SmartSDLC – AI-Powered Software Development Lifecycle Assistant

1. Introduction

- Project title : SmartSDLC – AI-Powered Software Development Lifecycle Assistant
- Team member : JAYARAJ R
- Team member : RITESH KUMAR PANDEY
- Team member : DINESH
- Team member : ASWIN KUMAR.M

2. Project Overview

- **Purpose :** The purpose of SmartSDLC is to empower software teams by automating and streamlining the Software Development Lifecycle (SDLC) using AI-driven intelligence.

Requirement Classification

Key Point: Automated requirement analysis
Functionality: Classifies raw requirements into SDLC phases and generates structured user stories.

AI Code Generator

Key Point: Production-ready code creation
Functionality: Transforms natural language prompts into executable, well-structured code.

Test Case Generator

Key Point: Automated testing support
Functionality: Creates functional test cases directly from requirements and code.

Bug Fixer

Key Point: AI debugging
Functionality: Analyzes and fixes syntactical and logical errors in code snippets.

Documentation Assistant

Key Point: Smart documentation
Functionality: Generates technical documentation and summaries automatically.

KPI Forecasting

Key Point: Development planning
Functionality: Provides predictions on project progress and bottlenecks.

Anomaly Detection

Key Point: Error prevention
Functionality: Identifies unusual patterns in code or project data to flag potential risks.

Multimodal Input Support

Key Point: Flexible data handling
Functionality: Accepts PDFs, text, and natural language prompts for processing.

Streamlit Dashboard

Key Point: User-friendly interface
Functionality: Interactive dashboard for developers to upload, generate, and monitor project assets.

3. Architecture

Frontend (Streamlit): Provides an interactive web UI for requirement uploads, code generation, bug fixing, and dashboards.
Backend (FastAPI): Handles API endpoints for requirement analysis, code generation, debugging, and test case creation.
LLM Integration (IBM Watsonx Granite): Used for natural language understanding, code generation, and debugging.
LangChain Integration: Manages prompt orchestration and chaining for different SDLC tasks. **ML Modules:** Lightweight models are used for KPI forecasting and anomaly detection to support project planning.

4. Setup Instructions

Prerequisites*:*

- Python 3.9 or later
- pip and virtual environment tools
- API keys for IBM Watsonx
- Internet access to use cloud services

Installation Process:

- Clone the repository
- Install dependencies from requirements.txt
- Create a .env file and configure credentials
- Run the backend server using FastAPI
- Launch the frontend via Streamlit
- Upload data and interact with the modules

5. Folder Structure

app/ – FastAPI backend logic
app/api/ – Modular API routes like requirements, code, bug fixing, and documentation ui/ – Streamlit frontend components
smart_dashboard.py – Entry script for launching the main Streamlit dashboard watson_llm.py – Handles communication with IBM Watsonx Granite model document_processor.py – Extracts and processes uploaded documents kpi_forecaster.py – Forecasts project KPIs
anomaly_checker.py – Detects anomalies in SDLC processes report_generator.py – Generates AI-based project documentation

6. Running the Application

➢ Launch the FastAPI server to expose backend endpoints.
➢ Run the Streamlit dashboard to access the web interface.
➢ Upload requirement documents or code snippets for processing.
➢ Generate structured requirements, code, bug fixes, and documentation.
➢ Use real-time APIs to update outputs dynamically on the frontend.

7. API Documentation

POST /requirements/classify – Uploads requirements and classifies into SDLC phases POST /code/generate – Generates code from natural language prompts
POST /bug/fix – Analyzes and fixes buggy code
POST /testcases/generate – Creates test cases from requirements POST /docs/generate – Produces technical documentation

8. Authentication

Secure deployments can integrate:
- Token-based authentication (JWT or API keys)
- OAuth2 with IBM Cloud credentials
- Role-based access (admin, developer, tester)
- User sessions and history tracking

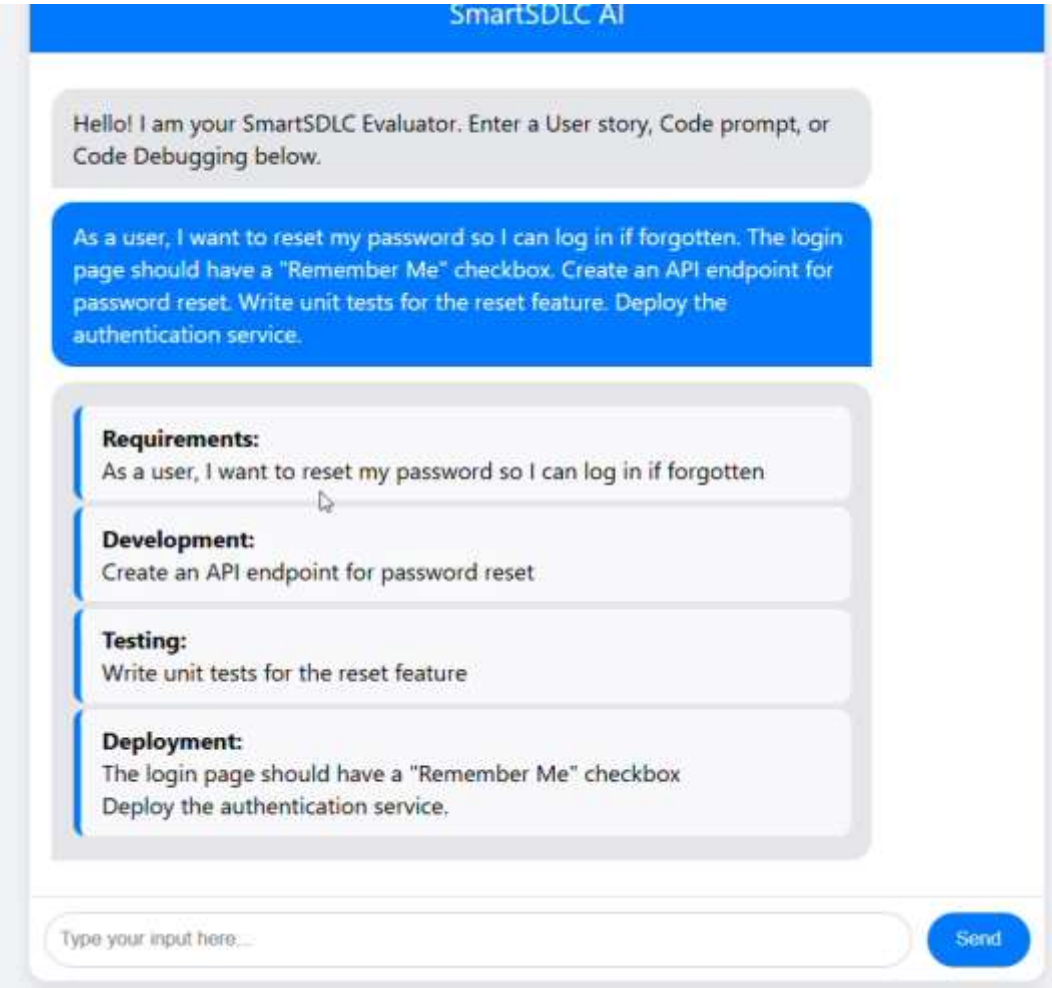
9. User Interface

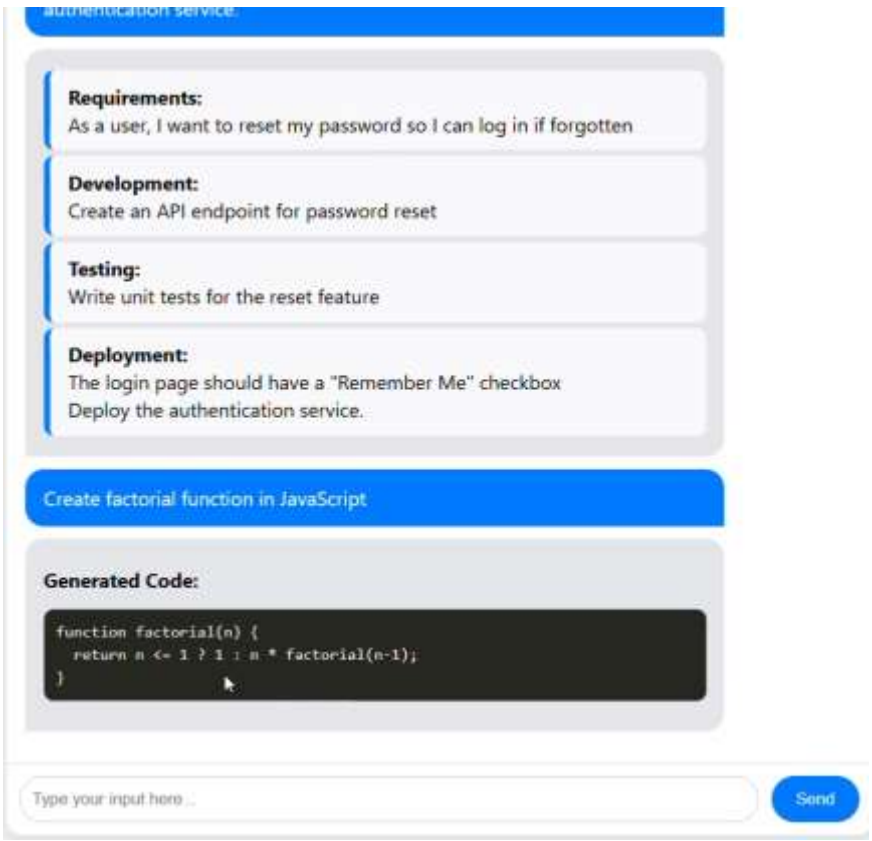
The interface provides a clean and developer-friendly dashboard with tabs for requirements, code generation, bug fixing, and reports. It supports uploads, real-time updates, and PDF exports.

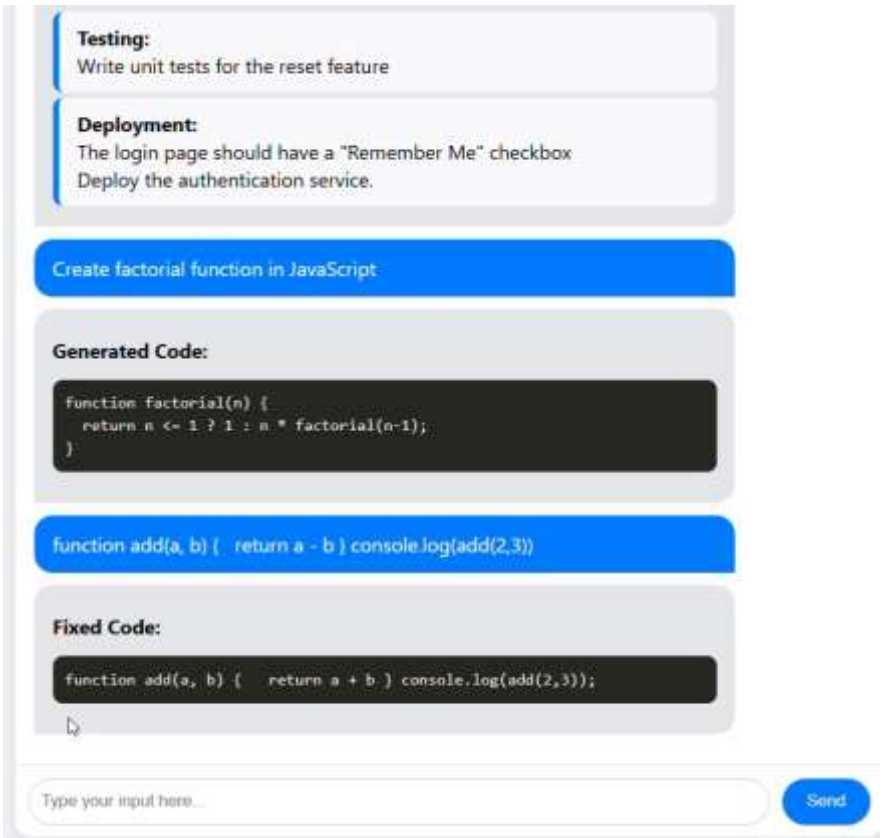
10. Testing

Unit Testing: For utility and AI prompt functions API Testing: Swagger UI, Postman
Manual Testing: Requirement uploads, code generation, and debugging Edge Case Handling: Malformed inputs, large files, invalid API keys

11. Screenshots




authentication service
Requirements:
As a user, I want to reset my password so I can log in if forgotten
Development:
Create an API endpoint for password reset
Testing:
Write unit tests for the reset feature
Deployment:
The login page should have a "Remember Me" checkbox
Deploy the authentication service.
Create factorial function in JavaScript
Generated Code:
function factorial(n) {
return n <= 1 ? 1 : n * factorial(n-1);
}
Type your input here...
Send


Testing:
Write unit tests for the reset feature
Deployment:
The login page should have a "Remember Me" checkbox
Deploy the authentication service.
Create factorial function in JavaScript
Generated Code:
function factorial(n) {
return n <= 1 ? 1 : n * factorial(n-1);
}
function add(a, b) { return a - b } console.log(add(2,3))
Fixed Code:
function add(a, b) { return a + b } console.log(add(2,3));
Type your input here...
Send

12. Known Issues

- AI-generated code may sometimes require manual adjustments for edge cases or project-specific libraries.
- Classification of requirements may produce overlaps if inputs are ambiguous or poorly structured.
- Large PDF uploads can slow down processing due to extraction and classification overhead.
- Bug fixer may not handle very complex logical errors that require deep domain expertise.
- Limited multi-language support (currently optimized for Python and JavaScript only).
- Requires stable internet connection for Watsonx API calls; offline functionality is limited.
- Model performance depends on the quality of training and updates from IBM Watsonx Granite.

13. Future Enhancements

- **Multi-language Support:** Extend code generation and bug fixing to more languages (Java, C++, Go, etc.).
- **Advanced Test Automation:** Integrate AI-driven regression and performance testing.
- **CI/CD Integration:** Seamless integration with GitHub Actions, Jenkins, or GitLab pipelines.
- **Requirement Traceability Matrix:** Automated linking between requirements, code, and test cases.
- **Collaborative Features:** Multi-user workspace with real-time edits and AI-assisted reviews.
- **Cloud Deployment:** Containerized deployment with Kubernetes and IBM Cloud for enterprise scalability.
- **Security Enhancements:** AI-driven vulnerability detection in generated code.
- **Offline Mode:** Limited offline functionality for requirement classification and code generation.
- **Enhanced Visualization:** Interactive analytics dashboards for project health and progress monitoring.
- **Custom Model Fine-tuning:** Allow teams to fine-tune Watsonx models with project-specific data.