# Project Report

**Title:** AI-SDLC – Smart Software Development Lifecycle with Artificial Intelligence

---

## 1. Introduction

**Project Title:** Smart AI-SDLC – AI-Driven Software Development Lifecycle

**Team Members:**

1. D. Mamtha
2. M. Malini
3. R. Keerthika
4. S. Keerthana
5. S. Anbumani

---

## 2. Project Overview

### Purpose

The goal of AI-SDLC is to transform the conventional Software Development Lifecycle by embedding artificial intelligence into its core phases. Instead of following a manual and time-consuming approach, the system automates requirement gathering, coding, testing, deployment, and maintenance with the help of AI models.

This system supports:

- Automatic requirement extraction from documents
- AI-based code generation
- Intelligent test case creation
- Error and anomaly detection
- Predictive monitoring for DevOps teams

The project acts like a **virtual assistant for developers, testers, and project managers** by reducing effort, improving accuracy, and speeding up delivery.

### Key Features

1. **Requirement Analyzer (AI-Powered)**
    - Uses NLP to convert user stories, emails, or raw documents into structured requirements
2. **Code Generator (AI-Assisted)**
    - Generates Python boilerplate and module code from structured prompts using LLMs like Codex/CodeGen
3. **Test Case Writer**
    - Produces unit and integration test cases by analyzing the uploaded codebase
4. **Bug and Error Detector**
    - Scans code and log files for anomalies using ML techniques like Isolation Forest and LSTM
5. **Predictive Monitoring Dashboard**
    - Uses time-series forecasting models (ARIMA/Prophet) to predict downtime or system slowdowns
6. **Conversational AI Assistant**
    - Answers queries, explains code, suggests improvements, and supports DevOps activities

---

## 3. System Architecture

### Frontend (Streamlit)

- User-friendly web interface with tab-based navigation
- Modules include: Requirement Upload, Code Generation, Test Writer, Bug Detection, Monitoring Dashboard
- Sidebar menu created using `streamlit-option-menu`

### Backend (FastAPI)

- REST APIs for requirement analysis, code/test generation, and anomaly detection
- All endpoints are asynchronous for better performance
- Swagger documentation enabled

### AI/ML Models Integrated

- **NER Models:** Requirement extraction
- **Codex/CodeGen:** Code snippet generation
- **TestBERT:** Test case generation

- **Isolation Forest / LSTM:** Anomaly detection
- **ARIMA / Prophet:** Predictive monitoring

---

## 4. Setup Instructions

### Requirements

- Python 3.9 or above
- API keys (OpenAI, IBM Watsonx, Hugging Face)
- Git installed
- Docker (optional)

### Steps to Run

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

---

## 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. Start the backend server using FastAPI
2. Open the Streamlit frontend in browser
3. Use the following workflow:
   - Upload requirement documents
   - Generate code snippets or tests
   - Monitor system performance forecasts
   - View detected bugs/anomalies

---

## 7. API Documentation

| Endpoint | Method | Function |
|---|---|---|
| `/parse-requirements` | POST | Extract requirements from plain text |
| `/generate-code` | POST | Produces Python code for given module |
| `/generate-tests` | POST | Creates test cases for uploaded code |
| `/detect-bugs` | POST | Identifies errors in code/logs |
| `/forecast-performance` | GET | Predicts server and system health metrics |
| `/chat` | POST | AI assistant for SDLC support |

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

---

## 8. Authentication

- **For Demo:** Open access
- **For Production:**
  - Role-based user access (Admin, Developer, Tester)
  - JWT tokens for authentication
  - Enterprise login with OAuth2/SSO

    - API keys for third-party integrations

---

## 9. User Interface

- Sidebar for navigation
- Tabbed layout for each SDLC phase
- AI assistant for interactive help
- Charts for real-time system monitoring
- Syntax-highlighted code and test viewer
- Download option for generated files

---

## 10. Testing Strategy

- **Unit Testing:** Individual functions and models
- **API Testing:** Swagger/Postman based validation
- **Mock Testing:** Dummy requirement documents used
- **Error Handling:** Invalid file formats, timeouts
- **CI/CD Integration:** GitHub Actions for automation

---

## 11. Screenshots

*(To be added after implementation)*

- Dashboard view
- Requirement Analyzer
- AI Assistant chat screen
- Test case generator output
- Monitoring dashboard

---

## 12. Known Limitations

- Sometimes AI generates irrelevant code for vague inputs
- Handling of very large requirement files is slow
- Anomaly detection model requires labeled data for accuracy
- Chat assistant has limited memory of previous queries

## 13. Future Scope

- Direct GitHub/GitLab integration for auto commits
- AI-powered CI/CD pipeline creation
- Refactoring suggestions from AI models
- Auto-documentation feature
- Multi-language interface support
- Fine-tuned AI models for industry-specific applications