**SmartSDLC** – **Project Documentation**

* Introduction

**Project Title: Smart SDLC**

Team Leader: UL. Kaviya

Team Members:

C. Kokila

A. Narmatha

S. Priyamani

* Project Overview

**Purpose**:

Smart SDLC is designed to simplify and optimize the software development lifecycle. It allows teams to manage requirements, track tasks, monitor bugs, and generate reports in a single platform. This system helps improve collaboration, speed, and quality of software delivery.

**Features**:

Requirement Management with traceability

Project Planning with timelines and Gantt charts

Task Assignment & Kanban-style tracking

Bug/Issue management with priority levels

Progress Report generation

Multi-role access (Admin, Developer, Tester)

Notifications & Alerts for deadlines

Document storage for project files and test cases

Git-based version control integration

Team collaboration support

* Architecture

Frontend (Gradie): Simple and interactive UI with tabs for requirement analysis and code generation.

Backend (Python & Transformers): Handles core logic and communication with LLM.

LLM (IBM Granite): Generates requirement lists and code outputs.

PDF Processing (PyPDF2): Reads text from uploaded PDF documents.

Database (Future Scope): MySQL/Firebase integration for storage.

**Workflow**:

1. User enters text or uploads PDF.

2. Backend extracts and processes content.

3. AI model classifies requirements or generates code.

4. Results displayed in Gradio interface.

5. Optionally stored in database (future).

* Setup Instructions
* Prerequisites:
* Python 3.9+
* pip (Python package manager)
* Internet access
* IDE (VS Code / PyCharm)
* Libraries Needed: torch, transformers, gradio, PyPDF2

Steps:

1. Download/clone SmartSDLC repository.

2. Open terminal in project folder.

3. Install dependencies using pip.

4. Run python app.py.

5. Open Gradio link in browser.

* Folder Structure
* app/ – Core backend logic and analysis functions
* app/api/ – API routes (future scope)
* ui/ – Frontend components (Gradio pages)
* models/ – Model integration and configs
* utils/ – Helpers for PDF and text operations
* requirements.txt – Library dependencies
* app.py – Main entry point
* README.md – Documentation

* Running the Application
* 1. Run python app.py.
* 2. Wait for Gradio startup message.
* 3. Open provided local/public link.
* 4. Use Requirement Analysis tab to upload PDF/text.
* 5. Use Code Generation tab for requirement → language → code.
* 6. View outputs instantly.

* API / Functions
* Requirement Analysis Function
* Code Generation Function
* PDF Reader Function
* AI Model Handler

* Authentication
* Currently open access. Planned:
* Role-based login
* Token-based authentication
* Session tracking

* User Interface
* Requirement Analysis Tab → PDF/text input → categorized results
* Code Generation Tab → requirement + language → code output
* Clean layout with textboxes, dropdowns, and outputs
* Testing
* Unit Testing – functions
* Integration Testing – frontend + backend
* Manual Testing – sample PDFs and prompts
* Edge Cases – empty/invalid inputs
* Known Issues
* Model may repeat/produce irrelevant text
* Processing slows with large PDFs
* No persistent storage yet
* Authentication missing

* 12. Future Enhancements
* Add database for saving results and reports
* Implement secure login with roles
* Support more languages/frameworks
* Generate detailed reports and dashboards
* Cloud deployment for remote access