### **Project Documentation: SmartSDLC**

#### **1. Introduction**

**Project Title:** SmartSDLC: AI-Enhanced Software Development Lifecycle using IBM Granite

**Project Team:**

**Naan Mudhalavan Project Id : NM2025TMID03615**

* **Team Leader:** SHARMILA R
* **Team Members:** SHAILAJA S A
* **Team Members:** SRIJA R
* **Team Members:** ASIN C B

#### 

#### **2. Project Overview**

**Purpose:** The SmartSDLC initiative is engineered to optimize and accelerate key phases of the Software Development Lifecycle through the integration of advanced Artificial Intelligence. The platform equips development teams, project managers, and stakeholders with a sophisticated toolset that transforms high-level linguistic requirements into executable artifacts, including source code, unit tests, bug remediations, and comprehensive documentation, all facilitated through a highly interactive and intuitive user interface.

**Core Features:**

* **Automated Requirement Ingestion:** Semantic analysis of uploaded software requirement specifications from PDF and Word documents.
* **Code Synthesis:** On-demand generation of backend source code from natural language specifications.
* **Intelligent Bug Remediation:** Automated detection of anomalies in code with AI-powered correction suggestions.
* **Test Case Automation:** Generation and exportation of unit test cases to ensure code quality and robustness.
* **Automated Documentation:** Algorithmic summarization of source code to assist in the documentation process.
* **Conversational AI Support:** An integrated chatbot assistant, providing expert support and query resolution across all SDLC phases.

#### 

#### 

#### 

#### **3. System Architecture**

The platform is designed with a decoupled, service-oriented architecture to ensure scalability and maintainability.

* **Frontend (Presentation Layer):** The user interface is constructed with Gradio, providing a reactive and user-centric web interface. This layer facilitates all user interactions, including file uploads, text-based queries, visualization of generated outputs (code, tests, documentation), and engagement with the IBM Watson Assistant-powered chatbot.
* **Backend (Service Layer):** Implemented using Python with the FastAPI framework, this asynchronous, high-performance layer serves as the application's core. It is responsible for request routing, orchestration of AI model interactions (via Watsonx.ai and Hugging Face APIs), persistent file management, and user session control.
* **Database (Persistence Layer):** The system utilizes SQLite for development environments and offers support for enterprise-grade databases like MySQL or PostgreSQL for production deployments. Data persistence is managed through SQLAlchemy, an Object-Relational Mapper (ORM) that provides a robust abstraction layer for all database operations, including the storage of user profiles, file metadata, generated artifacts, and system logs.

#### 

#### **4. Environment Setup and Configuration**

**Prerequisites:**

* Python environment, version 3.9 or higher.
* An active relational database instance (SQLite, MySQL, or PostgreSQL).
* Valid IBM Cloud service credentials for accessing Watson Assistant and Watsonx.ai APIs.

**Installation Protocol:**

1. **Clone the Source Repository:** git clone <repository-url>
2. **Navigate to Project Directory:** Change into the newly cloned project folder.
3. **Establish a Virtual Environment:** python -m venv venv && source venv/bin/activate (for Unix-based systems).
4. **Install Dependencies:** Execute pip install -r requirements.txt to install all required libraries.
5. **Configure Environment:** Create a .env file in the root directory to store sensitive credentials and configuration variables (e.g., IBM\_API\_KEY, DB\_URL).

#### **5. Application Screenshots**

The user interface provides a clean, modular experience for the primary functions of analysis and generation.

**Code Analysis in Action** The **Code Analysis** module ingests a natural language prompt (e.g., "create a code for a general login page") and automatically structures it into detailed Functional Requirements, Non-Functional Requirements, and Technical Specifications.

**Code Generation in Action** The **Code Generation** module takes the user's requirement and a specified programming language (e.g., Python) to synthesize a functional code snippet, such as a basic web server route for handling logins.

#### 

#### **6. API Endpoints**

The backend exposes a RESTful API for client-server communication.

|  |  |  |  |
| --- | --- | --- | --- |
| Endpoint | Method | Request Parameters | Response |
| /register | POST | email, password | Success/status message |
| /confirm-email | GET | token | Account activation confirmation |
| /login | POST | email, password | Auth token + profile |
| /upload | POST | file | Extraction summary, file ID |
| /generate-code | POST | requirement ID, language | Generated backend code |
| /test-cases | POST | code | Unit test cases output |
| /bug-fix | POST | buggy code | Suggested bug fixes |
| /summarize-code | POST | code | Code summary |
| /chatbot | POST | question | AI-powered SDLC support answer |
|  |  |  |  |

**7.Output screenshots**

#### 

#### 

#### 

#### **8. Authentication and Security**

The application implements a robust authentication mechanism based on JSON Web Tokens (JWT). Upon successful login, a JWT is issued to the client. This token must be included in the headers of subsequent requests to protected endpoints. A middleware layer is responsible for validating the token's integrity and expiration status, which is set to one hour, with refresh capabilities enabled for extended user sessions.

#### **9. Future Enhancements Roadmap**

* **Polyglot Expansion:** Extend code generation capabilities to support additional programming languages such as Java and Node.js.
* **CI/CD Integration:** Implement a continuous integration and deployment pipeline to automate testing and deployment workflows.
* **Voice-Activated Interface:** Integrate Speech-to-Text (STT) and Text-to-Speech (TTS) technologies to enable voice-based interactions with the SDLC chatbot.
* **Administrative Dashboard:** Develop an admin-level dashboard for monitoring system analytics, user activity, and application logs.
* **Model Specialization:** Fine-tune the underlying AI models on domain-specific datasets to improve accuracy and relevance for specialized software projects.

#### **10. Conclusion**

In conclusion, **SmartSDLC** represents a significant step toward the intelligent automation of the software development lifecycle. By leveraging the advanced capabilities of the **IBM Granite** model within a robust, modern architecture, the platform effectively bridges the gap between abstract human requirements and precise, machine-executable logic. It is engineered not merely as a collection of tools, but as an integrated ecosystem designed to enhance developer productivity, minimize manual errors, and ultimately accelerate the delivery of high-quality software. With a solid foundation and a clear roadmap for future enhancements—including polyglot support and CI/CD integration—SmartSDLC is poised to evolve into an indispensable asset for development teams, fostering a more efficient and dynamic synergy between human creativity and artificial intelligence.