# SmartSDLC – AI-Enhanced Software Development Lifecycle

## Project Documentation

### 1.Introduction

Project Title: **SmartSDLC – AI-Enhanced Software Development Lifecycle**

**Team ID :** NM2025TMID03843

**Team Leader :** ADITHYAN E

**Team member :** Madhavan K

**Team member :** AAROONRASHEETH A

**Team member :** AJAY K

Background & Motivation
The Software Development Life Cycle (SDLC) is a systematic process followed in software engineering to design, develop, and test high-quality software. However, traditional SDLC phases often involve manual tasks such as requirement analysis, bug fixing, and documentation, which can be time-consuming and prone to human error.

With the rise of Generative AI and Large Language Models (LLMs), there is a great opportunity to automate and optimize SDLC activities, improving productivity and reducing costs.

The Smart SDLC Assistant leverages AI models (IBM Granite LLM), machine learning, and natural language processing to accelerate development by automating:
- Requirement analysis
- Code generation in multiple languages
- Bug fixing and code improvement
- Summarization of project documents
- Developer support through a chatbot interface

## 2. Purpose

The primary goal of the Smart SDLC Assistant is to support developers, testers, and project managers by providing an AI-driven environment for common SDLC tasks.

Objectives:
- Automate requirement analysis from documents or user input.
- Generate multi-language source code based on requirements.
- Provide an AI-powered bug fixer to detect and correct coding errors.
- Summarize long documents into concise, actionable insights.
- Enable interactive Q&A through a conversational chatbot.

## 3. Key Features

Requirement Analysis:
- Key Point: Intelligent classification.
- Functionality: Upload a PDF or enter requirements manually, and the assistant extracts functional and non-functional requirements.
- Benefit: Saves time in manual documentation.

Code Generation:
- Key Point: Multi-language support.
- Functionality: Generates clean, structured code in Python, Java, C++, JavaScript, C#, PHP, Go, and Rust.
- Benefit: Accelerates prototyping and reduces development effort.

Bug Fixer:
- Key Point: Debugging automation.
- Functionality: Detects syntax and logical errors in code snippets and provides corrected versions.
- Benefit: Helps new developers quickly resolve issues without manual debugging.

Summarizer:
- Key Point: Information compression.
- Functionality: Summarizes long PDF/text documents into concise insights.
- Benefit: Assists project managers and developers in quickly understanding lengthy documents.

Chatbot:
- Key Point: Conversational AI.
- Functionality: Provides real-time Q&A on SDLC phases, programming concepts, or uploaded documents.
- Benefit: Acts as a virtual assistant for developers.

## 4. System Architecture

Frontend (Gradio UI):
- Built using Gradio, offering a tabbed interface for easy navigation.
- Tabs: Requirement Analysis, Code Generation, Bug Fixer, Summarizer, Chatbot.
- Clean, user-friendly design for both technical and non-technical users.

Backend (Python + Transformers):
- Powered by IBM Granite LLM using Hugging Face Transformers.
- Handles natural language understanding and generation tasks.
- Executes text extraction, summarization, and response generation.

Document Processing:
- PyPDF2 extracts text from PDF documents.
- Supports plain text inputs as well.

## 5. Setup Instructions

Prerequisites:
- Python 3.9+
- pip package manager
- Installed libraries: transformers, torch, gradio, PyPDF2
- Internet connection to fetch pretrained LLM models

Installation Steps:
1. Clone the repository.
2. Install dependencies: pip install -r requirements.txt
3. Run the application: python app.py
4. Access the Gradio web interface at the URL displayed in the terminal.

## 6. Folder Structure

```
smart_sdlc/
├── app.py              # Main Gradio application
├── requirements.txt    # Dependencies
├── docs/               # Documentation files
├── samples/            # Example input PDFs/text
├── modules/
   ├── analyzer.py      # Requirement analysis logic
   ├── generator.py     # Code generation module
   ├── bugfixer.py      # Bug fixing logic
   ├── summarizer.py    # Summarizer functions
   ├── chatbot.py       # Chat assistant functions
```

## 7. Running the Application

1. Launch the Gradio app.
2. Select the appropriate tab:
   - Upload or type requirements → get structured analysis.
   - Provide requirement → generate code in selected language.
   - Paste buggy code → get fixed output.
   - Upload PDF → get summarized output.
   - Chat → ask queries about SDLC or code.
3. View real-time results directly in the UI.

## 8. API Endpoints (Optional Future Extension)

- POST /analyze-requirements → Returns functional & non-functional requirements.
- POST /generate-code → Generates code based on user input.
- POST /fix-code → Returns corrected version of buggy code.
- POST /summarize → Summarizes uploaded text/PDF.
- POST /chat → Returns chatbot response.

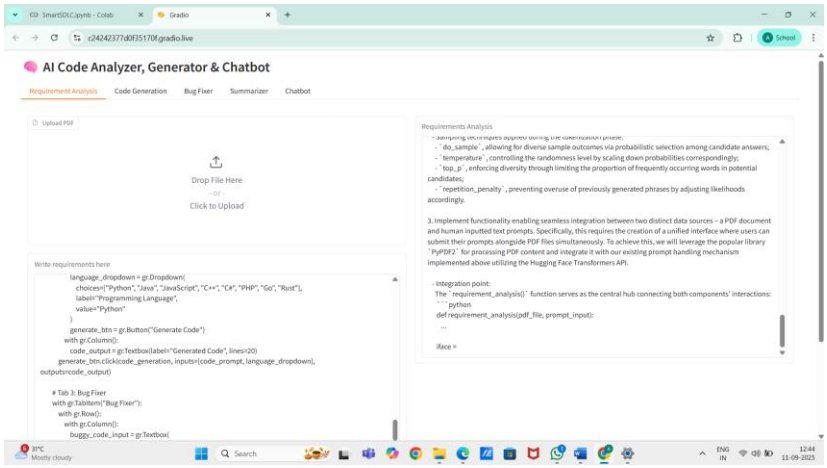
## 9. Security Considerations

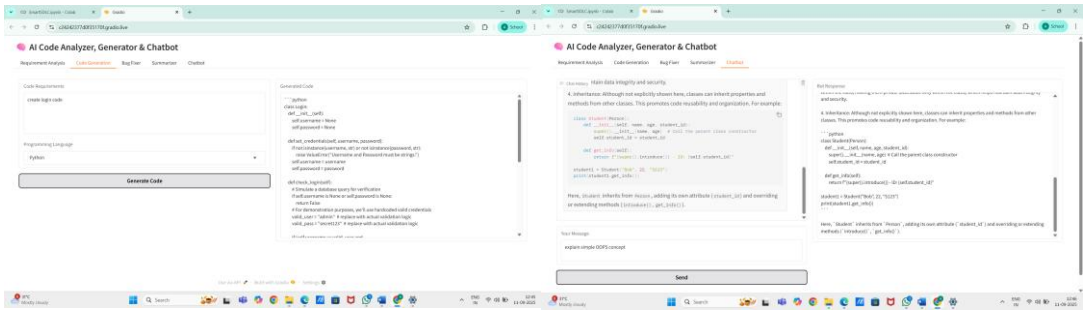
- Current version runs in open environment for testing.
- Future enhancements:
  - JWT-based authentication.
  - Role-based access (Admin, Developer, Tester).
  - API key protection for backend endpoints.

## 10. Testing

- Unit Testing: Validated core modules like requirement extraction and summarizer.
- API Testing: Verified endpoints with Swagger/Postman (planned).
- Manual Testing: Checked file uploads, code generation accuracy, and chatbot responses.
- Edge Cases: Handled large files, malformed inputs, and incorrect code formats.

## 11.Screenshot






## 12. Known Issues

- Sometimes long responses are truncated due to model token limits.
- PDF text extraction may miss formatting or images.
- Requires stable internet connection for model loading.

## 13. Future Enhancements

- Integration with Jira/GitHub for automatic ticket generation.
- Support for test case generation based on requirements.
- Deployment as a cloud API with scalable architecture.
- Adding multi-modal input (images, diagrams).

## 13. Conclusion

The Smart SDLC Assistant showcases how AI can transform traditional software engineering practices by automating requirement analysis, code generation, debugging, and documentation. With further enhancements, it can evolve into a full-fledged development assistant, reducing human effort and accelerating delivery in software projects.

## Project demo video link:

https://drive.google.com/file/d/148kJocHo5LHQFacr9tS9RCRGA17_4R-n/view?usp=drive_link