SmartSDLC Project Documentation

Introduction

- **Project Title**: SmartSDLC – AI-Powered Software Development Lifecycle Optimization
- **Team ID :** LTVIP2025TMID21159
- **Team Size :** 4
- **Team Members**:
  - Rajulapati Tejasri
  - Pedasingu Sanjana
  - Penneru Mokshagna Naga Vamsi
  - Pedasingu Raja Kumari

Project Overview: SmartSDLC

SmartSDLC is an AI-powered platform designed to optimize the Software Development Lifecycle (SDLC) using **NLP and generative AI**. It automates critical phases including:

- **Requirement classification**
- **Code generation**
- **Bug fixing**
- **Test case creation**
- **Code summarization**
- **SDLC chatbot assistance**

Tech Stack & Requirements

Your project relies on the following technologies:

AI & NLP

- `transformers`, `torch`, `huggingface_hub` — for generative models (e.g., IBM Granite 3.3)
- `dotenv` — for securely loading Hugging Face API tokens

Document Parsing

- `PyMuPDF` — extracts text from PDFs for requirement classification

Backend API

- `FastAPI` — provides endpoints for all the features
- `uvicorn` — runs the FastAPI server
- `requests` — used to call external APIs (fallback: Groq)

Frontend

- `streamlit`, `streamlit-lottie` — interactive UI with visual styling

Modules and Functionality

Here's what each of your modules does:

`pdf_parser.py`

- Extracts text from uploaded PDFs
- Uses `generate()` to classify sentences into SDLC phases using LLMs

`code_generator.py`

- Generates Python code from text requirements

`bug_fixer.py`

- Identifies and fixes bugs in provided Python code

`test_case_generator.py`

- Generates `pytest`-style test cases for a given code snippet

`summarizer.py`

- Summarizes the intent and functionality of a code block

`chatbot.py`

- Responds to SDLC-related queries in a conversational manner

`ai_engine.py`

- Loads IBM Granite model from Hugging Face
- Authenticates using `.env` token
- Defines a unified `generate()` method for all features

`main.py`

- FastAPI server defining endpoints for all six features
- Uses fallback to Groq API if local model fails or GPU is unavailable

`streamlit_ui.py`

- Streamlit web interface for all features
- Allows users to interact via text inputs or PDF uploads

Suggestions / Enhancements

1. **Error Handling Improvements**
   - You can add better structured JSON error responses in `main.py`.
2. **Security**
   - Move the `GROQ_API_KEY` out of code and into `.env`.
3. **Concurrency**
   - Use `asyncio.create_task()` in `main.py` to improve async performance.
4. **Add Logging**
   - Use `logging` for tracing issues instead of raw exception strings.
5. **GPU Detection UI**
   - Show GPU status in the Streamlit UI using `torch.cuda.is_available()`.

Architecture

- **Frontend**:
  Built with Streamlit (for simplicity and rapid prototyping), enhanced with Lottie animations.
- **Backend**:
  FastAPI server powering all feature endpoints (requirement parsing, code generation, bug fixing, etc.).
- **AI/NLP**:
  Uses transformers (IBM Granite 3.3 via Hugging Face) with fallback to Groq API.
- **Database**:
  [Optional: Add if you plan to persist data (e.g., user inputs, logs). Currently none.]

Setup Instructions

- **Prerequisites**:
  - Python 3.10+
  - pip
  - Access tokens for Hugging Face / Groq
- **Installation**:

```
bash
CopyEdit
git clone [repo-url]
cd SmartSDLC
pip install -r requirements.txt
Create a .env file and add API keys
```

Folder Structure

- **pdf_parser.py** – PDF text extraction + classification
- **code_generator.py** – Requirement-to-code logic
- **bug_fixer.py** – Bug detection & fixing
- **test_case_generator.py** – Test code generation
- **summarizer.py** – Code summarization
- **chatbot.py** – SDLC chatbot
- **ai_engine.py** – Unified AI model handler
- **main.py** – FastAPI app
- **streamlit_ui.py** – Streamlit UI

Running the Application

- **Backend (FastAPI)**:

```
css
CopyEdit
uvicorn main:app --reload
```

- **Frontend (Streamlit)**:

```
arduino
CopyEdit
streamlit run streamlit_ui.py
```

API Documentation

- **POST /classify-requirements** – Classify SDLC phase from text
- **POST /generate-code** – Generate Python code
- **POST /fix-bugs** – Return fixed code
- **POST /generate-tests** – Generate pytest code
- **POST /summarize-code** – Get code summary
- **POST /chatbot** – SDLC query response

Authentication

- Tokens (Hugging Face, Groq) loaded securely from `.env`.
- No user-level auth implemented (can be added for multi-user setups).

User Interface

- Streamlit app with tabs for each feature
- Lottie animations for better UX

Testing

- Manual testing of all features using example inputs
- Response time checks (target: <3s for code generation)

Known Issues

- Slight latency when fallback API triggers
- Limited support for non-Python code generation

Future Enhancements

- Multi-language code generation
- User authentication for saved sessions
- Advanced test case customization