SmartSDLC – AI-Enhanced Software Development Lifecycle Generative AI with IBM

1. Introduction

Project Title: SmartSDLC – AI-Enhanced Software Development Lifecycle Generative AI with IBM

Team ID: NM2025TMID03815

Team Size: 5

Team Leader: MATHUMITHA G

Team Member: DARVINA D

Team Member: HARITHA V

Team Member: ABHI VS

Team Member: MALAR ROJA L

2. Project Overview

Purpose: The purpose of SmartSDLC is to revolutionize the traditional Software Development Lifecycle by integrating Generative AI and IBM Watsonx to enhance automation, efficiency, and decision-making. This system acts as an intelligent assistant that supports developers, project managers, and testers throughout the SDLC stages. From requirement gathering to deployment, SmartSDLC ensures higher code quality, faster delivery, and reduced human errors.

Features:

• Requirement Analysis Assistant – AI-driven documentation and requirement refinement.

• Code Generation & Review – Generative AI for accelerated coding.

• Automated Test Case Generator – AI-powered quality assurance.

• Project Timeline Forecaster – Predictive project planning.

• Bug Detection & Resolution – Early issue identification.

• Documentation Generator – Automated knowledge base.

• KPI Dashboard – Performance insights.

• Collaboration Assistant – Seamless team communication.

3. Architecture

Frontend (Streamlit): Provides an intuitive dashboard for project managers and developers.

Backend (FastAPI): Handles API endpoints for requirements processing, AI code generation, test creation, and KPI forecasting.

LLM Integration (IBM Watsonx + Granite): Powers requirement analysis, documentation generation, and intelligent coding assistance.

Vector Search (Pinecone): Enables semantic search on project documentation and past repositories.

ML Modules: Forecasting & Bug Detection using ML models.

4. Setup Instructions

Prerequisites: Python 3.9+, pip, IBM Watsonx API key, Pinecone API key, Internet access.

Installation Process:

1. Clone the repository

2. Install dependencies from requirements.txt

3. Configure .env with credentials

4. Run backend using FastAPI

5. Launch frontend via Streamlit

6. Upload project data and interact with modules

5. Folder Structure

app/ – Backend logic (FastAPI routers, models, integrations) app/api/ –

Modular APIs (requirements, code, testing, forecasting, reporting) ui/ –

Streamlit frontend (dashboards, forms, charts) smart_sdlc.py – Main

Streamlit dashboard granite_llm.py – Handles IBM Watsonx integration

doc_generator.py – Auto documentation and reports test_case_ai.py – AI-

based test case generation kpi_forecaster.py – Forecasts delivery timelines

and risks bug_detector.py – Detects and suggests fixes for code issues

6. Running the Application

Start FastAPI server

Run Streamlit dashboard

Navigate via sidebar to modules

Upload documents, generate test cases, view forecasts, and download reports

7. API Documentation

POST /requirements/analyze – Processes client input and outputs structured requirements

POST /code/generate – Generates code snippets from requirements

POST /test/generate – Creates automated test cases

GET /kpi/forecast – Predicts project delivery timelines and metrics

POST /report/generate – Generates project documentation and reports

POST /bug/detect – Detects vulnerabilities and suggests fixes

8. Authentication

Token-based authentication (JWT)

Role-based access control (Project Manager, Developer, Tester, Client)

Planned features: session management & secure cloud integration

9. User Interface

Sidebar navigation

Requirement and code review tabs

KPI visualizations with charts

Automated documentation download (PDF/Word)

Real-time AI recommendations

10. Testing

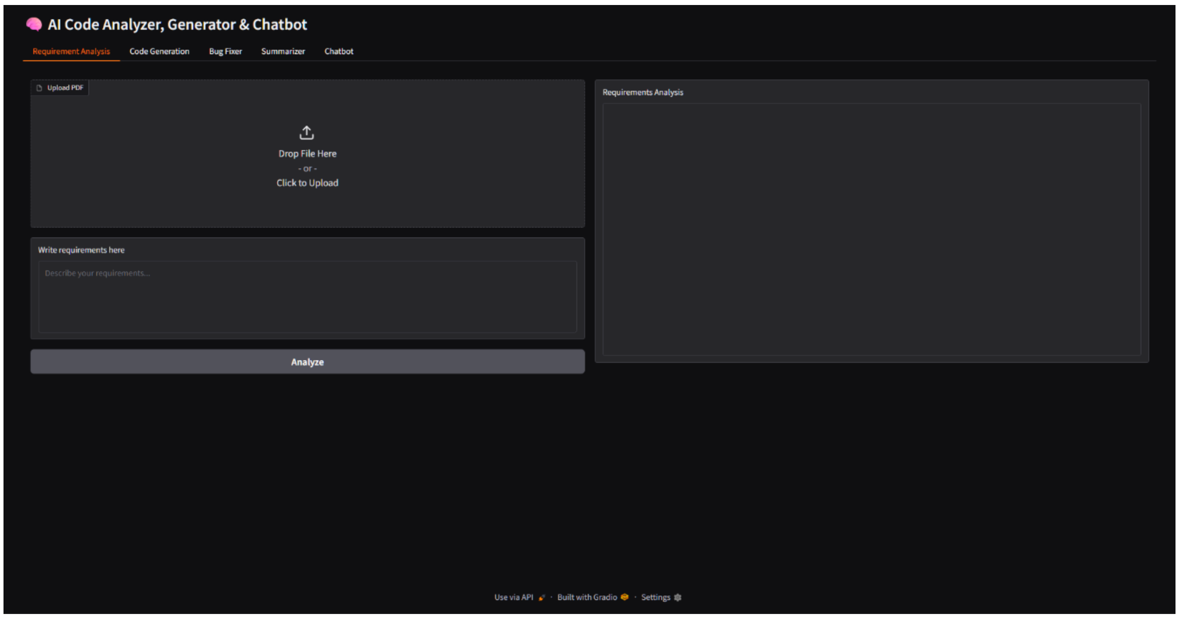
Unit Testing: Requirement parsing, code generation

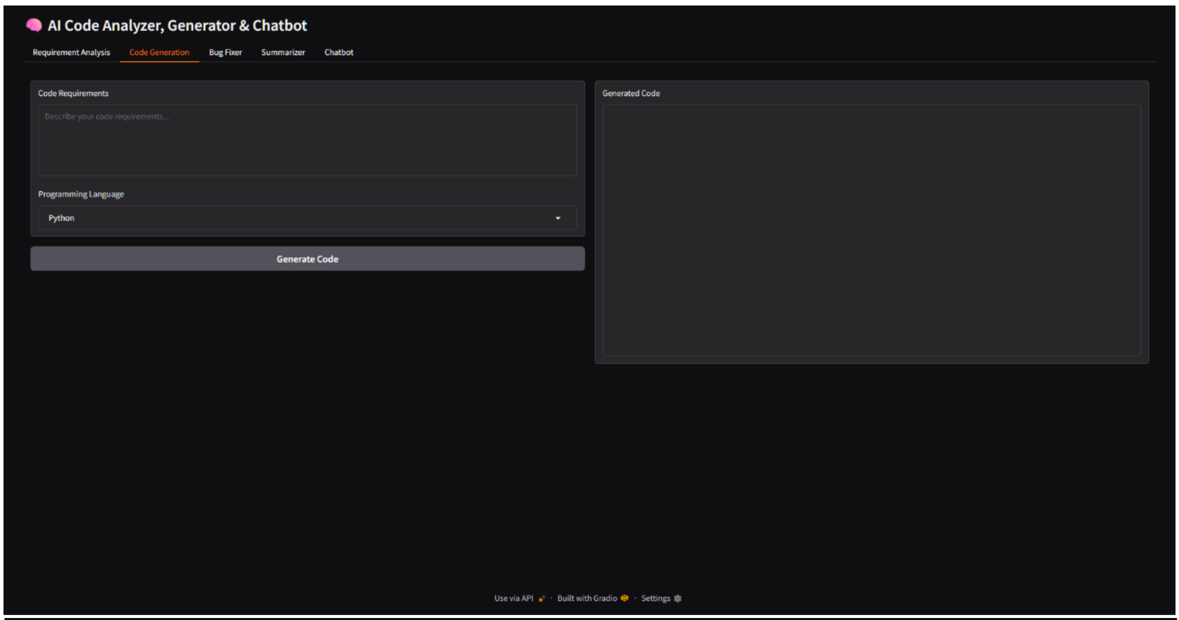
API Testing: Swagger & Postman

Manual Testing: End-to-end project flows

Edge Cases: Invalid inputs, missing data, API failures

11. Screenshots





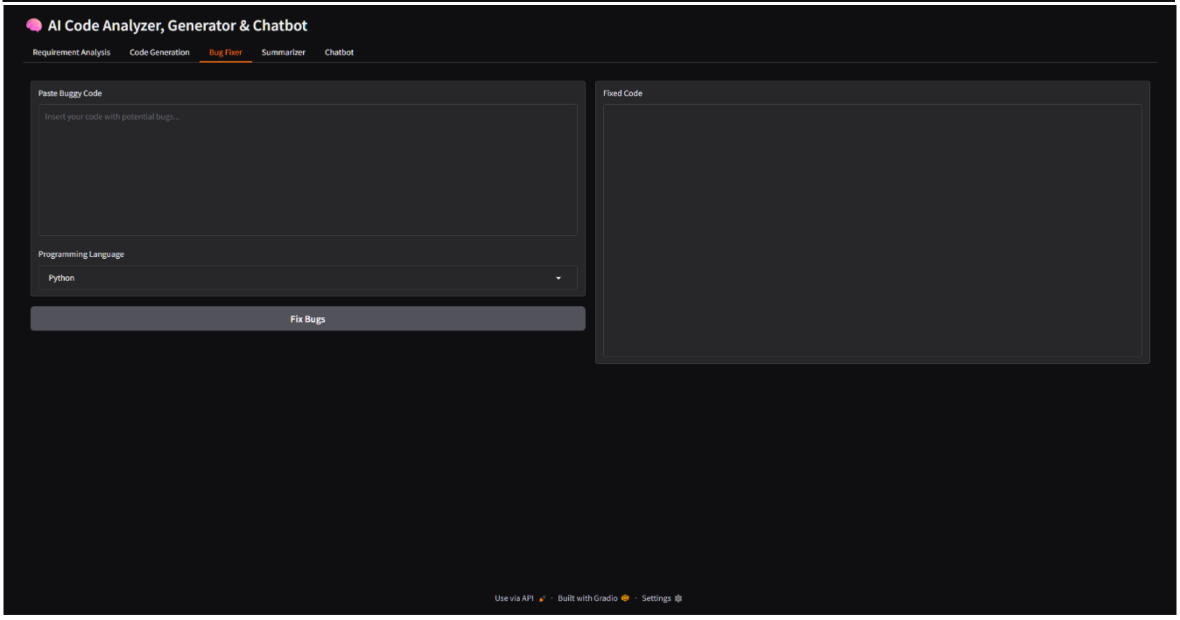
🧠 AI Code Analyzer, Generator & Chatbot

Code Requirements

Describe your code requirements...

Programming Language: Python

Generated Code



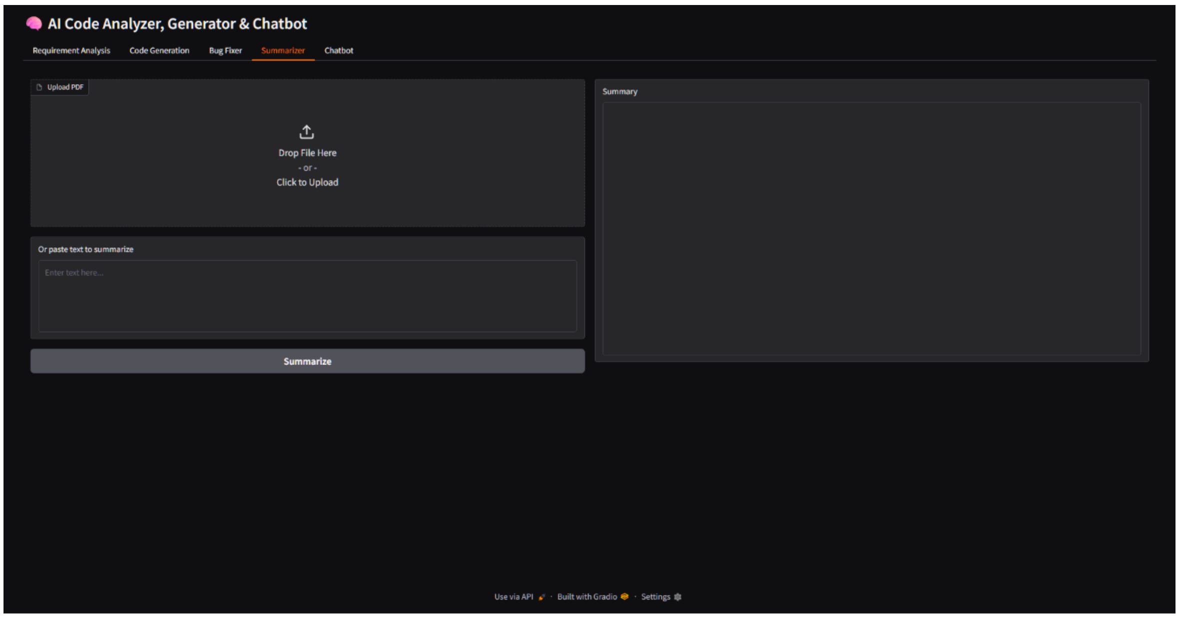
🧠 AI Code Analyzer, Generator & Chatbot

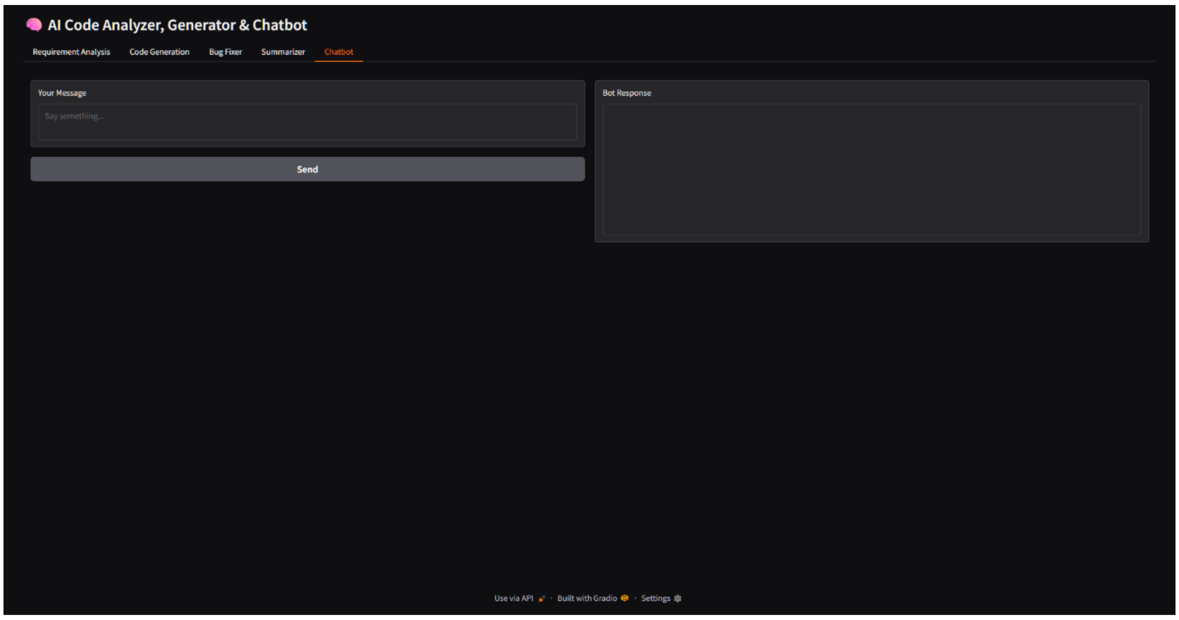
Paste Buggy Code

Insert your code with potential bugs...

Programming Language: Python

Fixed Code





12. Known Issues

Limited offline support

Heavy dependency on internet/cloud APIs

AI suggestions require manual validation in complex use cases

13. Future Enhancements

Multi-language support for requirements and documentation

Integration with Jira, GitHub, GitLab

Enhanced bug prediction using deep learning

Voice-enabled project assistant

14. Demo Link:

https://drive.google.com/file/d/1YCMGcK0GLcKnBbfuEGWN3wq9YCXbVSfW/view?usp=sharing