PROJECT DOCUMENTATION

SMART SDLC

TEAM ID : LTVIP2025TMID37665

CONTENTS

SmartSDLC: An AI-Enhanced Software Development Life Cycle Platform

TEAM ID - LTVIP202TMID37665

1. Introduction

Project Title:
SmartSDLC: An AI-Enhanced Software Development Life Cycle Platform

Team Members:

- Palavalsa Sai Tarun (Team Leader)
- Eswar Khandavali
- Greeshma Gudla
- Dharmana Gowrav Munindra

2. Project Overview

Purpose:
SmartSDLC is an AI-powered assistant designed to automate and streamline key phases of the Software Development Life Cycle (SDLC). Leveraging a large language model (IBM Granite), the platform enhances developer productivity by automating tasks such as requirement analysis, design documentation, code generation, testing, bug fixing, and providing conversational support. This enables developers to focus on solving complex, real-world problems while reducing time spent on repetitive or boilerplate tasks.

Key Features:

- **Requirement Analysis:** Extracts and classifies functional and non-functional requirements from uploaded documents and user prompts.
- **Design Generation:** Produces design documents, UML diagrams (in PlantUML format), and concise system summaries based on user descriptions.
- **Code Generation:** Generates clean, well-commented code in multiple languages (Python, JavaScript, Java) from natural language requirements.
- **Code Explanation:** Explains code snippets in detail for educational and documentation purposes.
- **Testing:** Creates comprehensive unit test cases and detects/fixes bugs in user-submitted code.

- **AI Chatbot Assistant:** Provides conversational support for SDLC, programming, and best practices.
- **Prompt History:** Maintains a session-based history of user prompts and AI responses for easy reference.

3. Architecture

SmartSDLC follows a modern client-server architecture:

- **Frontend:**
  - Built with Streamlit for a fast, interactive, and user-friendly dashboard.
  - Allows users to upload files, enter prompts, select tasks, and view AI-generated outputs.
  - Communicates with the backend via HTTP API calls.
- **Backend:**
  - Developed using FastAPI, exposing RESTful endpoints for all functionalities.
  - Handles file uploads, prompt processing, and interaction with the AI model.
  - Implements prompt engineering to tailor model queries for each SDLC task.
- **AI Model:**
  - Utilizes the IBM Granite large language model (downloaded from Hugging Face and loaded locally).
  - Backend constructs detailed, task-specific prompts for the model to maximize output quality.
- **Database (optional):**
  - A lightweight, file-based database (e.g., SQLite) can be used to store session history or logs.

4. Setup Instructions

Prerequisites

- Python 3.8+ and pip
- Git
- Streamlit and FastAPI
- (Optional) Virtual environment tool (`venv` or `conda`)

Installation

1. **Clone the Repository**

```
git clone https://github.com/yourusername/smart-sdlc.git
cd smart-sdlc
```

2. **Download the AI Model**

   - Download the IBM Granite model (e.g., `granite-3.3-2b-instruct-Q4_K_M.gguf`) from Hugging Face or your storage location.
   - Place it in `backend/models/`.

3. **Setup Backend**

```
cd backend
python -m venv venv
# On Windows:
venv\Scripts\activate
# On Linux/Mac:
source venv/bin/activate
pip install -r requirements.txt
```

4. **Setup Frontend**

```
cd ../frontend
pip install -r requirements.txt
```

5. **Environment Variables**

   - Create a `.env` file in `backend/` and add:

```
MODEL_PATH=./models/granite-3.3-2b-instruct-Q4_K_M.gguf
```

5. Folder Structure

```
smart-sdlc/
│
├── backend/
│   ├── main.py
```

```
│    ├─── llm_utils.py
│    ├─── pdf_utils.py
│    ├─── schemas.py
│    ├─── models/
│    │    └─── granite-3.3-2b-instruct-Q4_K_M.gguf
│    ├─── requirements.txt
│    └─── .env
│
├─── frontend/
│    ├─── app.py
│    ├─── ui_utils.py
│    ├─── requirements.txt
│
├─── .gitignore
├─── README.md
└─── Project Statement.pdf
```

6. Running the Application

Start the Backend:

```
cd backend
uvicorn main:app --reload
```

Start the Frontend:

```
cd ../frontend
streamlit run app.py
```

Access the Application:

- The Streamlit frontend will open in your browser (default: `http://localhost:8501`).
- The FastAPI backend docs are available at `http://localhost:8000/docs`.

7. Model Documentation & Interaction Flow

- **Model:** IBM Granite (e.g., `granite-3.3-2b-instruct-Q4_K_M.gguf`)
- **Interaction:**

- The frontend sends user prompts and task details to the backend via HTTP POST.
- The backend engineers a task-specific prompt and queries the local Granite model.
- The result is formatted and returned to the frontend for display.

- **Example API Request:**

```
{
  "prompt": "Generate a Python class for a To-Do item.",
  "task": "generate_code"
}
```

- **Example API Response:**

```
{
  "status": "success",
  "data": {
    "output_text": "class ToDoItem:\n    def __init__(self, description,
completed=False):\n        self.description = description\n        self.completed =
completed\n..."
  }
}
```

8. User Interface

- **Simple, intuitive dashboard** with:
  - Sidebar navigation for SDLC phases: Requirement Analysis, Design, Coding, Testing, Chatbot
  - Main area for file uploads, prompt input, and output display
  - Prompt history for session-based tracking

9. Testing

Testing Strategy:

- **Backend:**
  - Unit tests for API endpoints and core logic (using `pytest`)
- **AI Output:**
  - Manual testing of prompt variety and response quality

- **Frontend:**
    - Manual cross-browser and device testing for UI consistency

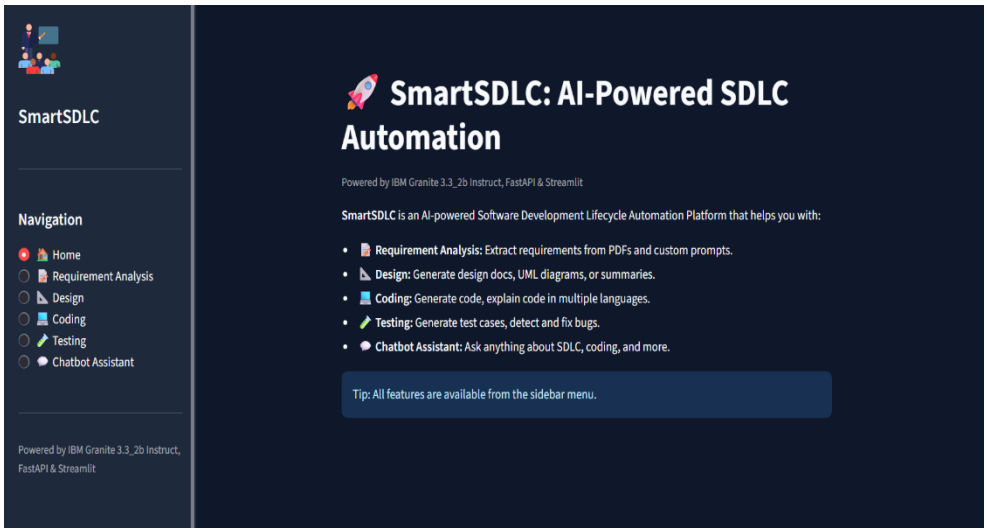
10. Known Issues

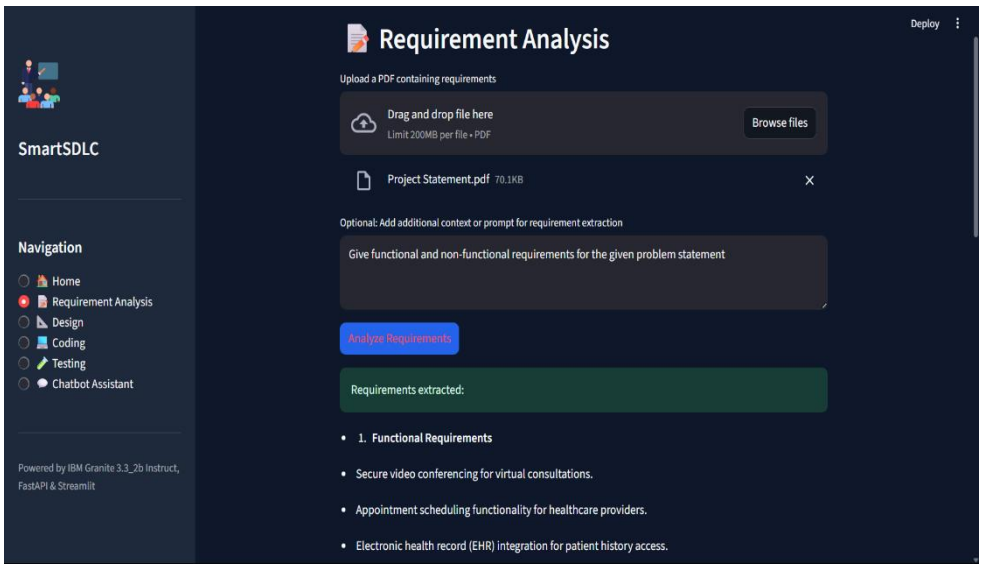
- **Model Hallucinations:**
  The AI model may occasionally produce inaccurate or suboptimal code/text. Always review outputs.
- **Processing Time:**
  Large or complex prompts may result in longer response times.
- **Context Limit:**
  The model has a finite context window; very long conversations may lose earlier context.

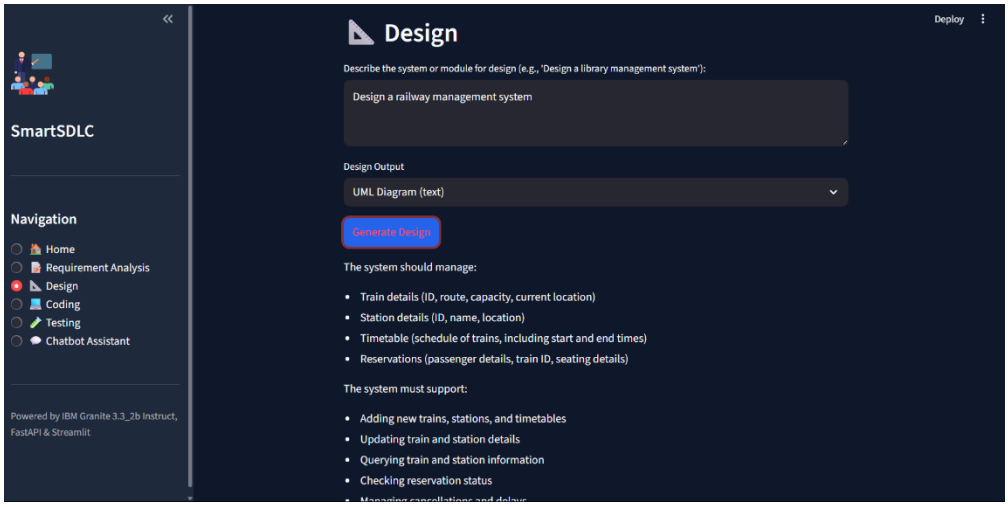
11. Future Enhancements

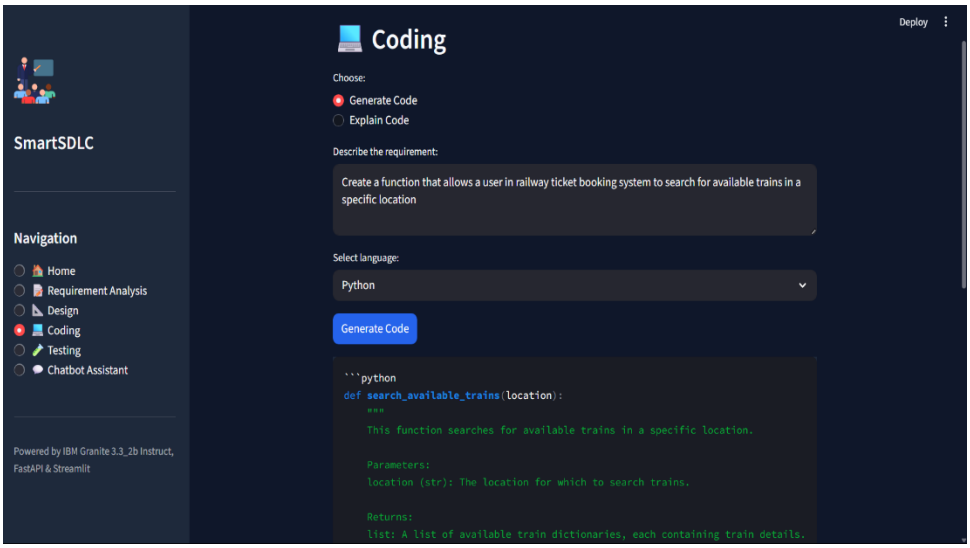
- User authentication and persistent history
- Multi-language code generation (JavaScript, Java, Go, etc.)
- Version control integration (e.g., GitHub/GitLab commit support)
- In-browser code editor with syntax highlighting
- Model fine-tuning on curated datasets
- Collaboration features for teams

12. Screenshots or Demo



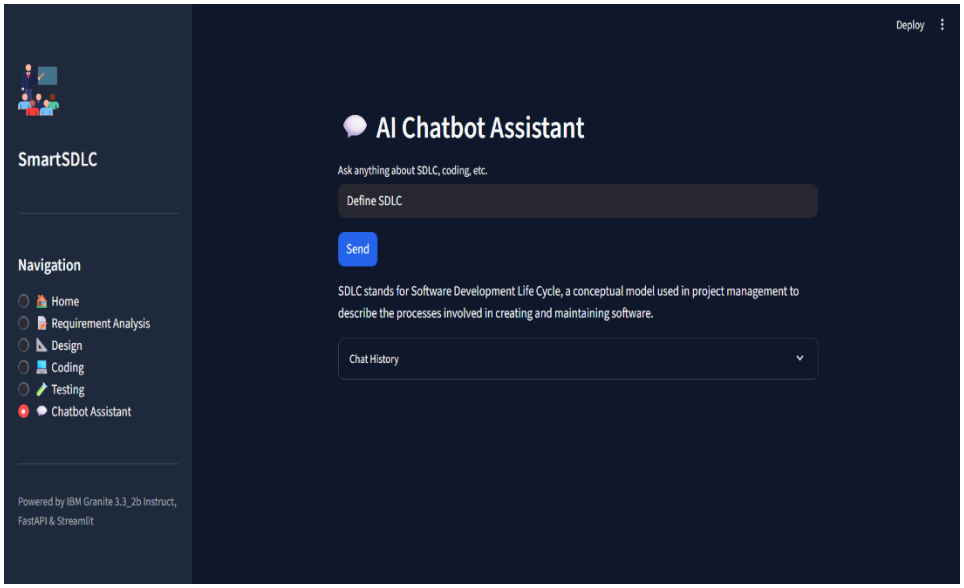

Deploy
Requirement Analysis
Upload a PDF containing requirements
Drag and drop file here
Limit 200MB per file • PDF
Browse files
Project Statement.pdf 70.1KB
Optional: Add additional context or prompt for requirement extraction
Give functional and non-functional requirements for the given problem statement
Analyze Requirements
Requirements extracted:
1. Functional Requirements
Secure video conferencing for virtual consultations.
Appointment scheduling functionality for healthcare providers.
Electronic health record (EHR) integration for patient history access.
SmartSDLC
Navigation
Home
Requirement Analysis
Design
Coding
Testing
Chatbot Assistant
Powered by IBM Granite 3.3_2b Instruct, FastAPI & Streamlit


Deploy
Design
Describe the system or module for design (e.g., 'Design a library management system'):
Design a railway management system
Design Output
UML Diagram (text)
Generate Design
The system should manage:
Train details (ID, route, capacity, current location)
Station details (ID, name, location)
Timetable (schedule of trains, including start and end times)
Reservations (passenger details, train ID, seating details)
The system must support:
Adding new trains, stations, and timetables
Updating train and station details
Querying train and station information
Checking reservation status
SmartSDLC
Navigation
Home
Requirement Analysis
Design
Coding
Testing
Chatbot Assistant
Powered by IBM Granite 3.3_2b Instruct, FastAPI & Streamlit


Deploy
Coding
SmartSDLC
Navigation
Home
Requirement Analysis
Design
Coding
Testing
Chatbot Assistant
Powered by IBM Granite 3.3_2b Instruct, FastAPI & Streamlit
Choose:
Generate Code
Explain Code
Describe the requirement:
Create a function that allows a user in railway ticket booking system to search for available trains in a specific location
Select language:
Python
Generate Code
```python
def search_available_trains(location):
    """
    This function searches for available trains in a specific location.

    Parameters:
    location (str): The location for which to search trains.

    Returns:
    list: A list of available train dictionaries, each containing train details.


Deploy
SmartSDLC
Navigation
Home
Requirement Analysis
Design
Coding
Testing
Chatbot Assistant
Powered by IBM Granite 3.3_2b Instruct, FastAPI & Streamlit
AI Chatbot Assistant
Ask anything about SDLC, coding, etc.
Define SDLC
Send
SDLC stands for Software Development Life Cycle, a conceptual model used in project management to describe the processes involved in creating and maintaining software.
Chat History
```

13. License

This project is for educational purposes only.
No part of this code may be used, copied, or distributed without explicit permission from the authors.

14. Contact

For questions or contributions, please contact the team via GitHub issues or [palavalasasaitarun.22.csm@anits.edu.in].

⁂