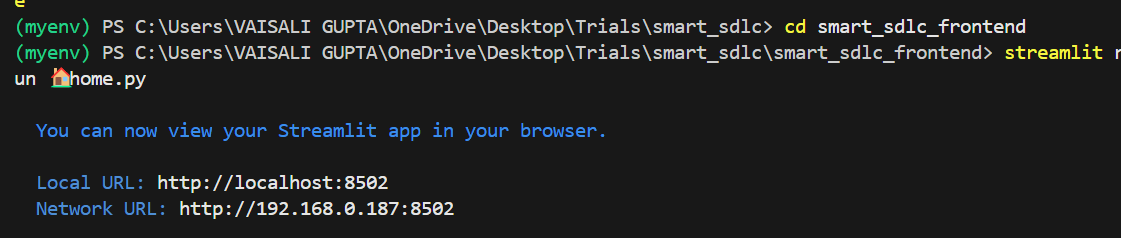
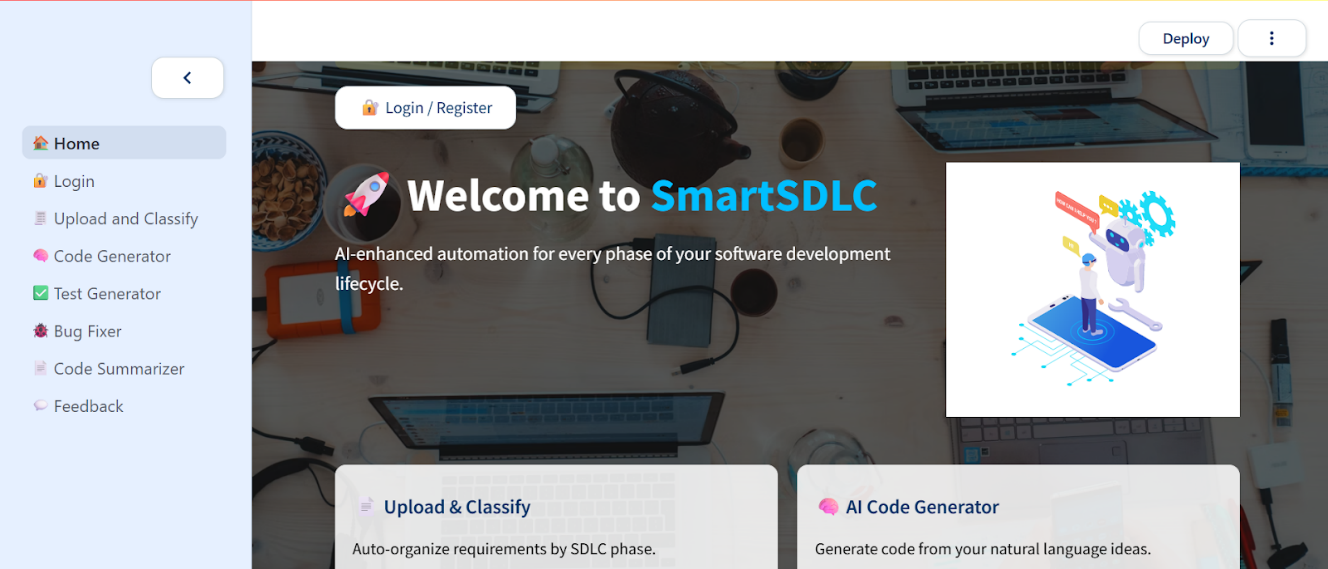
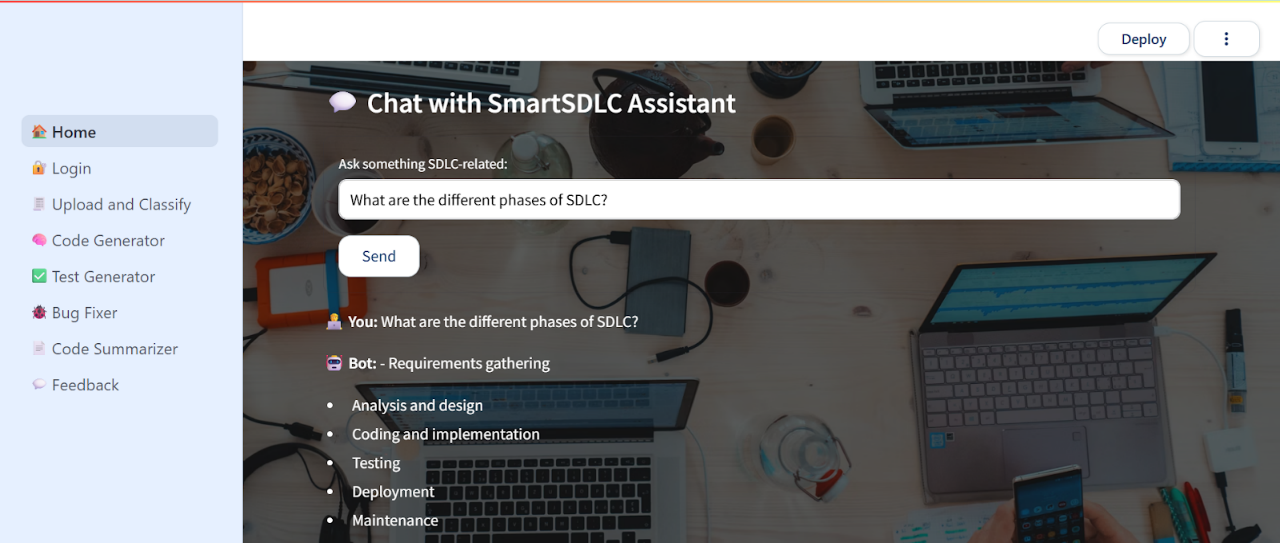
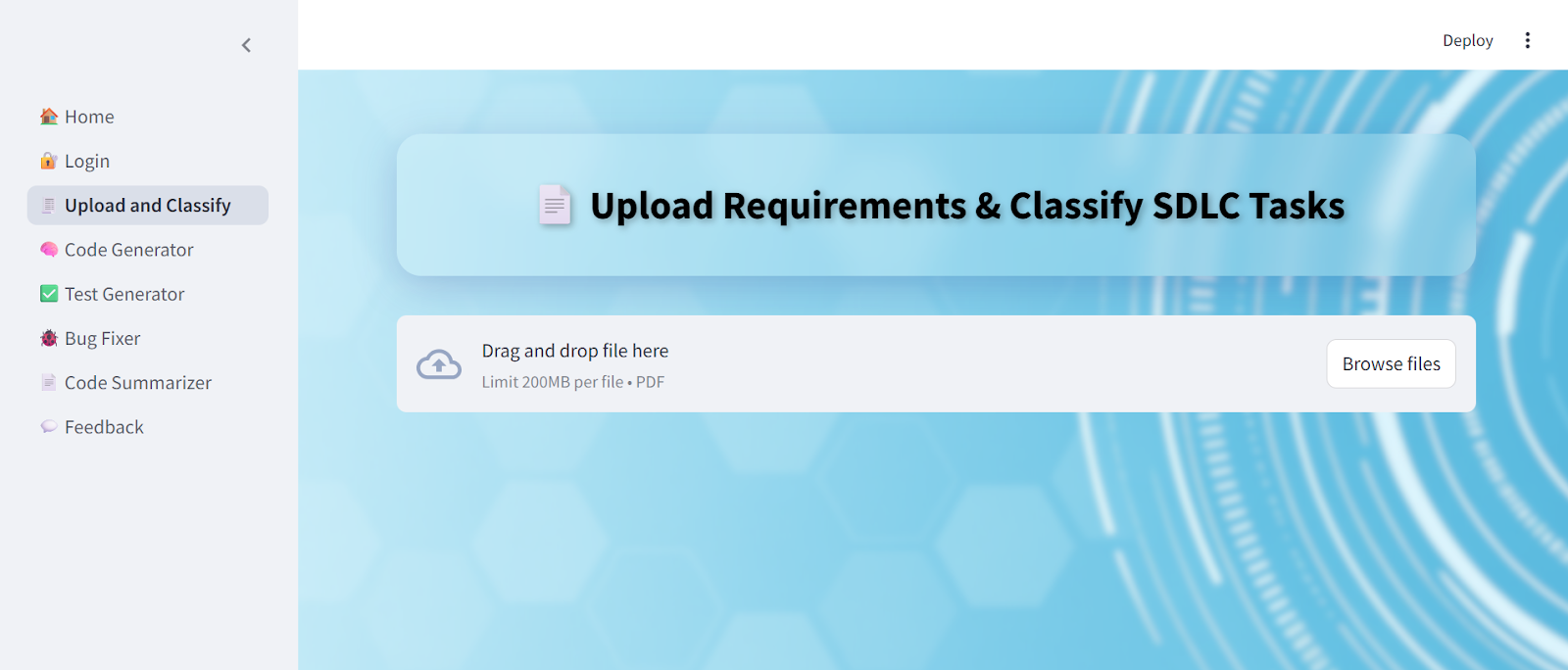
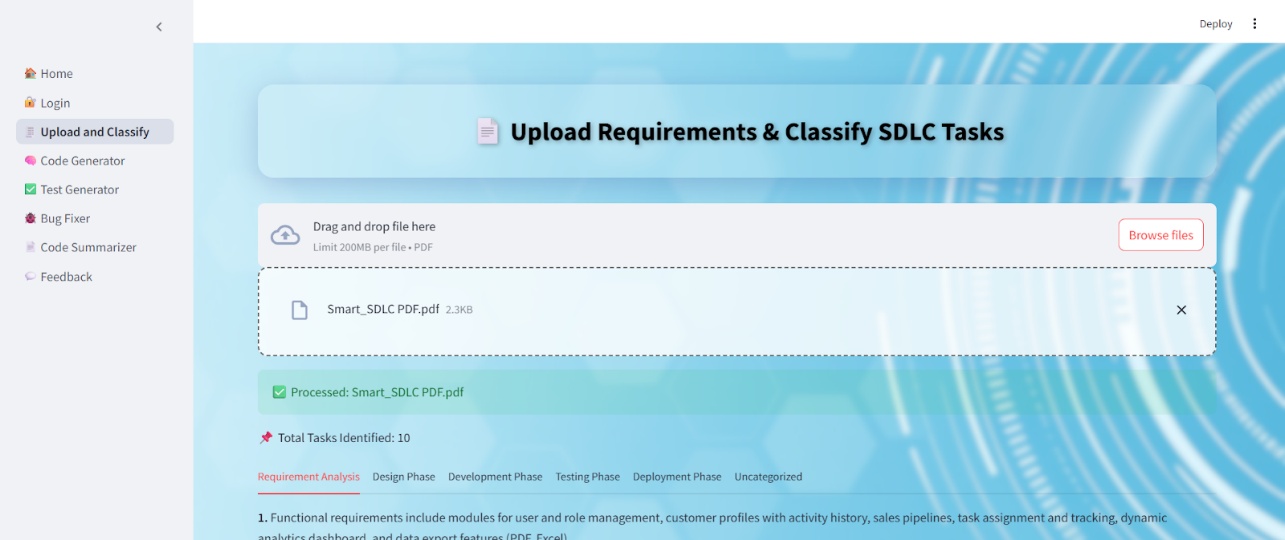
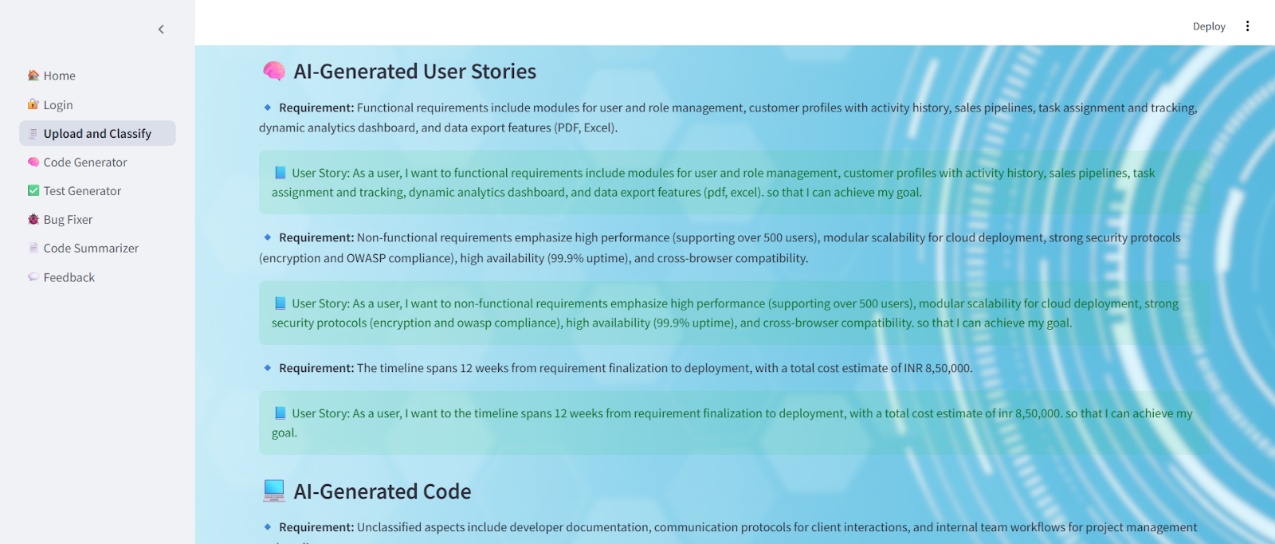
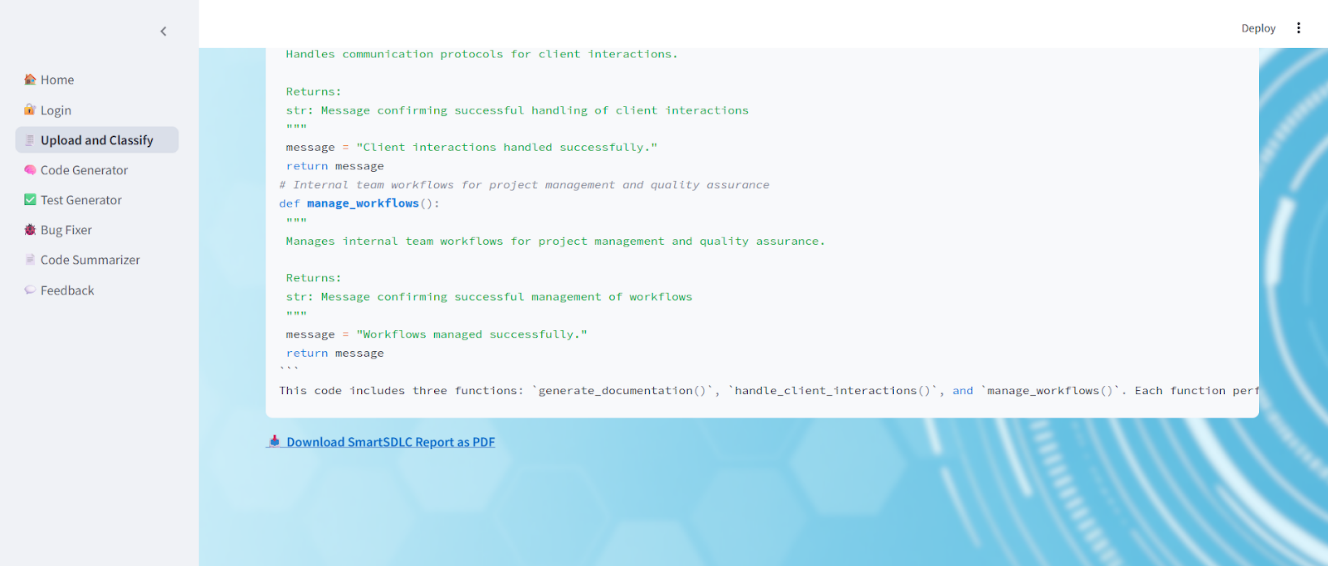
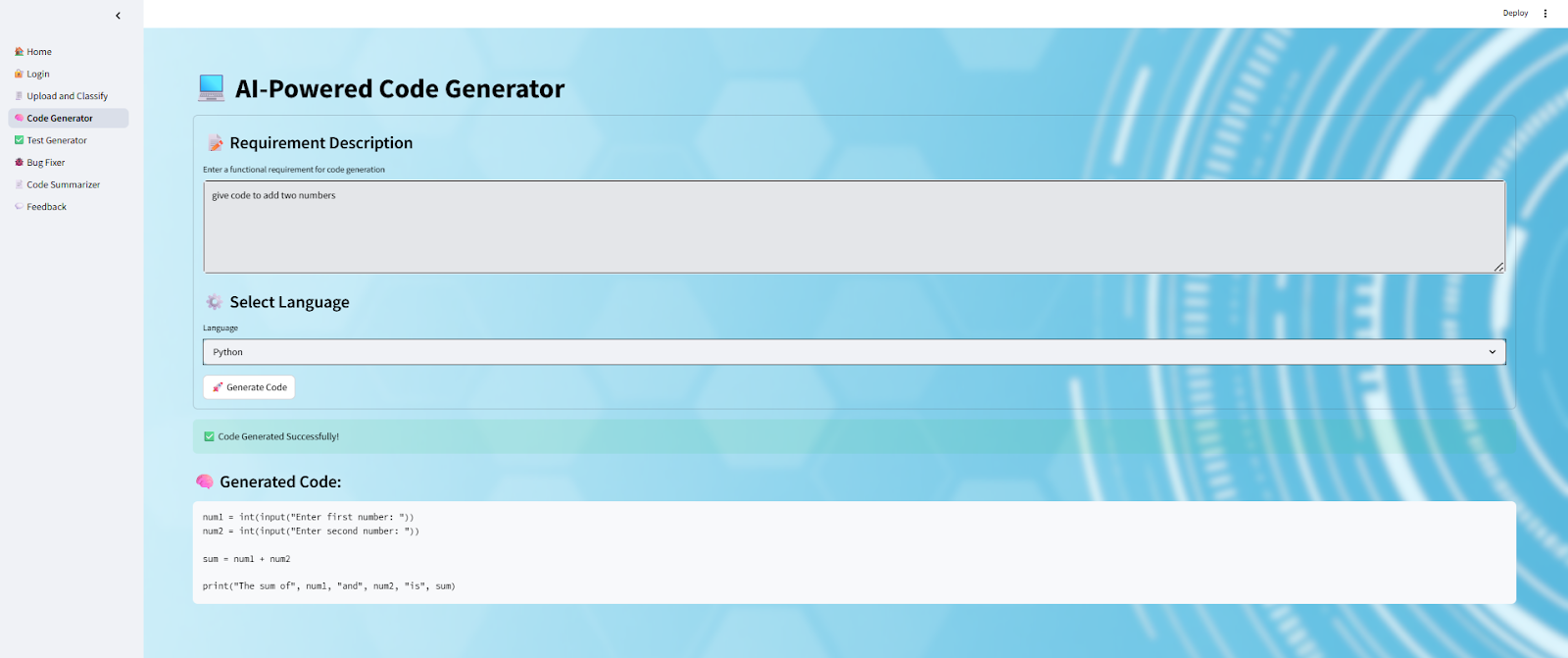
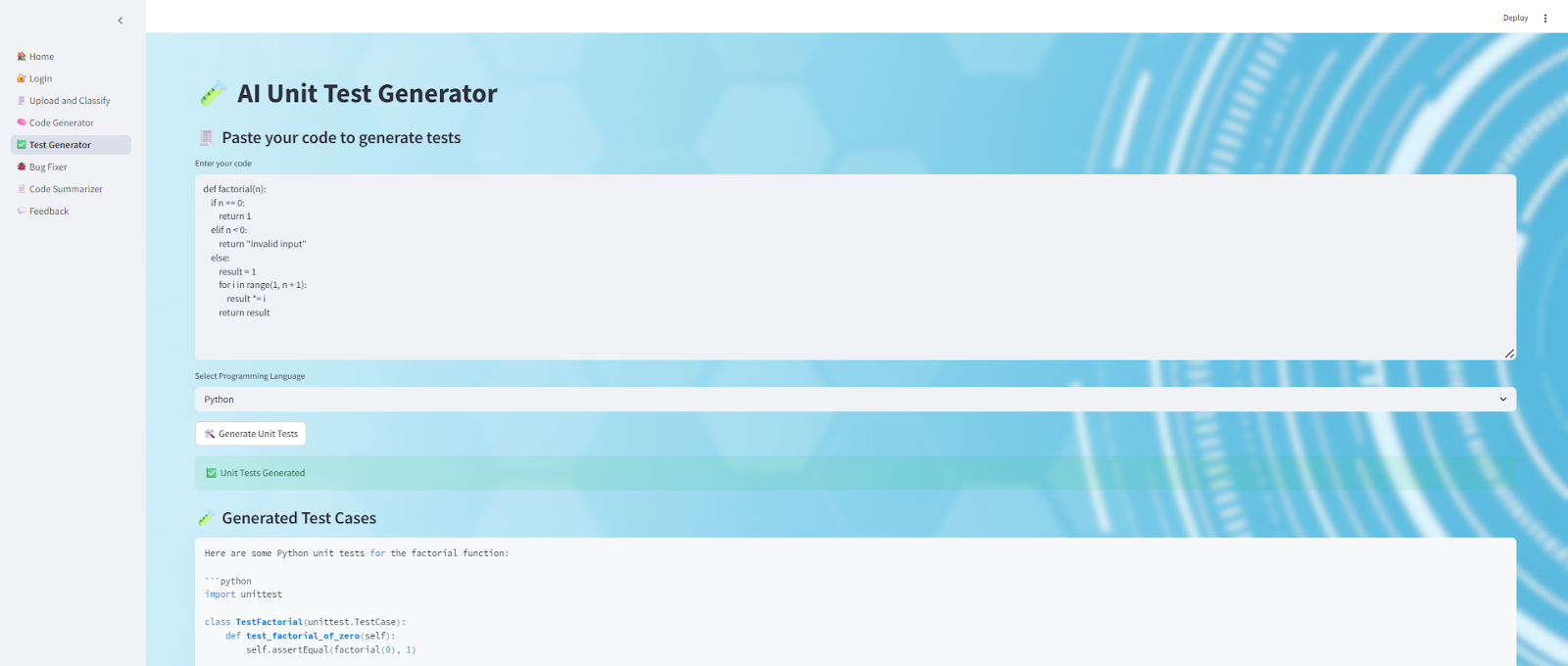
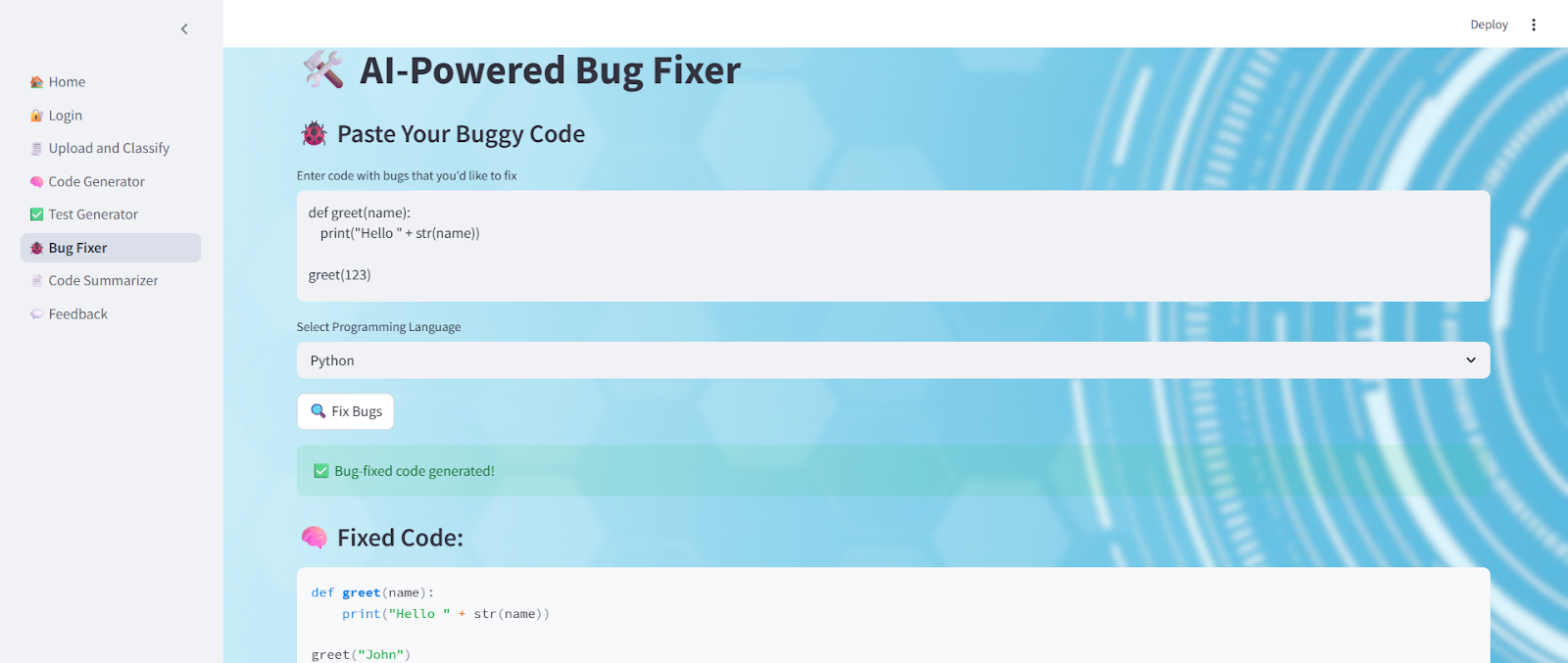
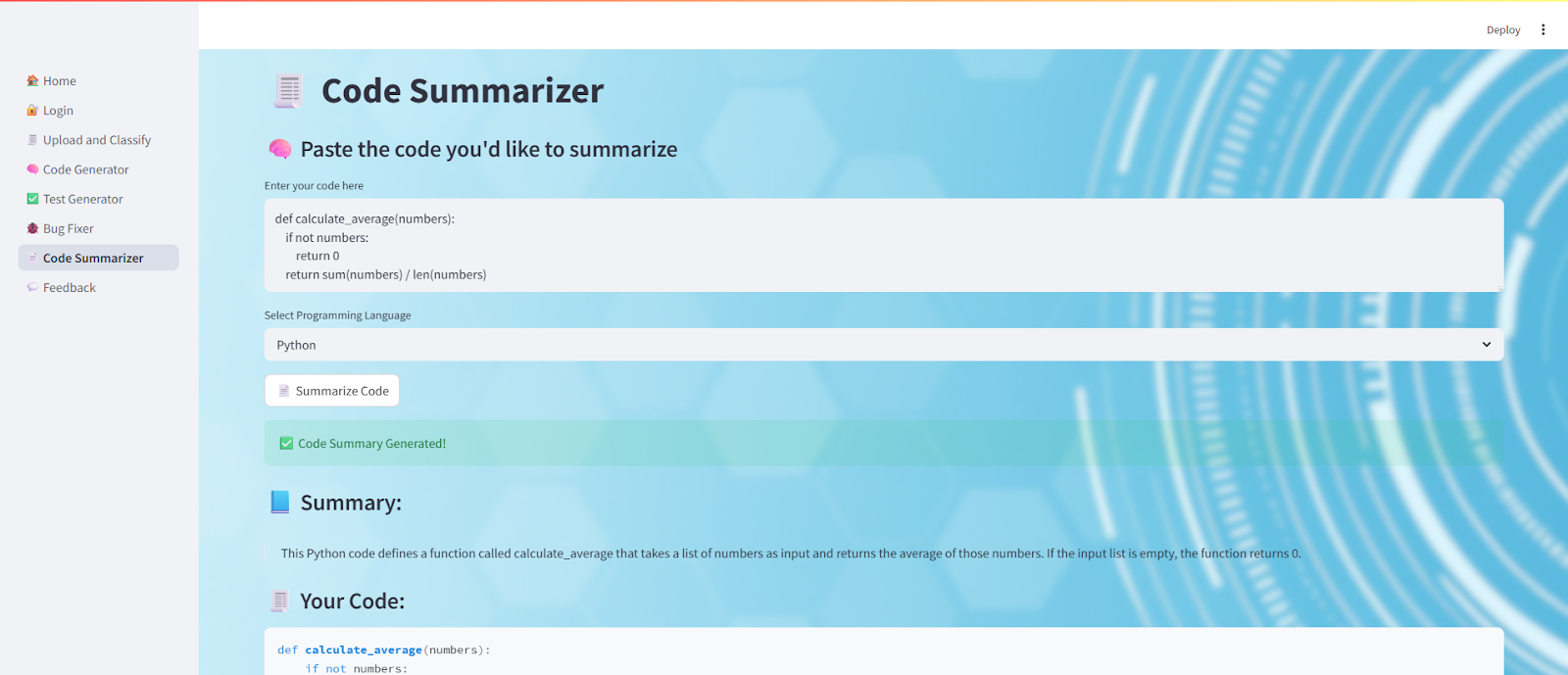
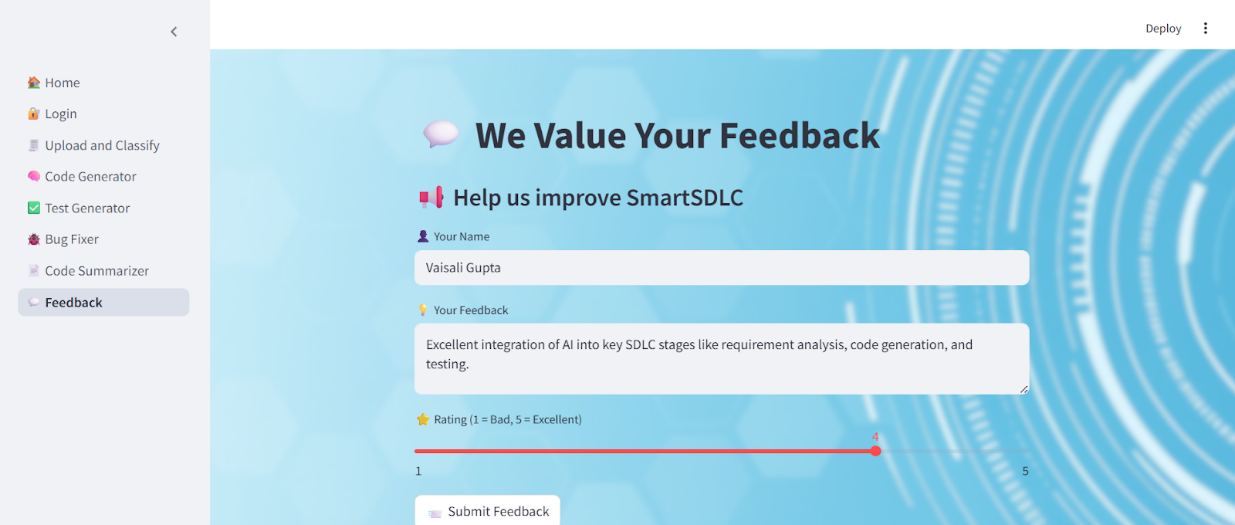
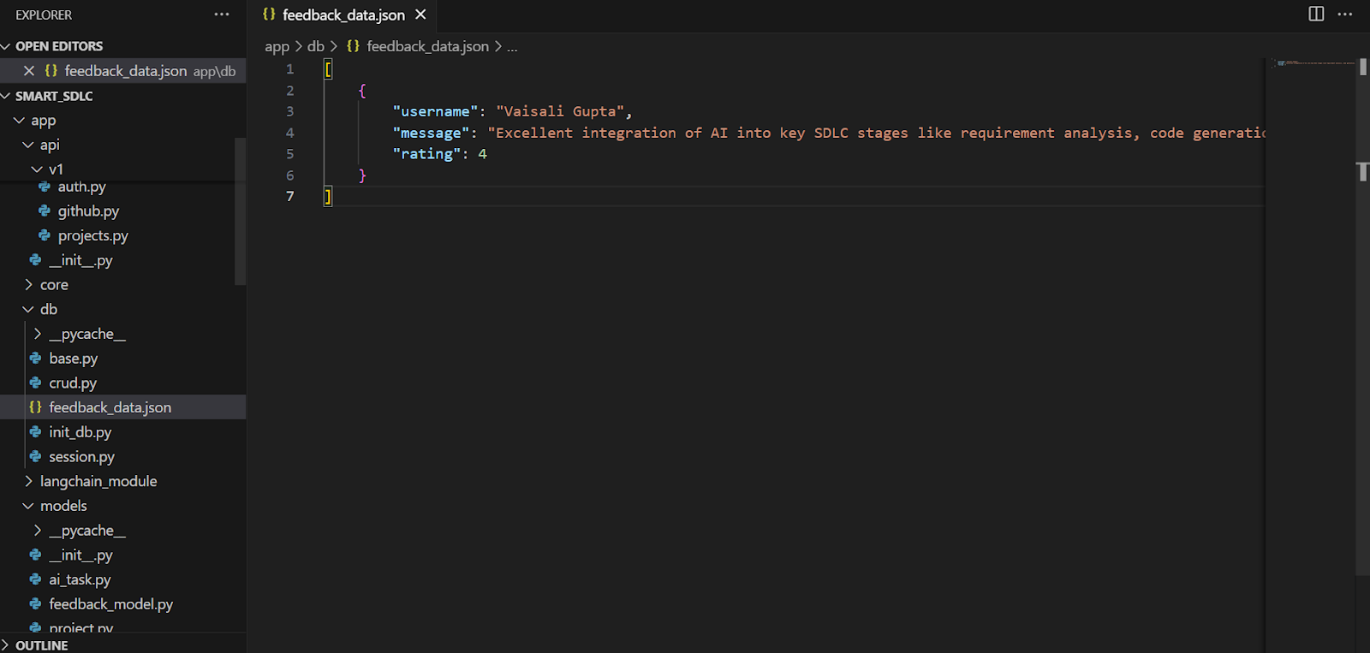
**python -m venv myenv**

**myenv\Scripts\activate**

1. **Install Required Libraries:** Install all required packages such as fastapi, uvicorn, pymupdf, requests, langchain, streamlit, and ibm-watsonx-ai using pip. Store these in requirements.txt.
2. **Set Up API Keys and Env Files:** Generate IBM Cloud API key and store it securely in a .env file. Include other configs like model IDs and LangChain agent settings.
3. **Organize Project Structure:** Set up the initial folder structure for both frontend (smart\_sdlc\_frontend/) and backend (app/), ensuring proper separation of api/, services/, models/, and utils/ modules for clean development.
4. **file structure**
5. 
6. **Milestone 2: Core Functionalities Development**
7. 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.
8. **Activity 2.1: Develop the core functionalities:**
9. This activity focuses on building the logic behind each AI-powered SDLC feature. These functionalities are the heart of SmartSDLC and are implemented using IBM Watsonx and LangChain integration.
10. **1. AI-Powered Requirement Analysis**
11. **Module:** ai\_story\_generator.py
12. **Description:** Extracts text from uploaded PDFs and uses AI to classify each sentence into SDLC phases (Requirements, Design, Development, etc.). Then it converts relevant sentences into structured user stories to be used in planning or code generation.
13. 
14. 
15. **2. Multilingual Code Generation**
16. **Module:** code\_generator.py
17. **Description:** Takes task descriptions or user stories and generates clean, production-ready code in Python or other languages using granite-20b-code-instruct. Ideal for initial development tasks or boilerplate generation.
18. 
19. **3. AI-Driven Test Case Generation**
20. **Module:** Likely part of code\_generator.py
21. **Description:** Generates unit tests or integration test cases based on the generated or user-provided code. Helps developers quickly validate core logic.
22. **4. Bug Fixing and Code Correction**
23. **Module:** bug\_resolver.py
24. **Description:** Takes buggy code snippets and automatically identifies and fixes errors using AI. Returns a clean version of the code along with optional explanations.  
    
25. **5. Code Summarization and Documentation**
26. **Module:** doc\_generator.py
27. **Description:** Extracts summaries or generates documentation-style descriptions of code blocks. Helps with code readability and maintainability.  
    
28. **6. Chatbot Assistance**
29. **Modules:** conversation\_handler.py, chat\_routes.py
30. **Description:** A floating AI chatbot helps users navigate the platform, answer questions, or give suggestions on SDLC tasks. Powered by LangChain and Watsonx integration.
31. 
32. 
34. **7. Project Feedback and Learning Loop**
35. **Module:** feedback\_routes.py, feedback\_model.py
36. **Description:** Collects user feedback on AI output and stores it in a database or JSON for improving prompts, models, or usability in future iterations.
37. 
38. 

41. **8. GitHub Integration**
42. **Module:** github\_service.py
43. **Description:** Automates GitHub workflows such as pushing AI-generated code to a repo, opening issues, or syncing generated documentation.
44. 
45. **Output: Fast API Backend**
46. 
47. *Figure 2: SmartSDLC API Documentation – Explore and interact with AI-powered microservices for login, chat, and workflow automation via FastAPI Swagger UI.*
48. **Activity 3.1: Writing the Main Application Logic in main.py**
49. **1: Define the Core Routes in main.py**
50. Import and include routers for each of the SmartSDLC’s functional areas: requirement analysis, code generation, testing, summarization, bug fixing, authentication, chatbot, and feedback.
51. Use FastAPI’s include\_router() method to modularize the route definitions and separate concerns across files.
52. Example router mounts:
53. /ai: Handles AI-based endpoints like code generation, test creation, bug fixing.
54. /auth: Manages login, registration, and user validation.
55. /chat: Powers the floating chatbot via LangChain.
56. /feedback: Handles submission and storage of feedback.
57. 
58. **2: Set Up Route Handling and Middleware**
59. FastAPI middleware is configured using CORSMiddleware to ensure the frontend (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.
60. A screenshot of a computer

    AI-generated content may be incorrect.
61. **3: Implement Application Metadata and Root Route**
62. 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).
63. A screen shot of a computer

    AI-generated content may be incorrect.
65. **Milestone 4: Frontend Development**
66. 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.
67. **Milestone 4: Frontend Development**
68. **Milestone 5: Deployment**
69. 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.
70. **Activity 5.1: Preparing the Application for Local Deployment**
71. **1: Set Up a Virtual Environment**
72. Create a Python virtual environment to manage dependencies and avoid conflicts with other projects.
73. Activate the environment and install dependencies listed in requirements.txt to ensure all libraries (FastAPI, Streamlit, Watsonx SDK, etc.) are available.
74. **2: Configure Environment Variables**
75. Set environment variables for sensitive data such as **IBM Watsonx API key**, **model IDs**, and database URLs.
76. Create a .env file in your project root to securely store and load these settings during runtime.
77. WATSONX\_API\_KEY=your\_ibm\_key\_here
78. WATSONX\_PROJECT\_ID=your\_project\_id
79. WATSONX\_MODEL\_ID=granite-20b-code-instruct
80. These values are loaded using python-dotenv inside your backend (e.g., config.py).
81. **Activity 5.1: Preparing the Application for Local Deployment**
82. **1: Set Up a Virtual Environment**
83. Create a Python virtual environment to manage dependencies and avoid conflicts with other projects.
84. Activate the environment and install dependencies listed in requirements.txt to ensure all libraries (FastAPI, Streamlit, Watsonx SDK, etc.) are available.
85. **2: Configure Environment Variables**
86. Set environment variables for sensitive data such as **IBM Watsonx API key**, **model IDs**, and database URLs.
87. Create a .env file in your project root to securely store and load these settings during runtime.
88. WATSONX\_API\_KEY=your\_ibm\_key\_here
89. WATSONX\_PROJECT\_ID=your\_project\_id
90. WATSONX\_MODEL\_ID=granite-20b-code-instruct
91. These values are loaded using python-dotenv inside your backend (e.g., config.py).
92. **Activity 5.2: Testing and Verifying Local Deployment**
93. **1: Launch and Access the Application Locally**
94. **Start the FastAPI backend** using Uvicorn: uvicorn app.main:app --reload
95. **Run the Streamlit frontend**: streamlit run frontend/Home.py
97. Open your browser and navigate to:
98. **Streamlit UI:** [http://localhost:850](http://localhost:8501/)2
99. **FastAPI Swagger Docs:** <http://127.0.0.1:8000/docs>
100. Test each SmartSDLC feature (requirement upload, code generation, bug fixing, chatbot, feedback) to ensure everything is connected and functioning correctly.
102. **Run the Web Application**
103. Now type “streamlit run ????home.py” command
104. Navigate to the localhost where you can view your web page
106. 
108. **Milestone 6: Conclusion**
109. 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.
110. 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.
111. Overall, SmartSDLC not only improves productivity and accuracy but also sets 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.
112. **The web application will open in the web browser:**
114. 
115. *Figure 3: "SmartSDLC Dashboard – Your AI-powered command center for accelerating every phase of the software development lifecycle."*
117. 
118. *Figure 4: "Interactive Chat with SmartSDLC Assistant – Instantly learn SDLC concepts through AI-powered conversations."*
120. 
121. *Figure 5: Input: "Upload and Classify – Effortlessly convert raw PDF requirements into structured SDLC tasks using AI."*
123. 
124. *Figure 6: Output: "Smart Requirement Classifier – Instantly extract and categorize software tasks into SDLC phases from your uploaded PDF."*
126. 
127. *Figure 7: "AI-Generated User Stories – Transform raw requirements into structured user-centric stories for agile development."*
129. 
130. *Figure 8: "Auto-Generated Summary with Downloadable Insights – Review AI-explained code and export the SmartSDLC report as a professional PDF."*
132. 
133. *Figure 9: "AI Code Generator – Instantly convert natural language requirements into clean, executable code with just one click."*
134. 
135. *Figure 10: AI-powered tool to auto-generate unit tests from user-provided code.*
137. 
138. *Figure 11: "AI-Powered Bug Fixer – Instantly detect and correct code issues for cleaner, error-free execution."*
140. 
141. *Figure 12: “AI-powered Code Summarizer interface displaying a generated summary for the provided code snippet.”*
143. 
144. *Figure 13: "User submitting feedback on the SmartSDLC system, highlighting effective AI integration across SDLC phases"*
146. 
147. *Figure 14:User feedback data stored in feedback\_data.json file within the SmartSDLC application's database module.*