SmartSDLC – AI-Enhanced Software Development Lifecycle

Generative AI with IBM

Project Documentation

1. Introduction

- **Project Title**: SmartSDLC – AI-Enhanced Software Development Lifecycle
- **Team Members**:
    - ❖ YASMIN BEEVI S
    - ❖ SRIVITHYA A
    - ❖ SHAHANA S
    - ❖ SHPINO J
    - ❖ SHREEMATHI G

---

2. Project Overview

Purpose

The purpose of **SmartSDLC** is to revolutionize the traditional software development lifecycle by integrating **Artificial Intelligence (AI), automation, and predictive analytics** into every phase. The system acts as a digital assistant for developers, testers, project managers, and stakeholders, enabling faster, more reliable, and data-driven software development.

SmartSDLC enhances productivity by:

- Automating requirement analysis and documentation.
- Generating optimized code snippets and design suggestions.
- Predicting project risks and delays with AI-driven forecasting.

- Enhancing testing through intelligent test case generation and defect detection.
- Providing continuous feedback and improvement recommendations.

Ultimately, SmartSDLC helps organizations **deliver high-quality software faster, with reduced cost and improved adaptability**.

Key Features

- **AI-Powered Requirement Analysis** – Converts natural language specifications into structured requirements.
- **Automated Design Assistance** – Suggests UML diagrams, architectures, and workflows using AI.
- **Code Generation & Optimization** – Produces boilerplate code and performance recommendations.
- **Intelligent Testing** – Auto-generates test cases, performs regression analysis, and detects anomalies.
- **Project Risk Forecasting** – Predicts possible delays, bottlenecks, and cost overruns.
- **Continuous Feedback Loop** – Collects input from team members and stakeholders for refinement.
- **KPI Monitoring & Reporting** – Tracks project health with AI-driven dashboards.
- **Integration with DevOps Tools** – Seamlessly connects with GitHub, Jenkins, JIRA, and CI/CD pipelines.

---

3. Architecture

Frontend (Streamlit/Gradio)

- Interactive dashboard for project managers and developers.
- Modules for requirements, coding, testing, and reporting.

- Visual charts for KPIs, risks, and progress.

Backend (FastAPI/Flask)

- Manages AI services, ML models, and project data.
- Provides APIs for requirement parsing, test generation, and code analysis.

AI/LLM Integration

- Uses large language models (LLMs) for:
    - Requirement-to-code translation.
    - Automated documentation.
    - Test case and bug report summarization.

Data & ML Modules

- **Forecasting** – Time-series models for project completion and cost.
- **Anomaly Detection** – Identifies defects, risks, and performance issues.
- **Knowledge Base** – Stores reusable project learnings.

Vector Search (Pinecone/FAISS)

- Enables semantic search across requirements, documentation, and code repositories.

---

4. Setup Instructions

Prerequisites:

- Python 3.9+
- Virtual environment tools
- API keys for LLM services (e.g., OpenAI, IBM Watsonx, or others)

- GitHub/JIRA integration enabled

Steps:

1. Clone repository
2. Install dependencies via requirements.txt
3. Configure .env with API keys and project settings
4. Run FastAPI backend
5. Launch Streamlit dashboard
6. Upload requirements, code, or test data

---

5. Folder Structure

- app/ – Backend logic (FastAPI services, ML modules, integrations)
- app/api/ – API routes (requirements, code, testing, reporting)
- ui/ – Frontend dashboard (Streamlit/Gradio components)
- models/ – Trained ML models for forecasting, anomaly detection
- requirement_parser.py – Converts natural language into structured requirements
- test_case_generator.py – Creates automated test scripts
- risk_forecaster.py – Predicts project risks and timelines
- report_generator.py – Generates AI-driven project status reports

---

6. Running the Application

- Start FastAPI backend
- Launch Streamlit UI

- Navigate modules via sidebar
- Upload requirements, interact with AI assistant, generate reports

---

7. API Documentation

- POST /analyze-req → Requirement parsing & structuring
- POST /generate-code → Code snippet generation
- POST /generate-tests → Test case creation
- GET /forecast-risks → Project risk forecasting
- GET /project-metrics → Retrieve KPIs and progress

---

8. Authentication

- JWT-based authentication
- OAuth2 integration with enterprise accounts
- Role-based access: Admin, Developer, Tester, Manager

---

9. User Interface

- Sidebar navigation (Requirements | Coding | Testing | Reports)
- KPI dashboards (velocity, quality, risk level)
- Tabbed layouts for tasks, feedback, and project history
- Exportable project reports (PDF/CSV)

---

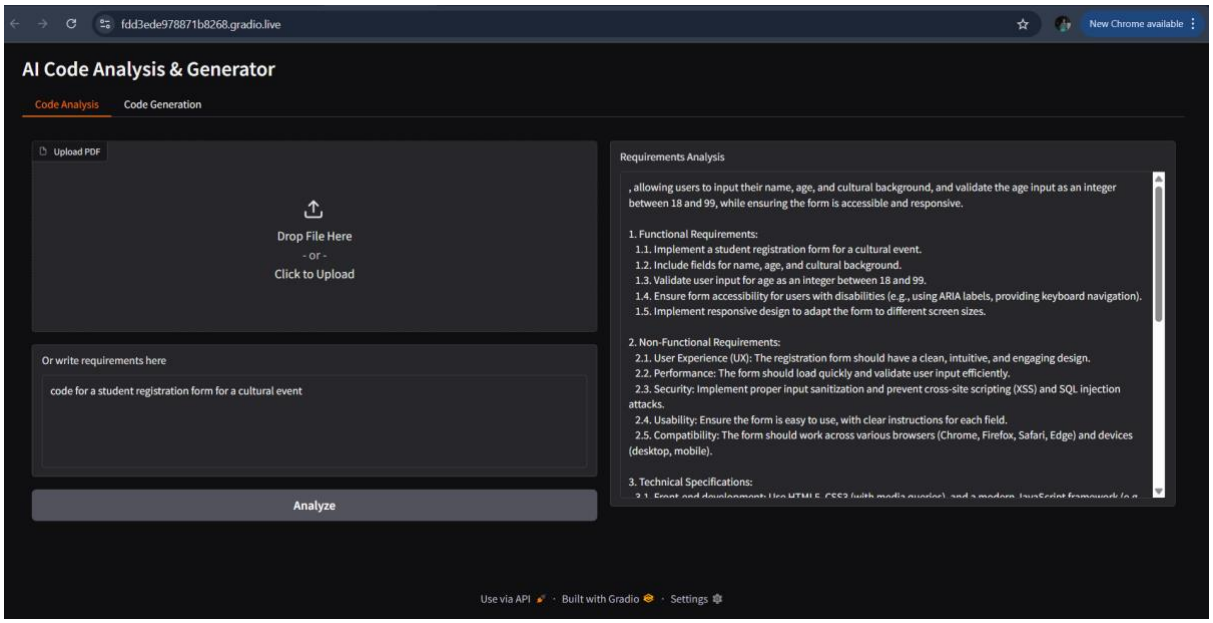
10. Testing

- **Unit Testing**: Requirement parser, code generation, test scripts
- **API Testing**: Swagger UI & Postman

- **Manual Testing**: Workflow validation, real-time predictions
- **Edge Case Handling**: Incomplete requirements, code conflicts

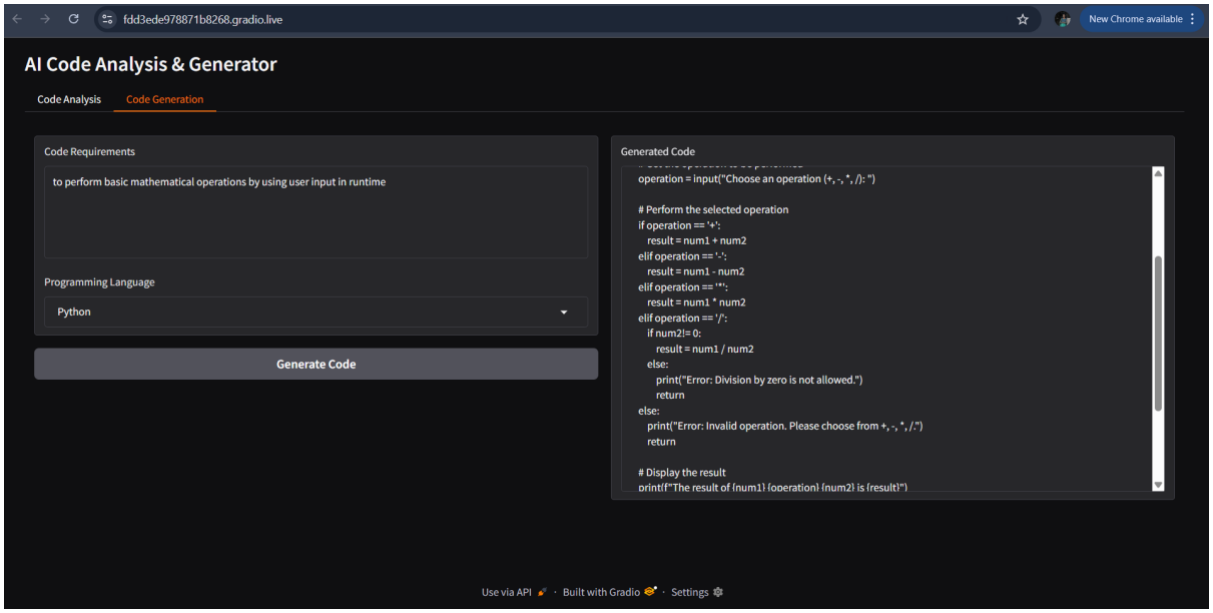
---

11. Screenshots



Here we can give our requirements for the code and it will analyze

And it shows that what are the specification add to the code.



And , now we can choose the programming language and fill the requirements and get code for given requirements and use it.

12. Known Issues

- AI model may produce incomplete or ambiguous requirements.
- Forecasting accuracy depends on quality of historical data.
- Limited offline support for some modules.

---

13. Future Enhancements

- Integration with **multi-agent AI systems** for collaborative coding.
- More advanced **automated debugging tools**.
- **Voice-based interface** for requirement input.
- **Cross-language support** for global development teams.
- Enhanced **CI/CD automation with AI-driven deployment suggestions**.
