smartSDLC – AI Enhanced Software Development Lifecycle

1. Introduction

- **Project Title:** smartSDLC – AI Enhanced Software Development Lifecycle
- **Team leader:NAVEEN KUMAR S**
- **Team member:**MANIARASU S
- **Team member:**NISANTH S
- **Team member:**PRADEEP RAJA T

2. Project Overview

- **Purpose:**.

  The purpose of smartSDLC is to integrate AI into every stage of the software development lifecycle (SDLC), enabling faster delivery, higher code quality, improved project predictability, and reduced human error. The system acts as a **co-pilot for developers, project managers, and QA teams**, automating repetitive tasks and offering intelligent insights

- **Features:**
  - **AI Requirements Analyzer**
    - *Key Point:* Natural language understanding
    - *Functionality:* Converts business requirements into structured user stories, acceptance criteria, and test cases.
  - **Code Generation & Review**
    - *Key Point:* Automated coding support
    - *Functionality:* Suggests code snippets, reviews commits for bugs/security issues, and enforces coding standards.
  - **Automated Testing Assistant**
    - *Key Point:* Test case generation
    - *Functionality:* Generates unit/integration test cases, validates coverage, and simulates edge cases.
  - **Project Timeline Forecaster**
    - *Key Point:* Predictive project management
    - *Functionality:* Uses ML to forecast sprint velocity, backlog burn-down, and delivery risks.

- **Defect Prediction & Anomaly Detection**
  - *Key Point:* Early issue detection
  - *Functionality:* Identifies modules with high defect probability based on commit history, churn, and complexity.
- **Documentation Generator**
  - *Key Point:* Always-updated docs
  - *Functionality:* Creates and maintains technical/project documentation automatically from code and commits.
- **Conversational DevOps Bot**
  - *Key Point:* Natural language interface
  - *Functionality:* Allows team members to query project status, deployments, and issues using chat.

---

3. Architecture

- **Frontend (Dashboard – Streamlit/Gradio):**
  Interactive UI for project managers, developers, and testers. Provides dashboards, backlog visualization, test coverage reports, and DevOps status.
- **Backend (FastAPI / Node.js):**
  Exposes APIs for requirements analysis, code review, forecasting, and test generation.
- **LLM Integration (Watsonx / OpenAI / Anthropic):**
  Large Language Models used for natural language understanding, code review, documentation, and test case generation.
- **Vector Database (Pinecone / Weaviate):**
  Stores embeddings of requirements, project documentation, and historical issues for semantic search and traceability.
- **ML Modules (Forecasting & Risk Detection):**
  Predicts delivery risks, identifies potential bottlenecks, and recommends optimizations using historical sprint/project data.
- **DevOps Integration:**
  Connects with GitHub/GitLab, Jira, Jenkins, Docker, Kubernetes for CI/CD pipeline intelligence.

---

4. Setup Instructions

Prerequisites:

- Python 3.9+ (for AI modules)
- Node.js (for optional backend services)
- Access to LLM APIs (Watsonx, OpenAI, or Anthropic)
- GitHub/GitLab integration tokens
- Docker & Kubernetes (for deployment pipeline)

Installation Process:

1. Clone the repository.
2. Install backend dependencies (requirements.txt).
3. Configure .env with API keys and integration tokens.
4. Start FastAPI backend server.
5. Launch frontend dashboard with Streamlit/Gradio.
6. Connect to project repo + CI/CD pipeline.

---

5. Folder Structure

```
smartSDLC/
├── app/              # Backend logic
│   ├── api/          # Endpoints: requirements, code review, testing, forecasting
│   ├── models/       # AI/ML models
│   └── integrations/ # GitHub, Jira, CI/CD connectors
│
├── ui/               # Frontend dashboard (Streamlit/Gradio)
├── llm_engine.py     # Handles prompts + LLM interactions
├── test_generator.py # AI-based test case generation
├── forecast_engine.py # ML-based sprint forecasting
├── doc_generator.py  # Auto-documentation generator
├── devops_bot.py     # Conversational DevOps assistant
└── smart_dashboard.py # Entry point for UI
```

---

6. Running the Application

- Start backend API with FastAPI.
- Run Streamlit/Gradio dashboard.
- Authenticate GitHub/ Jira integrations.
- Upload requirements or connect to existing backlog.
- Interact with AI modules for code review, test generation, and forecasting.

---

7. API Documentation

- **POST /requirements/analyze** – Convert natural language requirements into user stories + test cases.
- **POST /code/review** – Review code for bugs, security issues, and best practices.
- **GET /forecast/sprint** – Predict sprint velocity and delivery timeline.
- **POST /tests/generate** – Generate unit/integration test cases from code.
- **GET /project/status** – Return project KPIs, defect hotspots, and risks.
- **POST /chat/query** – Natural language queries for DevOps/project status.

---

8. Authentication

- Token-based (JWT) authentication.
- Role-based access: Developer, Tester, Manager, Admin.
- OAuth2 integration with GitHub/ Jira/Slack.

---

9. User Interface

- Sidebar navigation: Requirements → Code Review → Testing → Forecast → Docs → DevOps Bot.
- KPI dashboards (velocity, defect trends, coverage).
- Tabbed layouts for code review reports, backlog insights, and test coverage.
- Real-time notifications (Slack/MS Teams integration).

---

10. Testing

- **Unit Testing:** Prompt engineering functions, ML models.
- **API Testing:** Swagger/Postman collections.
- **Integration Testing:** With GitHub commits + CI/CD pipeline.
- **Edge Cases:** Large repos, ambiguous requirements, noisy commit history.

---

11. Known Issues

- Dependency on external APIs for LLM.
- Forecast accuracy may vary with limited historical data.
- High compute cost for large repos.

---

12. Future Enhancements

- Fine-tuned in-house LLM for domain-specific SDLC tasks.
- Multi-modal support (upload PRDs, UML diagrams, logs).
- Autonomous bug-fixing + auto-pull request creation.
- Predictive cost estimation for projects.
- Integration with AR/VR code review environments.

13.Screenshorts: