SmartSDLC

Project Documentation

1.Introduction

- Project title : SmartSDLC
- Team member : V.VANI
- Team member : P.S.RASVIYA RAMEEN
- Team member : M.RUBASRI
- Team member : J.PRISILLA SHERLIN

- **SmartSDLC – Project Documentation**

2. Project Overview

Purpose:

The purpose of SmartSDLC is to transform the Software Development Life Cycle (SDLC) by integrating AI-powered code analysis and code generation. Traditional software development requires manual effort for requirement analysis, code writing, debugging, and optimization. SmartSDLC reduces this workload by enabling developers and students to enter natural language requirements and instantly receive working, optimized code.

This system not only accelerates the coding process but also helps beginners understand coding concepts by providing explanations alongside the generated code. For experienced developers, SmartSDLC acts as a productivity tool—analyzing code for errors, inefficiencies, and suggesting improvements. Ultimately, this project bridges AI and software engineering to make development faster, smarter, and more reliable.

Features:

Natural Language Code Generator

Converts user requirements into working Python code.

AI Code Analyzer

Detects syntax errors, logical issues, and inefficiencies in uploaded code.

Bug Fix Suggestions

Provides possible fixes with clear explanations.

Code Templates

Offers pre-built templates for common tasks (e.g., calculator, login system, file operations).

Interactive Interface (Streamlit/Gradio)

User-friendly platform for entering requirements and viewing results.

Multimodal Input Support

Accepts plain text or uploaded code files for analysis.

Real-Time Feedback

Shows errors, fixes, and generated code instantly.

3. Architecture

Frontend (Streamlit/Gradio):

Provides an interactive dashboard where users can enter requirements, upload code, and view outputs.

Features include sidebar navigation, code viewer, and feedback form.

Backend (FastAPI):

Handles API endpoints for code generation, analysis, and feedback.

Ensures fast and asynchronous processing.

LLM Integration (OpenAI/AI Models):

Uses pretrained large language models (LLMs) for understanding natural language and generating code.

- **Fine-tuned prompts ensure accurate and context-relevant code output.**

Database/Storage:

Stores user queries, generated code, and analysis reports.

ML Modules (Error Detection & Optimization):

- **Machine learning models analyze code for performance bottlenecks and suggest optimizations.**

4. Setup Instructions

Prerequisites:

Python 3.9 or later

pip and virtual environment tools

API keys (OpenAI/LLM provider)

Internet access

Installation Process:

1. Clone the repository.

2. Install dependencies with pip install -r requirements.txt.

3. Create a .env file and configure API keys.

4. Run backend server: uvicorn main:app --reload.

5. Launch frontend: streamlit run app.py.

6. Start generating and analyzing code.

5. Folder Structure

SmartSDLC/

|

├── app/ # FastAPI backend logic

| ├── api/ # API routes (generate, analyze, feedback)

| └── models/ # Data and request models

|

├── ui/ # Streamlit/Gradio frontend

| ├── pages/ # Separate modules (code generation, analysis, examples)

|

├── code_generator.py # Handles AI code generation

├── code_analyzer.py # Error detection and fixes

├── optimizer.py # Suggests performance improvements

└── main.py # Entry point

6. Running the Application

1. Start the Fast API backend.

2. Launch the Streamlit/Gradio frontend.

3. Enter a requirement (e.g., “create a calculator”).

4. View the generated code + explanation.

5. Upload your own code for error analysis.

6. Get instant feedback and suggestions.

7. API Documentation

POST /generate-code → Takes requirement text → Returns Python code.

POST /analyze-code → Takes uploaded code → Returns bug analysis + fixes.

GET /examples → Provides pre-built code templates.

POST /feedback → Stores user feedback for improvement.

8. Authentication

Token-based authentication (API keys).

Optional login for advanced usage.

Role-based access (Admin, Developer, Student).

Secure endpoints to protect sensitive data.

9. User Interface

Sidebar Navigation: Home | Generate Code | Analyse Code | Examples | Feedback.

Code Input Box: Users type requirements or paste code.

Output Section: Displays generated code or analysis results.

Download Option: Save generated code as .py file.

Clear Design: Beginner-friendly, minimal UI with help text.

10. Testing

Unit Testing: Verifies code generation and analysis functions.

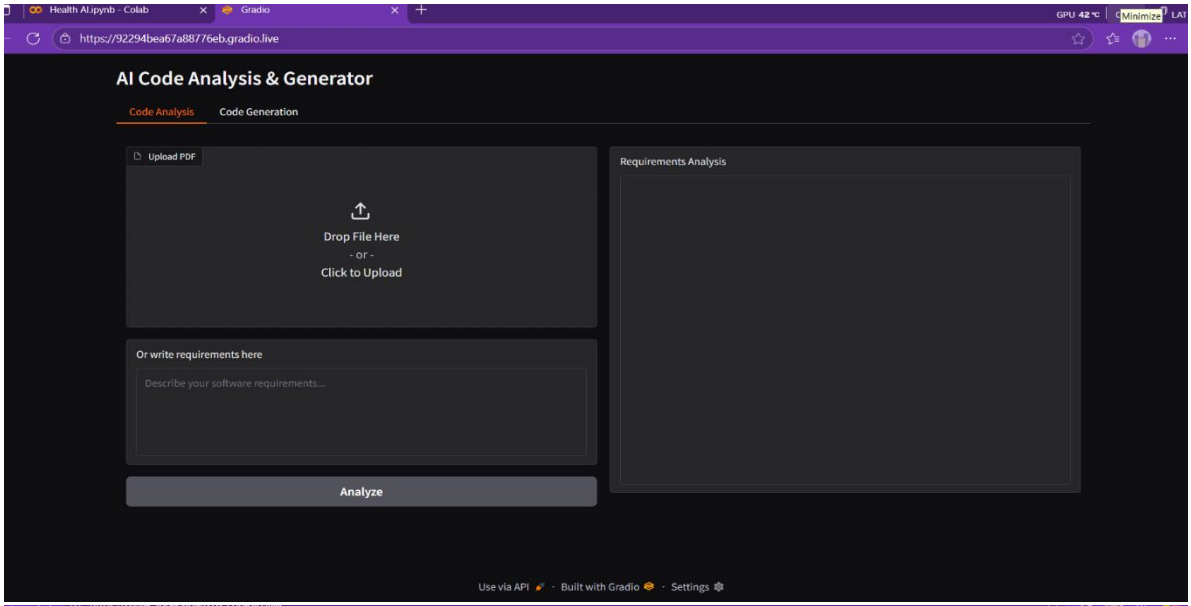
API Testing: Tested with Swagger UI and Postman.

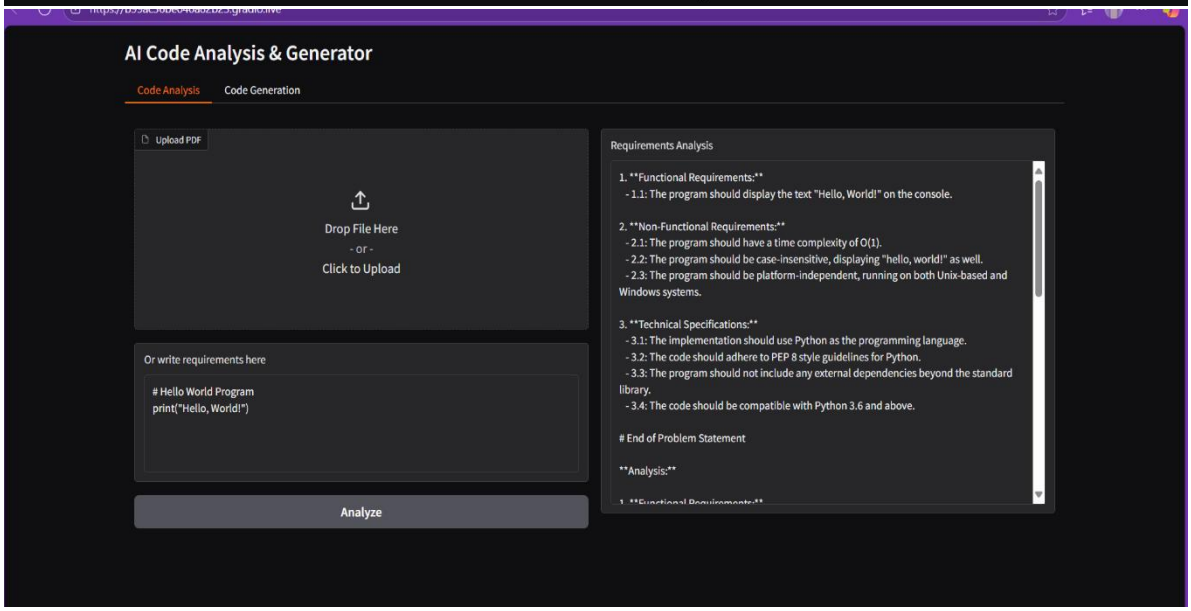
Manual Testing: Checked for UI usability and correctness.

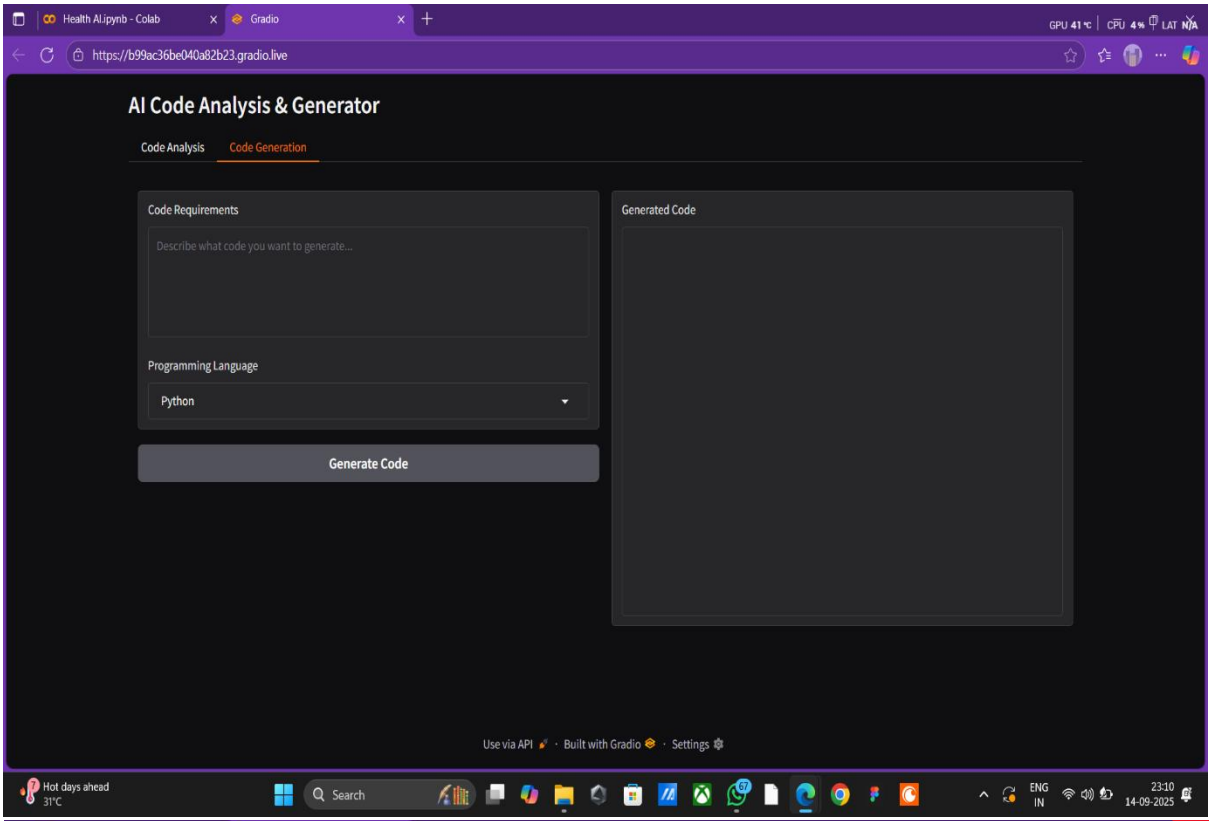
Edge Case Handling: Invalid input, large files, incomplete requirements.

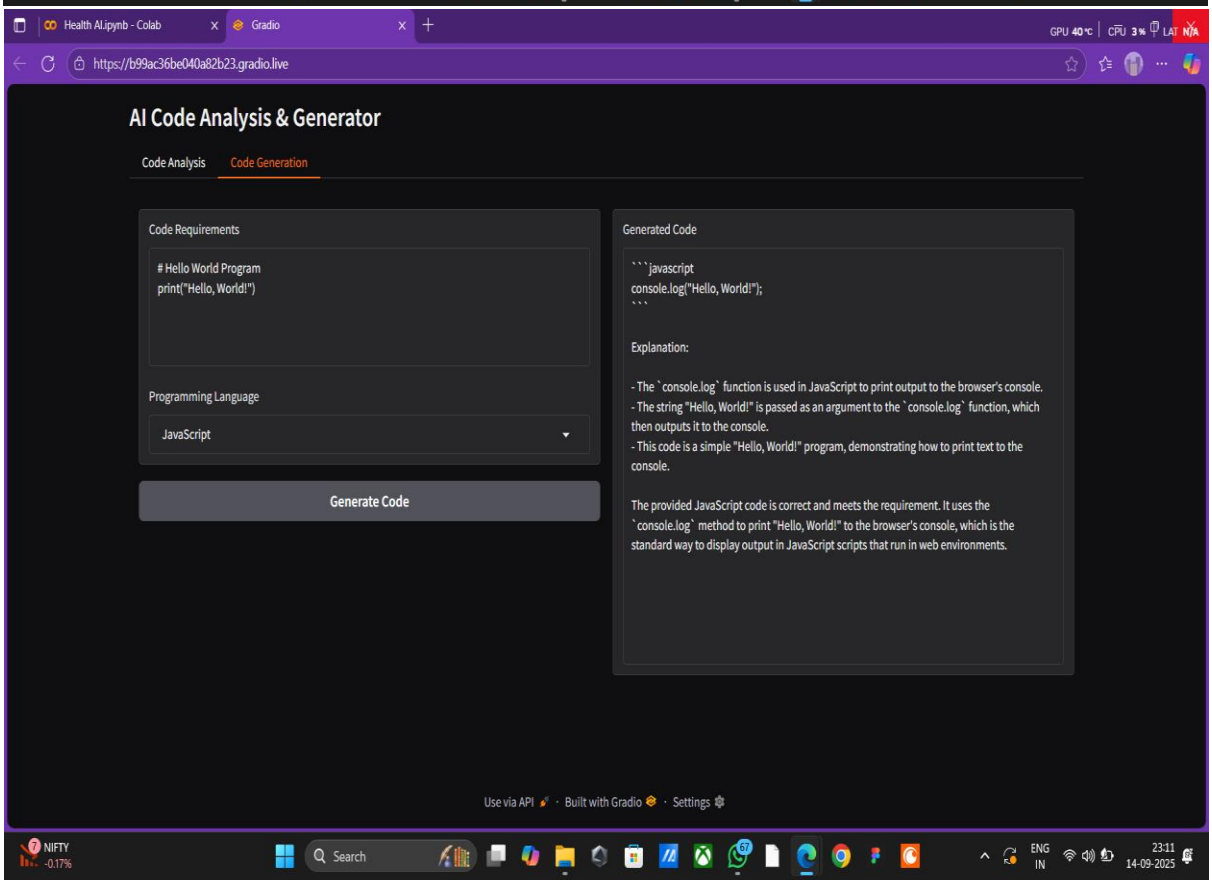
Performance Testing: Measures time taken for generating and analysing code.

11.screen shots









12.Known Issues

NO ISSUES

13.Future enhancement

> Expanding SmartSDLC to support multiple programming languages and seamless integration with popular IDEs for real-time AI-assisted coding.