AN INTERNSHIP REPORT

ON

SmartSDLC – AI-Enhanced Software Development Lifecycle

Team ID : LTVIP2025TMID37560

Team Size : 4

Team Leader : Keerthi Tirumani

Team member : Chodisetti Manasa Laxmi Sailaja

Team member : Daivala Rajamouli

Team member : Venu Gopala Swamy Manikala

College Name:

SWARNANDHRA COLLEGE OF ENGINEERING AND TECHNOLOGY

Abstract

SmartSDLC is an AI-powered automation framework designed to revolutionize the Software Development Life Cycle (SDLC) by integrating cutting-edge generative AI models and modern microservice architecture. The platform leverages IBM Watsonx's Granite foundation models and LangChain agents to automate key SDLC phases—requirement analysis, code generation, test case creation, bug detection and resolution, and documentation.

The backend, built with FastAPI, provides robust routing, authentication, and service orchestration, while the frontend, developed in Streamlit, offers a clean and responsive user interface. Each functionality is modularized to ensure scalability and maintainability. The system architecture features tightly coupled interactions between user input, AI model inference, and real-time feedback loops, supported by APIs and secure environment configurations.

SmartSDLC introduces eight core AI-enabled services, including an interactive chatbot assistant and GitHub integration, to enhance developer productivity. The development environment emphasizes clean code separation, .env-based configuration, and version-controlled service layers. Deployed locally with provisions for future cloud migration, SmartSDLC is a future-ready platform positioned to streamline and intelligently assist software teams in delivering quality code faster.

Table of Contents

---

1. Executive Summary

SmartSDLC is an end-to-end AI-powered automation framework for software engineering teams. It integrates powerful generative models from IBM Watsonx and LangChain agents to intelligently automate SDLC activities such as:

- Requirement analysis
- Source code generation
- Test case creation
- Bug detection and correction
- Code summarization and documentation
- Interactive chatbot guidance
- Feedback learning loop
- GitHub CI/CD integration

---

2. Project Objectives and Scope

The primary objective of SmartSDLC is to accelerate software development while maintaining quality and reducing human error. The solution is built upon modular FastAPI microservices, Streamlit as the UI layer, and Watsonx foundation models as the AI core.

---

3. Activity 1.1: Research and Select Generative AI Models

Understanding the Requirements

SmartSDLC must analyze requirements, generate multilingual code, and explain code behavior. These demands require models proficient in:

- Natural Language Understanding (NLU)
- Code synthesis
- Summarization

Exploring IBM Watsonx Granite Models

Models Reviewed:

- `granite-13b-chat-v1`: NLP generalist for prompts, conversation, summarization
- `granite-20b-code-instruct`: High-accuracy, multilingual model designed for structured code generation

Model Evaluation Process

We performed pilot tasks (e.g., “validate email address”) across multiple models to assess:

- Generation time (latency)
- Output quality (readability, correctness)
- Error handling (edge cases)

Final Selection

- **granite-13b-chat-v1**: For NLP tasks including user prompts and documentation
- **granite-20b-code-instruct**: For all coding-related features

---

4. Activity 1.2: Define System Architecture

Architecture Diagram Overview

The system comprises:

- Streamlit Frontend
- FastAPI Backend
- Watsonx AI Layer
- LangChain Agent Layer
- External Services: GitHub, Database

Data Flow: User Input → Streamlit → FastAPI → AI Model → Response → UI Display

Component Responsibilities

- **Frontend**: Input capture (files, prompts)
- **Backend**: Routing, validation, service dispatching
- **AI Service Layer**: Prompt formatting, Watsonx/LangChain interaction
- **Persistence Layer**: Feedback storage, credentials, etc.

---

5. Activity 1.3: Setup of Development Environment

Python Environment

Ensure installation of Python 3.10 and pip.

Virtual Environment Creation

```
python -m venv myenv
myenv\Scripts\activate
```

Library Installation

```
pip install -r requirements.txt
```

Required Libraries: fastapi, streamlit, pymupdf, ibm-watsonx-ai, langchain, uvicorn, python-dotenv

Configuration of Environment Variables

Create `.env` file:

```
WATSONX_API_KEY=...
WATSONX_PROJECT_ID=...
WATSONX_MODEL_ID=granite-20b-code-instruct
```

Folder Structure

```
smart_sdlc_frontend/
app/
├── routes/
├── services/
├── models/
├── utils/
```

---

6. Milestone 2: Core Functionalities Development

AI-Powered Requirement Analysis

Module: `ai_story_generator.py`

- Extracts raw text
- Applies AI classification per SDLC phase
- Generates structured user stories

Code Generation

Module: `code_generator.py`

- Inputs: task descriptions, user stories
- Outputs: Python, Java, JS code snippets

Test Case Generation

- Generated from existing or generated code using AI inference

Bug Detection and Correction

Module: `bug_resolver.py`

- Locates errors
- Suggests or directly patches code

Code Summarization

Module: `doc_generator.py`

- Converts source blocks into high-level documentation

Chatbot Assistant

Modules: `conversation_handler.py`, `chat_routes.py`

- Streamlit-integrated bot powered by LangChain

Feedback System

Modules: `feedback_routes.py`, `feedback_model.py`

- Stores structured feedback for ML fine-tuning

GitHub Integration

Module: `github_service.py`

- Auto-push code
- Raise issues
- Sync docs

---

7. Activity 2.2: Backend Routing and API Integration

API Routing Modules

- `ai_routes.py`: Feature endpoints
- `auth_routes.py`: Auth
- `chat_routes.py`: Chatbot
- `feedback_routes.py`: Feedback system

Authentication Module

- bcrypt password hashing
- Session-based validation

Service Layer Structure

All business logic handled in `services/`, such as:

- `pdf_processor_service.py`
- `watsonx_service.py`

AI Integration Flow

- Watsonx and LangChain APIs
- Prompt templates used to standardize queries

---

8. Milestone 3: Main Application Controller (main.py)

Include Core Routers

```
app.include_router(ai_router, prefix="/ai")
app.include_router(auth_router, prefix="/auth")
app.include_router(chat_router, prefix="/chat")
app.include_router(feedback_router, prefix="/feedback")
```

Middleware Setup

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Metadata and Root

```
app = FastAPI(title="SmartSDLC", version="1.0")
```

---

9. Milestone 4: Streamlit Frontend Development

Homepage Layout (Home.py)

- Header with animation
- Feature tiles in columns

Responsive Design

- Uses Streamlit containers, columns
- Custom CSS for branding

Page Modules

Each feature has its own `.py` file:

- Upload_and_Classify.py
- Code_Generator.py
- Test_Generator.py
- Bug_Fixer.py
- Code_Summarizer.py
- Feedback.py

---

10. Activity 4.2: Dynamic Interaction Between Frontend and Backend

API Client

- Communicates with backend using `requests`

```
response = requests.post("http://localhost:8000/ai/generate-code",
json=payload)
```

Chatbot Integration

- Live conversation with `/chat/chat` endpoint
- Emoji avatars, memory session

---

11. Milestone 5: Local Deployment Strategy

Local Setup

- Setup Python venv
- Install all packages
- Configure `.env`

Running Backend and Frontend

```
uvicorn app.main:app --reload
streamlit run smart_sdlc_frontend/Home.py
```

Access URLs

- Frontend: `http://localhost:8501`
- Backend Docs: `http://127.0.0.1:8000/docs`

---

12. Final Notes and Recommendations

Next Steps

- Containerize using Docker
- Deploy to IBM Cloud or AWS Lambda
- Build analytics dashboard
- Implement role-based access control

SmartSDLC demonstrates how modular microservices and foundation models can automate a traditionally manual and error-prone software lifecycle. The current stack is scalable, pluggable, and suitable for enterprise use with additional fine-tuning and deployment pipelines.