Project Report

Title: Smart SDLC – AI-Enhanced Software Development Lifecycle (AI-SDLC)

---

1. Introduction

Project Title: AI-Enhanced Software Development Lifecycle (AI-SDLC)

Team Members:

1. Raksha N
2. Santhiya M
3. Sathana M
4. Sandhiya K
5. Aathilakshmi A

---

2. Project Overview

Purpose

The purpose of AI-SDLC is to revolutionize traditional software engineering processes by embedding artificial intelligence into every phase of the Software Development Lifecycle. By integrating advanced AI models for planning, coding, testing, deployment, and monitoring, the system increases development efficiency, reduces human errors, and enables predictive maintenance.

It provides intelligent tools for:

- Code generation
- Automated test case creation
- Real-time bug detection
- Performance monitoring
- Role-specific insights for developers, testers, project managers, and DevOps engineers

Key Features

1. **AI-Powered Requirement Analysis**

    - **Key Point:** NLP-driven extraction of user needs
    - **Functionality:** Converts user stories, emails, and documents into formal requirements using LLMs
2. **Smart Code Generator**
    - **Key Point:** Auto-generates boilerplate or module code
    - **Functionality:** Uses AI models (CodeGen, Codex) to write functions from structured prompts
3. **AI Test Writer**
    - **Key Point:** Auto-generates test cases
    - **Functionality:** Analyzes source code and creates unit and integration test cases
4. **Bug and Anomaly Detector**
    - **Key Point:** Identifies runtime bugs and logical errors
    - **Functionality:** Integrates ML-based anomaly detection and static analysis tools
5. **Predictive DevOps Dashboard**
    - **Key Point:** Forecasts system reliability
    - **Functionality:** Uses time-series models to predict downtimes, deployment issues, or performance drops
6. **Conversational Assistant**
    - **Key Point:** DevOps and code assistant
    - **Functionality:** AI chatbot for answering technical questions, explaining code, and suggesting improvements

---

3. System Architecture

Frontend (Streamlit)

- Interactive web dashboard with multi-tab support
- Modules: Requirement Upload, Code Generation, Test Generator, Bug/Anomaly Viewer, Monitoring Dashboard
- Uses `streamlit-option-menu` for sidebar navigation

Backend (FastAPI)

- REST API endpoints for requirement parsing, code/test generation, bug detection, and monitoring
- Asynchronous endpoints for performance
- Swagger UI enabled for documentation

LLM Integration

- Uses Codex, IBM Watsonx, or open-source models
- Handles code generation, test generation, documentation summarization, and explanations

AI Models Used

- **NER Models:** Requirement extraction
- **CodeGen/Codex:** Code generation
- **TestBERT:** Test case generation
- **Isolation Forest/LSTM:** Bug and anomaly detection
- **ARIMA/Prophet:** Predictive DevOps metrics

---

4. Setup Instructions

Prerequisites

- Python 3.9+
- Virtual environment tools
- API keys (OpenAI/IBM Watsonx, Hugging Face)
- Docker (optional)
- Git

Installation Steps

1. Clone the repository
2. Create and activate a virtual environment
3. Run: `pip install -r requirements.txt`
4. Create `.env` file with API credentials
5. Start backend: `uvicorn app.main:app --reload`
6. Launch frontend: `streamlit run ui/dashboard.py`
7. Access features via the dashboard

---

5. Folder Structure

```
ai_sdlc/
├── app/
│   ├── api/
│   │   ├── requirements.py
│   │   ├── codegen.py
│   │   ├── testing.py
│   │   ├── bugs.py
│   │   └── monitor.py
│   ├── models/
│   │   └── ml_models.py
│   ├── utils/
│   │   └── helpers.py
│   └── main.py
├── ui/
│   ├── dashboard.py
│   ├── pages/
│   │   ├── Requirements.py
│   │   ├── CodeGen.py
│   │   ├── Tests.py
│   │   ├── Monitor.py
│   │   └── Assistant.py
├── README.md
├── .env
└── requirements.txt
```

---

6. Running the Application

1. Run the FastAPI backend server
2. Launch the Streamlit frontend
3. Navigate through tabs:
    - Upload requirement documents
    - Generate code or tests
    - View performance forecasts or anomalies
4. All operations return real-time responses via API

---

7. API Documentation

Endpoint	Method	Description
`/parse-requirements`	POST	Extracts structured requirements from plain text
`/generate-code`	POST	Returns Python code for described modules
`/generate-tests`	POST	Returns unit tests for uploaded code
`/detect-bugs`	POST	Analyzes code/log files for bugs or anomalies
`/forecast-performance`	GET	Predicts system health metrics
`/chat`	POST	AI assistant for SDLC-related queries

Swagger UI: http://localhost:8000/docs

---

8. Authentication

- **Demo Mode:** Open access
- **Production Mode:**
  - JWT-based authentication
  - Role-based access (admin, developer, tester)
  - API key usage for external services
  - OAuth2 / SSO for enterprises

---

9. User Interface

Key features:

- Sidebar navigation
- Tabbed layout for each SDLC phase
- Real-time monitoring charts
- AI assistant for code/test Q&A
- Syntax highlighting for code and tests
- Export option for generated code and tests

10. Testing

Testing methodology:

- **Unit Tests:** API and model functions
- **API Tests:** Swagger/Postman validation
- **Mock Testing:** Dummy requirement sets
- **Error Handling:** Invalid inputs, timeouts, unsupported formats
- **CI/CD:** GitHub Actions workflow for automated testing

11. Screenshots

(Placeholders – add during implementation)

- Dashboard view
- AI chat assistant
- Code generator output
- Test case viewer
- Monitoring dashboard

12. Known Issues

- Occasional hallucinations in code generation with vague prompts
- Limited performance with very large documents (chunking required)
- Anomaly detection accuracy depends on labeled datasets
- Chat assistant lacks full context continuity

13. Future Enhancements

- GitHub/GitLab integration for auto-commit and pull
- AI-based CI pipeline generator
- AI-assisted refactoring recommendations
- Auto-documentation from source code
- Multilingual support
- Domain-specific fine-tuning (healthcare, finance, etc.)