SmartSDLC - AI-Enhanced Software Development Lifecycle

SmartSDLC is an intelligent AI-powered platform that automates the various phases of the Software Development Lifecycle (SDLC) using IBM Watsonx, LangChain, Streamlit, and FastAPI. It empowers users to accelerate software development through intelligent requirement classification, code generation, test automation, bug fixing, and much more.

Team Information

Team ID: LTVIP2025TMID38128
Team Size: 4
Team Leader: Batta Pavan Kumar
Team Members:

- Rasmee Ramya Bhavana

- Burlagadda Rajendra

- Aswitha Kannoju

This project was developed as part of a collaborative academic/innovation initiative, combining expertise in AI, software engineering, and full-stack development.

Problem Statement

Traditional software development is time-consuming, error-prone, and manually intensive. Developers spend approximately 70% of their time on repetitive tasks such as:

- Manual requirement analysis and documentation

- Writing boilerplate code and test cases

- Debugging and fixing common errors

- Creating technical documentation

- Managing SDLC workflows

SmartSDLC addresses these challenges by leveraging generative AI to streamline and automate critical SDLC tasks, reducing development time by up to 60% while improving code quality and consistency.

Core Architecture

SmartSDLC follows a modern microservices architecture with the following components:

Backend Architecture

- **FastAPI Server**: High-performance API server for handling requests

- **IBM Watsonx Integration**: Advanced AI model for code generation and analysis

- **LangChain Framework**: Orchestrates AI workflows and prompt engineering

- **PDF Processing Engine**: Extracts and classifies requirements from documents

- **Authentication System**: JWT-based secure user management

Frontend Architecture

- **React + TypeScript**: Modern, type-safe user interface

- **Streamlit Dashboard**: Alternative Python-based interface for quick prototyping

- **Tailwind CSS**: Utility-first styling for responsive design

- **Real-time Chat**: WebSocket-based AI assistant integration

AI/ML Pipeline

- **Natural Language Processing**: Advanced text analysis and classification

- **Code Generation Models**: Multi-language code synthesis

- **Bug Detection Algorithms**: Pattern recognition for common errors

- **Test Case Automation**: Intelligent test scenario generation

Features & Functionalities

Feature	Description	Technology Stack
Requirement Analysis	Extracts SDLC phases from uploaded PDF requirements and generates structured user stories	IBM Watsonx + NLP
AI Code Generator	Generates production-ready code from natural language descriptions in 5+ programming languages	GPT Models + LangChain
Intelligent Test Case Generation	Produces comprehensive test cases with edge cases and error conditions	AI Pattern Recognition
Smart Bug Fixer	Detects, analyzes, and resolves bugs with detailed explanations	Static Analysis + AI
Code Summarizer	Converts code into readable documentation and technical specifications	Documentation AI
AI Chatbot Assistant	Floating chatbot providing real-time SDLC guidance and best practices	LangChain + Watsonx

Feature	Description	Technology Stack
Advanced Feedback System	Multi-dimensional feedback collection with sentiment analysis	Analytics Engine
GitHub Integration	Automated code push, issue creation, and documentation sync	GitHub API
Project Management	Task tracking, milestone management, and progress analytics	Custom Dashboard
Code Quality Metrics	Automated code review with quality scores and suggestions	Static Analysis Tools

Advanced Features

1. Intelligent Requirement Classification

- **Multi-format Support**: PDF, DOCX, TXT file processing

- **SDLC Phase Detection**: Automatic categorization into Requirements, Design, Development, Testing, Deployment

- **User Story Generation**: Converts raw requirements into Agile user stories

- **Traceability Matrix**: Links requirements to code and test cases

2. Multi-Language Code Generation

Supported Languages:

- Python (Flask, Django, FastAPI)

- JavaScript/TypeScript (React, Node.js, Express)

- Java (Spring Boot, Maven projects)

- C++ (Standard Library, Modern C++)

- C# (.NET Core, ASP.NET)

- Go (Gin, Echo frameworks)

- Rust (Actix, Rocket frameworks)

Code Quality Features:

- Clean, commented, production-ready code

- Best practices implementation

- Security vulnerability scanning

- Performance optimization suggestions

3. Comprehensive Testing Suite

- **Unit Test Generation**: Framework-specific test cases (Jest, pytest, JUnit)

- **Integration Test Scenarios**: API and database testing

- **Performance Test Cases**: Load and stress testing templates

- **Security Test Cases**: Vulnerability and penetration testing

4. Advanced Bug Detection & Resolution

- **Static Code Analysis**: Syntax and logic error detection

- **Runtime Error Prediction**: Potential runtime issue identification

- **Performance Bottleneck Detection**: Code optimization suggestions

- **Security Vulnerability Scanning**: Common security flaw identification

Technical Specifications

System Requirements

- **Backend**: Python 3.10+, 4GB RAM minimum, 8GB recommended

- **Frontend**: Node.js 18+, npm 8+

- **Database**: SQLite (development), PostgreSQL (production)

- **AI Services**: IBM Watsonx API access, OpenAI API (optional)

Performance Metrics

- **Code Generation**: ~3-5 seconds for 100 lines of code

- **Bug Fixing**: ~2-4 seconds for common issues

- **Test Generation**: ~5-8 seconds for comprehensive test suites

- **PDF Processing**: ~10-15 seconds for 50-page documents

Security Features

- **JWT Authentication**: Secure token-based authentication

- **API Rate Limiting**: Prevents abuse and ensures fair usage

- **Input Validation**: Comprehensive input sanitization

- **Data Encryption**: End-to-end encryption for sensitive data

- **Audit Logging**: Complete activity tracking and monitoring

Installation & Setup

Prerequisites

System Requirements

Python 3.10 or higher

Node.js 18+ and npm

IBM Watsonx AI account and API key

Git for version control

Quick Start Installation

1. Clone the repository

```
git clone <repository-url>

cd SmartSDLC


# 2. Backend Setup

python -m venv venv

source venv/bin/activate  # Linux/Mac

# venv\Scripts\activate   # Windows


pip install -r requirements.txt


# 3. Frontend Setup

npm install


# 4. Environment Configuration

cp .env.example .env

# Edit .env with your IBM Watsonx credentials


# 5. Database Setup

python manage.py migrate  # If using Django ORM
```

```
# Or setup SQLite database
```

```
# 6. Start Services

python run_backend.py    # Terminal 1

python run_frontend.py   # Terminal 2

npm run dev              # Terminal 3 (for React)
```

Docker Deployment

```
# Build and run with Docker Compose

docker-compose up --build
```

```
# Access services:

# - Streamlit: http://localhost:8501

# - React: http://localhost:3000

# - API: http://localhost:8000
```

API Documentation

Authentication Endpoints

```
POST /auth/register    - User registration

POST /auth/login       - User authentication

POST /auth/refresh     - Token refresh
```

DELETE /auth/logout   - User logout

AI Service Endpoints

POST /ai/upload-pdf   - PDF requirement analysis

POST /ai/generate-code   - Code generation

POST /ai/fix-bugs   - Bug detection and fixing

POST /ai/generate-tests   - Test case generation

POST /ai/summarize-code   - Code documentation

POST /ai/analyze-quality   - Code quality assessment

Project Management Endpoints

GET /projects   - List user projects

POST /projects   - Create new project

PUT /projects/{id}   - Update project

DELETE /projects/{id}   - Delete project

GET /projects/{id}/stats   - Project analytics

Usage Examples

1. Code Generation Example

Input: Natural language description

"Create a REST API endpoint for user authentication with JWT tokens"

```python
# Output: Complete FastAPI implementation
@app.post("/auth/login")
async def login(credentials: UserCredentials):
    user = authenticate_user(credentials.username, credentials.password)
    if not user:
        raise HTTPException(status_code=401, detail="Invalid credentials")

    access_token = create_access_token(data={"sub": user.username})
    return {"access_token": access_token, "token_type": "bearer"}
```

2. Test Generation Example

```python
# Input: Function to test
def calculate_discount(price, discount_percent):
    return price * (1 - discount_percent / 100)


# Output: Comprehensive test suite
```

```python
def test_calculate_discount():
    # Normal cases
    assert calculate_discount(100, 10) == 90.0
    assert calculate_discount(50, 20) == 40.0

    # Edge cases
    assert calculate_discount(100, 0) == 100.0
    assert calculate_discount(100, 100) == 0.0

    # Error conditions
    with pytest.raises(ValueError):
        calculate_discount(-100, 10)
```

Configuration Options

Environment Variables

```
# IBM Watsonx Configuration
WATSONX_API_KEY=your_api_key_here
WATSONX_PROJECT_ID=your_project_id
WATSONX_URL=https://eu-de.ml.cloud.ibm.com
```

```
# Application Configuration

SECRET_KEY=your-secret-key

API_HOST=0.0.0.0

API_PORT=8000

STREAMLIT_HOST=0.0.0.0

STREAMLIT_PORT=8501


# Database Configuration

DATABASE_URL=sqlite:///./smartsdlc.db

# DATABASE_URL=postgresql://user:pass@localhost/smartsdlc
# Production


# External Services

GITHUB_TOKEN=your_github_token  # For GitHub integration

OPENAI_API_KEY=your_openai_key  # Optional, for enhanced
AI features


# Feature Flags

ENABLE_GITHUB_INTEGRATION=true

ENABLE_ADVANCED_ANALYTICS=true
```

ENABLE_REAL_TIME_COLLABORATION=false

Monitoring & Analytics

Built-in Dashboards

- **Usage Analytics**: Track feature usage and user engagement

- **Performance Metrics**: Monitor response times and system health

- **Code Quality Trends**: Analyze generated code quality over time

- **User Feedback Analysis**: Sentiment analysis and improvement insights

Integration Options

- **Prometheus**: Metrics collection and monitoring

- **Grafana**: Advanced dashboard visualization

- **ELK Stack**: Centralized logging and analysis

- **Sentry**: Error tracking and performance monitoring

Security & Compliance

Security Measures

- **OWASP Compliance**: Following top 10 security practices

- **Data Privacy**: GDPR and CCPA compliant data handling

- **API Security**: Rate limiting, input validation, SQL injection prevention
- **Code Security**: Automated vulnerability scanning for generated code

Compliance Features

- **Audit Trails**: Complete activity logging
- **Data Retention**: Configurable data retention policies
- **Access Controls**: Role-based access management
- **Encryption**: AES-256 encryption for sensitive data

Future Roadmap

Phase 1 (Q2 2025)

- Advanced GitHub workflow automation
- Multi-project workspace management
- Enhanced collaboration features
- Mobile application development

Phase 2 (Q3 2025)

- Machine learning model customization
- Advanced code refactoring capabilities
- Integrated development environment (IDE) plugins
- Enterprise-grade deployment options

Phase 3 (Q4 2025)

- Natural language to database query conversion

- Automated API documentation generation

- Advanced security scanning and remediation

- Multi-tenant architecture support

Support & Contributing

Getting Help

- **Documentation**: Comprehensive guides at /docs

- **API Reference**: Interactive API docs at /docs/api

- **Community**: Join our Discord/Slack community

- **Issues**: Report bugs on GitHub Issues

Contributing Guidelines

1. Fork the repository

2. Create a feature branch (git checkout -b feature/amazing-feature)

3. Commit your changes (git commit -m 'Add amazing feature')

4. Push to the branch (git push origin feature/amazing-feature)

5. Open a Pull Request

Code Standards

- **Python**: Follow PEP 8 style guidelines

- **JavaScript/TypeScript**: ESLint and Prettier configuration

- **Testing**: Maintain 80%+ code coverage

- **Documentation**: Document all public APIs and functions

License & Acknowledgments

License

This project is licensed under the MIT License - see the LICENSE file for details.

Acknowledgments

- IBM Watsonx team for AI platform support

- LangChain community for framework contributions

- Streamlit team for rapid prototyping capabilities

- FastAPI developers for high-performance API framework

Third-Party Libraries

- **IBM Watsonx**: AI model integration

- **LangChain**: AI workflow orchestration

- **Streamlit**: Rapid web app development

- **FastAPI**: Modern Python web framework

- **React**: User interface library

- **Tailwind CSS**: Utility-first CSS framework

SmartSDLC - Revolutionizing software development with AI!

Built with ❤️ *by Team LTVIP2025TMID38128*