Generative AI with IBM

SmartSDLC – AI-Enhanced Software Development Lifecycle

Project Documentation

1.Introduction :

- **Project Title : SmartSDLC AI-Enhanced Software Development Lifecycle**
- **Team ID : NM2025TMID05494**
- **Team member : P.Dhanalakshmi**
- **Team member : M.Fowjana**
- **Team member : J.Dharshini**
- **Team member : J.Janiffer**

2.Project Overview :

The Smart SDLC project revolutionizes traditional software development by integrating Artificial Intelligence (AI) into every stage of the Software Development Lifecycle (SDLC). This innovative approach enhances efficiency, reduces manual effort, and improves software quality.

- **Purpose:**

To empower development teams to build high-quality software faster and more efficiently, while reducing costs and improving overall productivity.

- **Features of SmartSDLC:**

1. **AI-driven Requirement Analysis:**

→ Key Point: Improved requirement understanding and reduced ambiguity.

→ ***Functionality: Analyzes requirements, identifies potential issues, and provides insights.***

2. Automated Code Generation:

→ ***Key Point: Increased productivity and reduced coding errors.***

→ ***Functionality: Generates code snippets, completes partial code, and suggests optimization.***

3.AI-powered Testing:

→ ***key Point: Improved test coverage and reduced testing time***

→ **Functionality: Automates testing, identifies potential issues, and provides test coverage analysis**

4.Smart Deployment:

→ **Key Point: Faster and more reliable deployment.**

→ **Functionality: Optimizes deployment processes, reduces downtime, and enables quick rollbacks.**

5. Predictive Maintenance:

→ ***Key Point: Proactive issue detection and prevention.***

→ ***Functionality: Predicts potential issues, monitors performance, and provides alerts and recommendations.***

3.Architecture :

Frontend (React):

The frontend is built with React, offering a dynamic and responsive web UI with multiple pages including project dashboards, AI-driven code suggestions, collaboration tools, feedback mechanisms, and report viewers. Navigation is handled through a sidebar for intuitive user experience. Each page is modularized for scalability and maintainability.

Backend (Fast API):

Fast API serves as the backend REST framework that powers API endpoints for AI-enhanced code analysis, automated testing, deployment management, collaboration interactions, report creation, and model training. It is optimized for asynchronous performance and easy Swagger integration for API documentation.

4.Setup instructions:

Prerequisites:

- ***Python 3.9 or later***
- ***pip and virtual environment tools***
- ***AI/ML libraries (e.g., TensorFlow, PyTorch)***
- ***API keys for AI models (if applicable)***
- ***Code editor or IDE***

Installation process:

- ***Clone the repository: git clone https://github.com/[username]/smartSDLC.git***
- ***Create a virtual environment: python -m venv smartSDLC-env***
- ***Activate the virtual environment***
- ***Install dependencies: pip install -r requirements.txt***
- ***Configure API keys and settings***

- ***Run the application: python app.py***

5. Folder Structure :

```
smartSDLC/
├── app/
│   ├── __init__.py
│   ├── models/
│   │   ├── __init__.py
│   │   ├── requirement_analysis.py
│   │   ├── code_generation.py
│   │   └── ...
│   ├── services/
│   │   ├── __init__.py
│   │   ├── ai_service.py
│   │   └── ...
│   ├── utils/
│   │   ├── __init__.py
│   │   ├── helpers. py
│   │   └── ...
│   └── app.py
├── config/
│   ├── __init__.py
│   └── settings.py
├── data/
│   ├── requirements/
│   ├── code/
│   └── ...
├── docs/
│   ├── README.md
│   └── ...
├── tests/
│   ├── __init__.py
│   ├── test_requirement_analysis.py
│   ├── test_code_generation.py
```

```
│   └── ...
├── requirements.txt
├── .gitignore
└── README.md
```

This structure includes:

- ***app/: Application code, including models, services, and utilities.***
- ***config/: Configuration files, including settings and API keys.***
- ***data/: Data storage for requirements, code, and other project data.***
- ***docs/: Documentation, including README files and other project documentation.***
- ***tests/: Unit tests and integration tests for the application.***
- ***requirements.txt: Dependencies required for the project.***
- ***.gitignore: Files and directories to ignore in the Git repository.***

6. Running the Application:

To start the project:

- **Launch the FastAPI server to expose backend endpoints.**
- **Run the Streamlit dashboard to access the web interface.**
- **Navigate through pages via the sidebar**
- **Upload documents or interact with the AI assistant, and view outputs like code suggestions, collaboration tools, and reports.**

Frontend (Streamlit):

The frontend is built with Streamlit, offering an interactive web UI with multiple pages including project dashboards, code uploads, AI-driven suggestions, collaboration tools, and report viewers. Navigation is handled through a sidebar.

Backend (Fast API):

Fast API serves as the backend REST framework that powers API endpoints for AI-enhanced code analysis, automated testing, deployment

management, collaboration, and report creation. It is optimized for asynchronous performance and easy Swagger integration.

7. API Documentation:

Backend APIs available include:

- **POST /code-suggest – Accepts code snippets and responds with AI-enhanced suggestions.**
- **POST /upload-code – Uploads and processes code for analysis.**
- **GET /search-code – Returns semantically similar code patterns to the input query.**
- **POST /submit-feedback – Stores user feedback for later review.**

Each endpoint is tested and documented in Swagger.

8. Authentication:

This version of SmartSDLC runs in an open environment for demonstration. Secure deployments can integrate :

- **Token-based authentication (JWT or API keys)**
- **OAuth2 with third-party credential**
- **Role-based access (developer, tester, admin)**
- **Planned enhancements include user sessions and history tracking.**

9. User Interface:

The interface is optimized for developers with features like code editors, AI suggestions, collaboration tools, and report viewers for efficient AI-enhanced software development.

10.Testing:

Testing for SmartSDLC AI-Enhanced Software Development was done in multiple phases:

- **Unit Testing: For prompt engineering functions and utility scripts.**
- **API Testing: Via Swagger UI, Postman, and test scripts.**

- **Manual Testing: For file uploads, chat responses, and output consistency.**
- **Edge Case Handling: Malformed inputs, large files, invalid API keys.**

Each function was validated to ensure reliability in both offline and API-connected modes.

11.Screenshots:

Screenshots of the SmartSDLC AI-Enhanced Software Development interface can showcase:

- **Dashboard views with project metrics.**
- **Code upload and AI-driven suggestion interfaces.**
- **Collaboration tools and report viewers.**

12.Known issues:

Known issues in SmartSDLC AI-Enhanced Software Development might include:

- **Handling very large codebases for analysis.**
- **Optimizing AI suggestion accuracy for specific languages.**
- **UI responsiveness on lower-end devices.**

13. Future Enhancements:

Future enhancements for SmartSDLC AI-Enhanced Software Development could include:

- **Integration of more advanced AI models for code analysis.**
- **Adding support for more programming languages.**
- **Enhanced user authentication and session tracking.**