SmartSDLC is a full-stack, AI-powered platform that redefines the traditional Software Development Lifecycle (SDLC) by automating key stages using advanced Natural Language Processing (NLP) and Generative AI technologies.

It is not just a tool — it's an intelligent ecosystem that allows teams to convert unstructured requirements into code, test cases, and documentation instantly, thereby minimizing manual intervention, enhancing accuracy, and accelerating the delivery pipeline.

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.

Scenario 4: Test Case Generator

Test Case Generator focuses on quality assurance, where users provide functional code or a requirement, and the AI generates suitable test cases. These are structured using familiar testing frameworks like unittest or pytest, enabling seamless validation and automation of software testing. This module eliminates the need for manual test writing, ensuring consistency and completeness in test coverage.

Scenario 5: Code Summarizer

Code Summarizer, the platform aids documentation by accepting any source code snippet or module and generating a human-readable explanation. Watsonx analyzes the logic and purpose of the code, then summarizes its function and use cases. This feature is especially helpful for onboarding new developers or for maintaining long-term documentation, making complex codebases easier to understand.

Scenario 6: Floating AI Chatbot Assistant

Floating AI Chatbot Assistant provides real-time, conversational support throughout the application. Integrated using LangChain, the chatbot responds intelligently to user queries about the SDLC, such as “How do I write a unit test?” or “What is requirement analysis?”. The backend handles prompt routing based on keyword detection, and the frontend presents the response in a styled chat interface, offering an intuitive and interactive help system.

Collectively, these scenarios demonstrate how SmartSDLC intelligently automates core development tasks, enhances team collaboration, and empowers both technical and non-technical users to efficiently engage with the software development process.

The content outlines a comprehensive three-part activity for developing a SmartSDLC platform powered by IBM's Watsonx generative AI models. It begins with a research phase to select appropriate AI models for automating various software development lifecycle tasks, focusing on evaluating IBM's Granite models for both natural language processing and code generation capabilities. The second activity involves defining the system architecture, detailing the interaction between the Streamlit frontend and FastAPI backend, and establishing the AI integration workflow through Watsonx APIs and LangChain orchestration. The final activity provides step-by-step instructions for setting up the development environment, including installing dependencies, creating virtual environments, securing API keys, and organizing the project structure for efficient development across frontend and backend components.