Smart SDLC AI Assistant Project Documentation

1 Introduction

1.1 Project Title

Smart SDLC AI Assistant

1.2 Team Members

[List team members and their roles]

2 Project Overview

2.1 Purpose

The Smart SDLC AI Assistant aims to streamline the Software Development Life Cycle (SDLC) by automating tasks such as requirement analysis, code generation, test case creation, bug fixing, and code summarization, enabling developers and small teams to save time, reduce errors, and enhance productivity.

2.2 Features

- AI-driven requirement analysis to convert plain English into structured modules.
- Multilingual code generation (Python, Java, C++).
- Automated test case generation for input code.
- Bug detection and fixing with detailed explanations.
- Code summarization for quick understanding.
- Chatbot for real-time SDLC query resolution.

3 Architecture

3.1 Frontend

The frontend is built using React, providing a dark-themed, intuitive UI. Components are modular, handling user inputs for requirements, code, and queries, with real-time feedback for code editing and summarization.

3.2 Backend

The backend uses Node.js and Express.js to manage API requests, process AI model outputs, and handle interactions between the frontend and database. FastAPI is integrated for high-performance endpoints.

3.3 Database

MongoDB stores user inputs, generated code, test cases, and summaries. The schema includes collections for user profiles, project requirements, code outputs, and test case results, ensuring efficient data retrieval and storage.

4 Setup Instructions

4.1 Prerequisites

- Node.js (v16 or higher)
- MongoDB (v4.4 or higher)
- Python (v3.8 or higher)
- Kaggle account for cloud deployment

4.2 Installation

1. Clone the repository: `git clone [repository-url]`
2. Navigate to the client directory: `cd client`
3. Install frontend dependencies: `npm install`
4. Navigate to the server directory: `cd ../server`
5. Install backend dependencies: `npm install`
6. Set up environment variables in `.env` (e.g., MongoDB URI, API keys).

5 Folder Structure

5.1 Client

- `src/components/`: React components for UI (e.g., RequirementInput, CodeEditor).
- `src/pages/`: Page-level components (e.g., Dashboard, CodeGenerator).
- `src/styles/`: CSS and Tailwind CSS for styling.

5.2 Server

- `routes/`: API routes for requirements, code generation, and testing.
- `controllers/`: Logic for processing requests and AI model integration.
- `models/`: MongoDB schemas for data storage.

6 Running the Application

- **Frontend**: Navigate to `client` directory and run `npm start`.
- **Backend**: Navigate to `server` directory and run `npm start`.

7 API Documentation

- **POST /api/requirements**: Convert plain text requirements to structured modules.
  - Parameters: `{ text: string }`
  - Response: `{ modules: object }`
- **POST /api/code/generate**: Generate code in specified language.
  - Parameters: `{ text: string, language: string }`
  - Response: `{ code: string }`
- **POST /api/test-cases**: Generate test cases for input code.
  - Parameters: `{ code: string }`
  - Response: `{ testCases: array }`

8 Authentication

Authentication is handled using JWT tokens. Users authenticate via API keys for secure access to endpoints. Data encryption (AES) ensures secure transmission of sensitive inputs.

9 User Interface

[Screenshots or GIFs showcasing UI features to be included.]

10 Testing

- **Unit Testing**: Jest for frontend components and Mocha for backend logic.
- **Integration Testing**: Test API endpoints with Postman.
- **AI Output Validation**: Manual checks for AI-generated code and test cases.

11 Screenshots or Demo

[Screenshots or demo link to showcase the application.]

12 Known Issues

- Limited language support for code generation (Python, Java, C++ only).
- Occasional delays in AI processing for large inputs.

13 Future Enhancements

- Support for additional programming languages.
- Enhanced AI model for faster processing.
- Integration with CI/CD pipelines for automated deployment.