SmartSDLC – AI-Enhanced Software Development Lifecycle

Project Documentation

1. Introduction

- Project Title: Smart SDLC – AI-Enhanced Software Development Lifecycle
- Team Member: Sree Sudharshini .T.R
- Team Member: Ayisha .SK
- Team Member: Deena.S
- Team Member: Tahaimeena .S

2. Project Overview

Purpose:

The purpose of Smart SDLC is to bring intelligence, automation, and adaptability into the traditional Software Development Lifecycle (SDLC). Instead of relying on manual and error-prone processes, Smart SDLC integrates AI models, predictive analytics, and automation tools to:

- Analyze requirements automatically and highlight inconsistencies.
- Provide AI-assisted code reviews and bug detection.
- Generate test cases and predict project risks.
- Summarize documentation and suggest improvements.
- Help managers forecast timelines, KPIs, and resources.

This project aims to improve quality, speed, and collaboration across all stages of software development—ensuring efficient delivery, reduced costs, and continuous improvement.

Features:

- AI-Powered Requirement Analysis
- Key Point: Eliminates ambiguity in requirements.
- Functionality: Uses NLP to extract, validate, and prioritize requirements.
- Code Quality & Review Assistant

Key Point:

- Functionality: Detects vulnerabilities and suggests fixes.
- Test Case Generator
- Key Point: Automates QA processes.
- Functionality: Generates functional, regression, and integration tests.
- Project Risk Forecasting
- Key Point: Avoids unexpected delays.
- Functionality: Predicts risks in scheduling, resources, and delivery.
- KPI Forecasting & Sprint Tracking
- Key Point: AI-driven project monitoring.
- Functionality: Visual dashboards to track progress and performance.
- Anomaly Detection
- Key Point: Early issue identification.
- Functionality: Detects unusual code or workflow patterns.
- Documentation Summarizer
- Key Point: Faster project knowledge sharing.
- Functionality: Converts long documents into concise summaries.
- Interactive Dashboard (Streamlit/Gradio)
- Key Point: Simplified UI.
- Functionality: Real-time dashboards for developers and managers.

3. Architecture

- **Frontend (Stream lit /Gradio):**

Provides dashboards, file uploads, sprint tracking, bug reports, and project chat assistance.

- **Backend (Fast API):**

Manages APIs for requirement parsing, bug detection, test generation, KPI analysis, and reporting.

- **LLM Integration (Open AI/Watsonx/Granite):**

Used for natural language understanding in requirement analysis, summarization, and code review.

- **Vector Search (Pinecone):**

Stores embeddings of project docs, requirements, and past bug reports for semantic search.

- ML Modules (Forecasting + Anomaly Detection):
- Forecasts project timelines, costs, and performance.
- Detects unusual errors, build failures, or delays.

4. Setup Instructions

Prerequisites:

- Python 3.9+
- pip & virtualenv
- API keys (for AI/LLM + Pinecone)
- Internet access

Installation:

1. Clone repository.
2. Install dependencies (pip install -r requirements.txt).
3. Create.env with API credentials.
4. Run backend server (u vi corn main: app).
5. Launch frontend (stream lit run dashboard.py).
6. Upload project docs/code and interact with Smart SDLC.

5. Folder Structure

- app/ → Fast API backend logic (routers, models, analyzers).
- app/ api/ → Endpoints for requirements, code review, bug detection, forecasting.
- ui / → Stream lit dashboards, project forms, visual reports.
- smart_dashboard.py → Launch script for UI.
- llm_helper.py → AI model integration (summarization, review).
- document_embedder.py → Converts docs into embeddings.

- kpi_forecaster.py → Predicts sprint velocity & delivery.
- anomaly_checker.py → Detects irregularities in workflow.
- report_generator.py → Generates AI-driven project reports.

6. Running the Application

1. Start Fast API backend.
2. Run Stream lit dashboard.
3. Upload requirements/docs/code.
4. Interact with assistant (queries, bug analysis, test generation).
5. Review dashboards for progress, risks, and reports.
6. All results update in real-time via backend APIs.

7. API Documentation

- POST /chat/ask – Ask queries about project lifecycle.
- POST /upload-doc – Upload & embed requirement/code files.
- GET /search-docs – Semantic search across project documents.
- GET /forecast-kpi – Predict delivery time, sprint progress, risks.
- POST /submit-feedback – Collects user/project team feedback.

All APIs documented in Swagger UI.

8. Authentication

Supports:

- Token-based authentication (JWT/API keys).
- OAuth2 for secure cloud integration.
- Role-based access (admin, developer, tester, manager).

Planned: session tracking, history, and audit logs.

9. User Interface

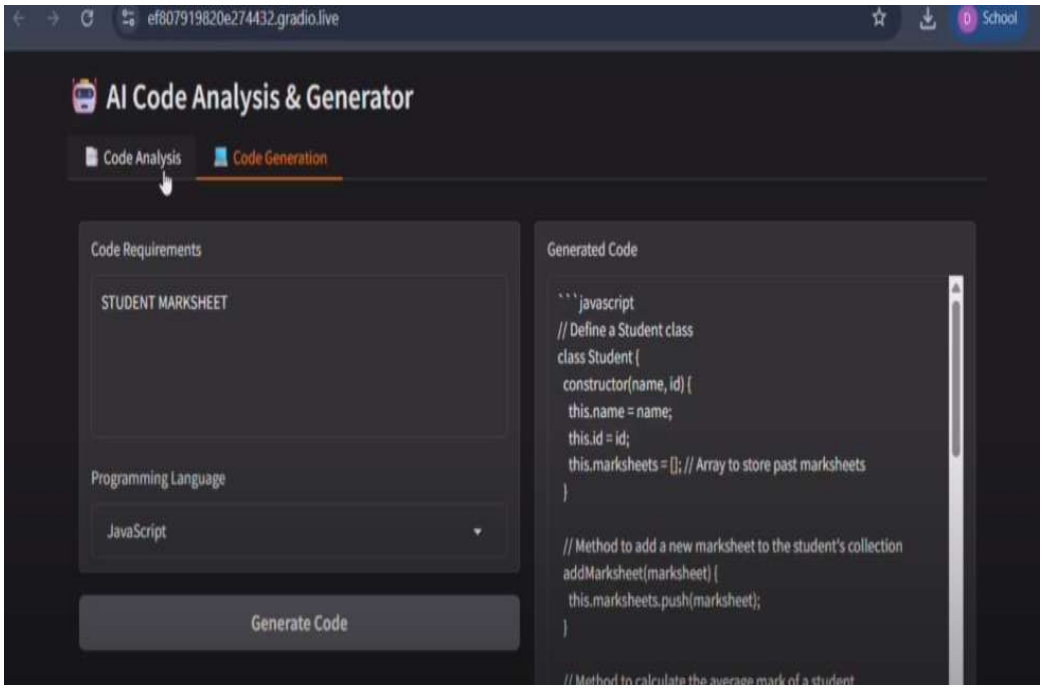
- Sidebar navigation.
- KPI dashboards with progress cards.
- Tabs for requirements, testing, bug reports.
- Real-time AI assistant.
- PDF/CSV report downloads.

Focus on clarity, speed, and accessibility for both tech & non-tech users.

10. Testing

- Unit Testing: Requirement parsers, bug detectors.
- API Testing: Swagger, Postman.
- Manual Testing: File uploads, AI responses, dashboards.
- Edge Cases: Corrupted files, invalid inputs, missing APIs.

11. Screenshots



12. Known Issues

- Limited to AI accuracy for requirement ambiguities.
- Heavy datasets may slow down embedding/forecasting.
- Some automation features still require manual review.

13. Future Enhancements

- Integration with Jira / GitHub for real-time project tracking.
- Deeper AI code completion and bug-fixing support.
- Multi-language support for global teams.
- Advanced Dev Ops pipeline integration.
- Continuous learning from past project data.