**Smart SDLC: AI-Enhanced Software Development Lifecycle Leveraging IBM Granite**

**1. Introduction**

**Project Title:**  
Smart SDLC – AI-Enhanced Software Development Lifecycle Leveraging IBM Granite

**Team Members:**

* Kaviya S. (Team Leader)
* Jessy Evangeline M.
* Jothika R.
* Jesin Mary T.

**2. Project Overview**

**Purpose:**  
Smart SDLC is designed to simplify and automate key phases of the Software Development Lifecycle by leveraging advanced AI technologies. It empowers developers and project managers to seamlessly convert natural language requirements into functional backend code, unit tests, bug fixes, and comprehensive documentation — all through an intuitive and interactive interface.

**Key Features:**

* Upload and analyze requirement documents in PDF or Word format
* AI-driven generation of backend code from natural language prompts
* Intelligent bug detection with automatic suggestions for fixes
* Automated unit test case generation and export capability
* Summarization of code for efficient documentation
* AI-powered chatbot for SDLC guidance and on-demand Q&A

**3. Architecture**

**Frontend:**  
Built using Gradio, offering an interactive user interface where users can upload files, enter queries, view generated code/test outputs, and engage with a chatbot powered by IBM Watson Assistant.

**Backend:**  
Developed with FastAPI in Python, responsible for routing requests, orchestrating AI model workflows (IBM Watsonx & Hugging Face), managing file uploads, and handling user sessions.

**Database:**  
Employs SQLite for development and MySQL/PostgreSQL in production environments using SQLAlchemy to store user data, file histories, generated outputs, and logs.

**4. Setup Instructions**

**Prerequisites:**

* Python 3.9+
* Optional: SQLite or MySQL/PostgreSQL
* IBM Cloud credentials for Watson Assistant & Watsonx.ai

**Installation Steps:**

1. Clone the repository:  
   git clone <repo-url>
2. Navigate into the project directory
3. Create a virtual environment:  
   python -m venv venv && source venv/bin/activate
4. Install dependencies:  
   pip install -r requirements.txt
5. Set up .env file with required credentials (e.g., IBM\_API\_KEY, DB\_URL)

**5. Folder Structure**

* **Client (Gradio UI):**
  + app.py: Main UI entrypoint
  + components/: Reusable UI components (file upload, chatbot, output viewers)
* **Server (Python Backend):**
  + main.py: FastAPI backend application
  + models/: AI-related logic and model wrappers
  + services/: Integration with IBM Watsonx and Hugging Face APIs
  + db/: Database models and ORM logic
  + tests/: Unit tests and API endpoint tests

**6. Running the Application**

* **Frontend:**  
  python app.py
* **Backend:**  
  uvicorn main:app --reload

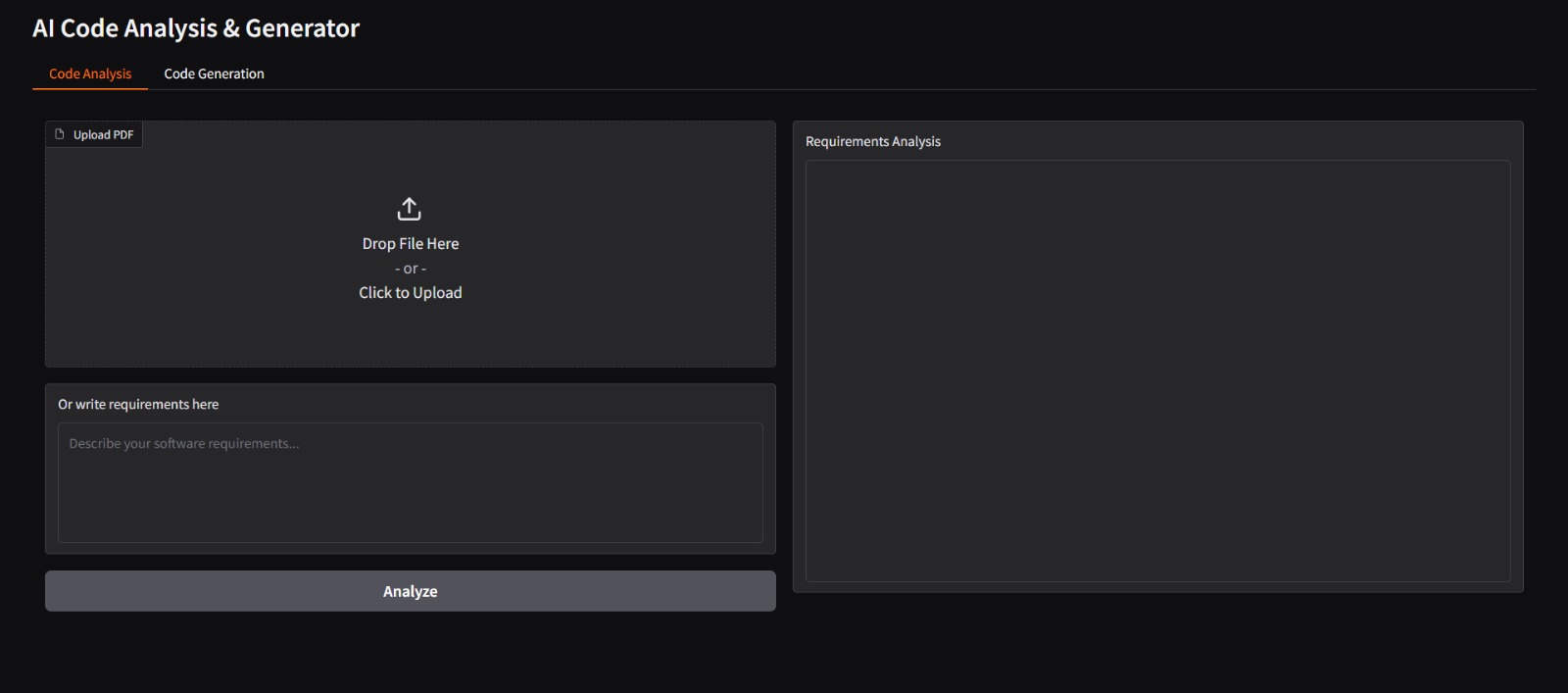
**7. API Documentation**

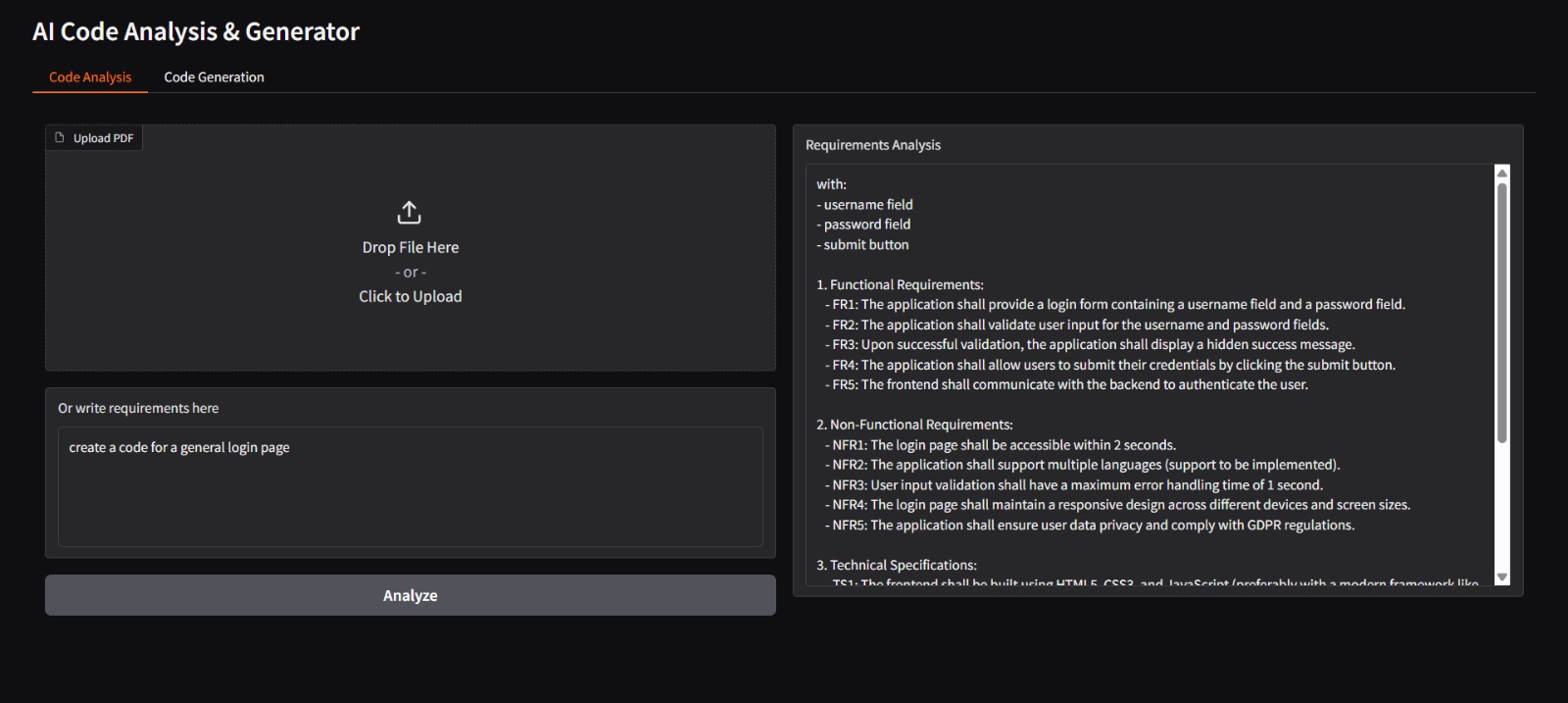
| **Endpoint** | **Method** | **Request Parameters** | **Response** |
| --- | --- | --- | --- |
| /register | POST | email, password | Success/status message |
| /confirm-email | GET | token | Account activation confirmation |
| /login | POST | email, password | Auth token + user profile |
| /upload | 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 |

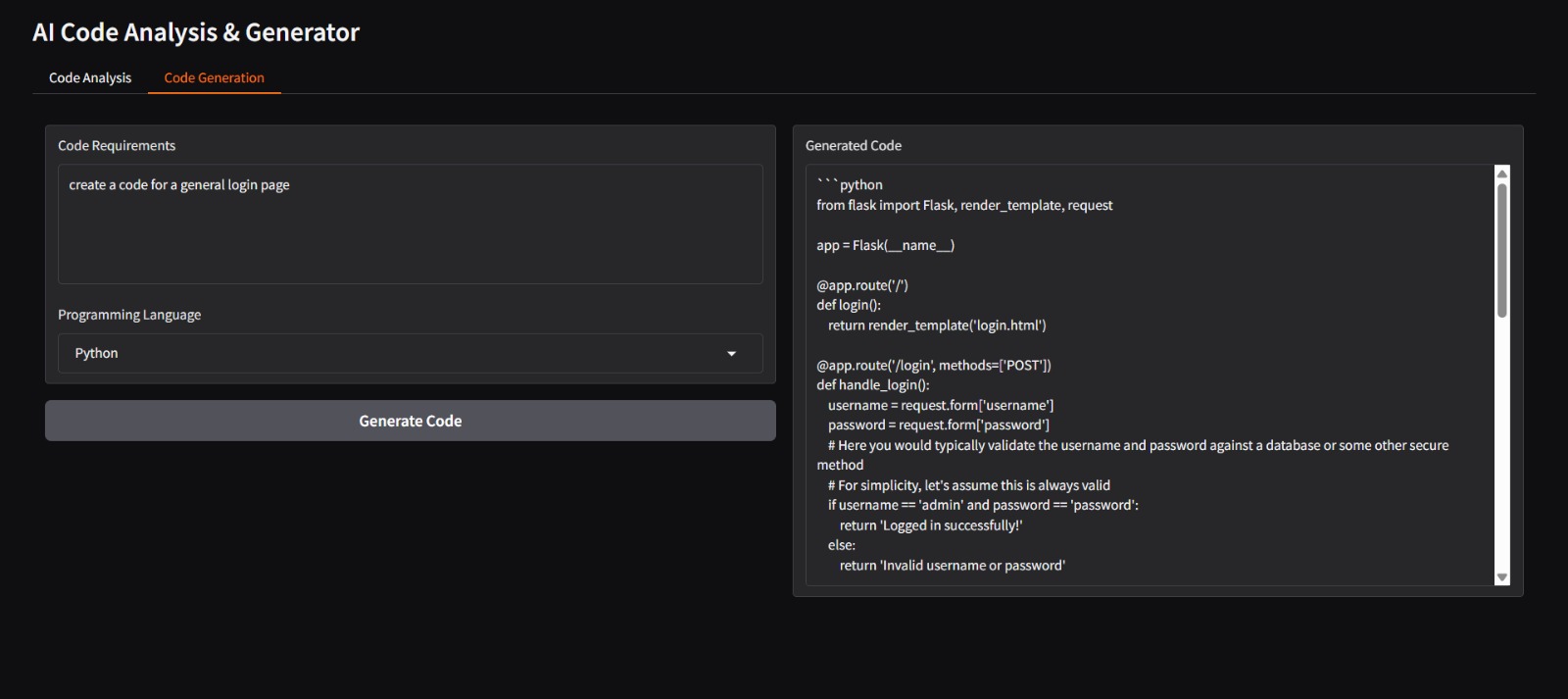
**8. Authentication**

* Implements JWT-based token authentication after login
* Middleware validates tokens for protected routes
* Token expires after 1 hour with refresh support for extended sessions

**9. Screenshots**







**10. Testing**

**Strategy & Tools:**

* Unit testing performed using pytest
* API tested with FastAPI TestClient
* Manual UI testing via Gradio interface

**11. Future Enhancements**

* Multi-language code generation (Java, Node.js)
* CI/CD auto-deploy integration
* Voice-driven chatbot with STT/TTS support
* Admin dashboard with analytics and logs
* Fine-tuning AI models for domain-specific projects

**12. Conclusion**

The **Smart SDLC – AI-Enhanced Software Development Lifecycle** project successfully demonstrates how generative AI technologies can transform traditional software engineering practices. By automating critical SDLC phases such as requirement analysis, code generation, test case creation, bug fixing, and documentation, Smart SDLC significantly reduces manual workload, accelerates development cycles, and enhances overall software quality.

Its modular architecture, interactive UI, and seamless integration with IBM Granite and other AI services provide a practical, scalable solution for developers and project teams. The intuitive design ensures accessibility for both technical and non-technical users, making it an ideal tool for educational purposes, prototyping, and real-world software development.

Looking ahead, planned future enhancements—such as multi-language support, CI/CD integration, and AI fine-tuning for domain-specific projects—will further empower developers, providing a more intelligent, automated, and efficient SDLC platform.