Project Documentation

1. Introduction

Project Title: SmartSDLC — AI-Driven Software Development Lifecycle

Team Members:

1. Giritharan (Team Leader)
2. Guruchandran
3. Rishi
4. Vishal
5. Wadood

2. Project Summary

SmartSDLC uses IBM Granite large language models to speed up and streamline the software development lifecycle. The system accepts PDFs and text, extracts and organizes requirements, produces code snippets, creates test cases, helps diagnose and fix bugs, and offers an AI assistant to support developers. The goal is to boost productivity, reduce mistakes, and provide a repeatable, traceable workflow for building dependable software.

1. Requirement extraction — parse uploaded documents and categorize requirements.
2. Code generation — produce language-specific code (Python, Java, C++, etc.).
3. Automated test creation — generate unit tests from requirements.
4. Bug diagnosis and docs — suggest fixes and produce documentation.
5. Interactive AI assistant — answer developer queries and guide workflows.
6. GitHub integration — save and share project artifacts.
7. Colab-ready deployment — designed to run easily in Google Colab with GPU support.
8. User-friendly UI — Gradio interface for quick interaction.

3. System Design

Frontend: Gradio-based interface with tabs for requirement analysis and code generation.
Backend: Python stack leveraging Hugging Face-style transformer models and IBM Granite (granite-3.2-2b-instruct).
Deployment: Intended for Google Colab (T4 GPU); integrates with GitHub for persistence.
Main libraries and tools: transformers, torch, gradio, PyPDF2.

4. Setup & Prerequisites

Requirements: Python 3.9 or newer, Google Colab (T4 GPU recommended), Hugging Face account with IBM Granite access, GitHub account (optional).

1. Install dependencies: !pip install transformers torch gradio PyPDF2 -q
2. Open the provided Colab script or run the repository code.
3. Upload a requirements PDF or paste text into the UI.

4 Use the Gradio controls to analyze requirements, request code, or generate tests.
5 (Optional) Push outputs to a configured GitHub repository.

5. Repository Layout

1 app/ — primary application modules
2 requirements_analysis.py — requirement parsing and classification logic
3 code_generation.py — code synthesis for target languages
4 pdf_utils.py — PDF text extraction helpers
5 app_ui.py — Gradio interface definitions
6 main.py — application entrypoint

6. How to Run the Application

1 Open the project in Google Colab and enable GPU acceleration.
2 Run the installation cell(s) to install dependencies.
3 Launch the main script to start the Gradio app.
4 In the UI, use the Requirement Analysis tab to upload documents or paste text.
5 Use the Code Generation tab to select a target language and request snippets/tests.
6 Outputs appear within the app and can be downloaded or pushed to GitHub.

7. Team Responsibilities

1 Giritharan — Project coordination, UI refinements, and documentation.
2 Guruchandran — Backend model integration and requirement analysis implementation.
3 Rishi — Code generation logic, unit-test generation, and debugging.
4 Vishal — Support for code generation, testing and model evaluation.
5 Wadood — Colab deployment, GitHub integration, and setup automation.

8. Future Work / Enhancements

1 Provide REST API endpoints (e.g., /analyze, /generate-code, /generate-tests).
2 Add authentication and role-based access controls.
3 Improve GitHub collaboration via OAuth and multi-user workflows.
4 Expand language support and improve model prompting for higher-quality code and tests.