# SmartSDLC – AI-Enhanced Software Development Lifecycle

Team ID : NM2025TMID12129

Team Size : 4

1.Team Leader : AL JESIRA J

2.Team member : JAYANTHI J

3.Team member : DEVI SRI T

4.Team member : ALSANI BEEVI M

## 1.Introduction

The **SmartSDLC (AI-Enhanced Software Development Lifecycle)** project is designed to transform traditional software engineering by embedding **Generative AI** into every stage of the SDLC. It provides intelligent automation for tasks such as requirement analysis, code generation, test case creation, bug fixing, documentation, and real-time chatbot assistance.

Two complementary approaches have been developed:

1. **Enterprise-Grade Platform (DOCX version)**
    - Built with **FastAPI** (backend) and **Streamlit** (frontend), integrated with **IBM Watsonx Granite models** and **LangChain**.
    - Supports modular microservices, requirement classification from PDFs, multilingual code generation, AI-driven testing, smart bug fixing, GitHub automation, and continuous feedback loops.
    - Designed for **scalability, modularity, and production deployment**, suitable for professional developers and project teams.
2. **Student-Friendly Prototype (PDF version)**
    - Runs in **Google Colab** with **Gradio UI**, leveraging lightweight **IBM Granite models** from Hugging Face.
    - Focused on quick prototyping and ease of learning, allowing students to upload PDFs, generate code, run tests, and interact with an AI helper in a simplified environment.

- Designed for **educational use and rapid experimentation**, with GitHub integration for project submission.

Together, these implementations illustrate how **AI can streamline SDLC processes** in different contexts—from **lightweight training prototypes** to **enterprise-grade automation platforms**. By reducing manual effort, enhancing accuracy, and accelerating development cycles, SmartSDLC demonstrates the future of AI-driven software engineering.

## 2.Project Overview

The **SmartSDLC (AI-Enhanced Software Development Lifecycle)** project leverages **Generative AI** to automate and streamline key phases of the software development process. Its goal is to reduce manual effort, improve code quality, and accelerate delivery timelines by embedding AI-driven intelligence into each stage of the SDLC.

🔹 **Core Objectives**

- **Requirement Analysis** – Convert unstructured requirements (e.g., PDFs) into structured user stories and SDLC phase classifications.
- **Code Generation** – Transform natural language prompts into clean, production-ready code across multiple programming languages.
- **Test Case Creation** – Automatically generate unit and integration tests aligned with the generated or existing code.
- **Bug Fixing & Optimization** – Detect and correct both syntactic and logical errors in code, returning optimized solutions.
- **Documentation & Summarization** – Produce human-readable explanations and technical documentation from code.
- **AI Chatbot Assistance** – Provide interactive guidance, answering SDLC-related queries and assisting in workflow navigation.
- **Feedback & Continuous Learning** – Capture user feedback to refine prompts, outputs, and overall system performance.
- **Version Control Integration** – Enable seamless GitHub workflows for code storage, issue tracking, and project synchronization.

🔹 **Dual Implementation Approaches**

1. **Enterprise-Grade Platform (DOCX Version)**
    - **Tech Stack**: FastAPI (backend), Streamlit (frontend), IBM Watsonx Granite models, LangChain, GitHub API.
    - **Deployment**: Local hosting with Uvicorn & Streamlit, API documentation via Swagger UI.

- **Audience**: Professional developers, project teams, and organizations seeking scalable automation.

2. **Student-Friendly Prototype (PDF Version)**
    - **Tech Stack**: Gradio UI in Google Colab, Hugging Face IBM Granite models, Python libraries (Transformers, Torch, PyPDF2).
    - **Deployment**: Cloud-based, runs in Google Colab with GPU support.
    - **Audience**: Students, interns, and beginners focusing on hands-on learning and rapid prototyping.

🔹 **Impact**

SmartSDLC demonstrates how **AI can enhance modern software engineering practices** by:

- Reducing the manual workload of developers.
- Improving accuracy and consistency in requirement handling, coding, and testing.
- Supporting agile methodologies through faster iteration cycles.
- Enabling both **educational training environments** and **enterprise-ready production systems**.

**Purpose**

The primary purpose of **SmartSDLC (AI-Enhanced Software Development Lifecycle)** is to **integrate Artificial Intelligence into every phase of the SDLC** in order to:

- **Automate repetitive tasks** such as requirement analysis, code writing, testing, bug fixing, and documentation.
- **Accelerate software delivery** by reducing manual effort and enabling rapid prototyping.
- **Improve accuracy and quality** through AI-driven validation and optimization of code and requirements.
- **Support diverse audiences** by providing:
    - A **scalable enterprise-grade solution** for professional developers and organizations.
    - A **lightweight, student-friendly prototype** for learners and quick experimentation.

---

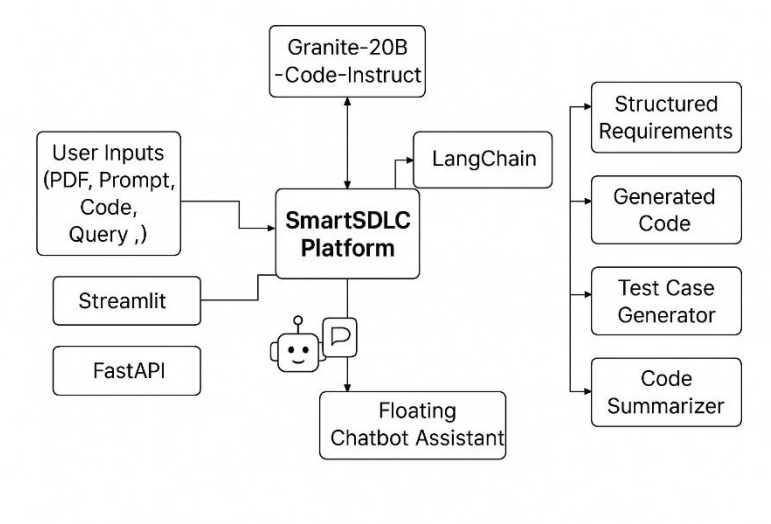
**Key Features**

🔹 **Common Features (Across Both Versions)**

- **Requirement Upload & Classification** – Upload PDF documents and automatically classify requirements into SDLC phases (Requirements, Design, Development, Testing, Deployment).
- **AI-Powered Code Generation** – Convert natural language prompts or user stories into executable, production-ready code.
- **Automated Test Case Creation** – Generate unit tests and integration tests aligned with the provided or AI-generated code.
- **Bug Detection & Fixing** – Identify and correct logical or syntactic errors in code snippets instantly.
- **Documentation & Code Summarization** – Create human-readable technical documentation and summaries for better maintainability.
- **AI Chatbot Assistance** – Provide real-time guidance and support for SDLC tasks, helping users navigate the platform.

# 3.Architecture

The **SmartSDLC architecture** is designed to embed **AI-powered intelligence** into every stage of the Software Development Lifecycle. It follows a **modular and layered approach**, ensuring scalability, flexibility, and usability across both **enterprise** and **student-friendly** implementations.




**1. Enterprise-Grade Architecture (DOCX Version)**

**Layers and Components**

- **Frontend (Streamlit UI)**
    - Interactive dashboard with feature-specific pages (requirement analysis, code generator, bug fixer, etc.).

    - Floating AI chatbot for user assistance.
    - Displays AI responses with syntax highlighting, structured outputs, and summaries.
- **Backend (FastAPI)**
    - **API Routing**: Endpoints for requirement upload, code generation, bug fixing, summarization, testing, chatbot, authentication, and feedback.
    - **Service Layer**: Encapsulates business logic for AI-powered functionalities (requirement analysis, code generator, bug resolver, doc generator, etc.).
    - **Authentication & Security**: User login/registration, password hashing, session validation.
    - **Feedback Management**: Collects user feedback for improving AI responses.
- **AI Layer**
    - **IBM Watsonx Granite Models**:
        - granite-13b-chat-v1 → natural language understanding & requirement classification.
        - granite-20b-code-instruct → code generation & bug fixing.
    - **LangChain**: Manages prompt templates, conversational memory, and AI response chaining.
- **Integration Layer**
    - **PyMuPDF**: Extracts text from PDF requirements.
    - **GitHub Service**: Automates commits, issue creation, and documentation sync.
    - **Swagger UI**: Provides API documentation for testing and developer access.
- **Deployment**
    - Local hosting with **Uvicorn (FastAPI)** and **Streamlit**.
    - Environment configuration via .env files for API keys and credentials.

---

**2. Student-Friendly Architecture (PDF Version)**

**Layers and Components**

- **Frontend (Gradio UI in Google Colab)**
    - Simple, browser-based interface for uploading PDFs, entering prompts, and interacting with AI.
    - Real-time display of generated code, tests, and bug fixes.
- **Backend (Colab Notebook)**

    - Implemented directly in **Python (Colab cells)**, no microservices.
    - Uses Hugging Face **IBM Granite lightweight model (granite-3.2-2b-instruct)** for quick response.
    - Supports requirement extraction (via **PyPDF2**), code generation, bug fixing, and documentation.
- **AI Layer**
    - Hugging Face **Transformers** + **IBM Granite models** for NLP and code tasks.
    - **Torch** for model execution with GPU acceleration (T4 GPU in Colab).
- **Integration Layer**
    - **Google Colab GPU Runtime** for fast execution.
    - **GitHub Upload**: Export .py project files and push to GitHub manually.
- **Deployment**
    - Cloud-hosted via **Google Colab**, eliminating setup complexity.
    - Runs interactively with minimal prerequisites.

---

**3. Comparative View**

| Component | Enterprise (DOCX) | Student-Friendly (PDF) |
|---|---|---|
| **Frontend** | Streamlit (rich dashboard) | Gradio (lightweight UI) |
| **Backend** | FastAPI (modular microservices) | Google Colab notebook |
| **AI Models** | IBM Watsonx Granite (13B, 20B) | Hugging Face Granite (2B) |
| **Integration** | GitHub automation, Swagger UI, LangChain | Manual GitHub upload, Colab runtime |
| **Deployment** | Local hosting (FastAPI + Streamlit + Uvicorn) | Cloud-based (Colab with GPU) |
| **Target Users** | Professional developers, teams | Students, beginners, interns |

## 4.setup Instructions

**1. Enterprise-Grade Platform (DOCX Version)**

**Pre-requisites**

- Python 3.10 → Download
- FastAPI → Docs

- Streamlit → Docs
- IBM Watsonx AI & Granite Models → IBM Watsonx
- LangChain → Docs
- Uvicorn → Docs
- PyMuPDF → Docs
- Git & GitHub → Docs

**Step-by-Step Setup**

1. **Clone the Repository**
2. git clone <your_repo_link>
3. cd SmartSDLC
4. **Create a Virtual Environment**
5. python -m venv venv
6. source venv/bin/activate  # Linux/Mac
7. venv\Scripts\activate  # Windows
8. **Install Dependencies**
9. pip install -r requirements.txt
10. **Set Environment Variables**
    Create a .env file in the project root:
11. WATSONX_API_KEY=your_ibm_key_here
12. WATSONX_PROJECT_ID=your_project_id
13. WATSONX_MODEL_ID=granite-20b-code-instruct
14. **Run Backend (FastAPI)**
15. uvicorn app.main:app --reload

Access API docs → http://127.0.0.1:8000/docs

16. **Run Frontend (Streamlit)**
17. streamlit run frontend/Home.py

Access UI → http://localhost:8502

---

**2. Student-Friendly Prototype (PDF Version)**

**Pre-requisites**

- Google Account for Colab access

- Hugging Face Account → Sign Up
- Python basics and GitHub knowledge

**Step-by-Step Setup**

1. **Open Google Colab**
    - Go to Google Colab.
    - Create a new notebook.
2. **Set Runtime Environment**
    - Click on Runtime → Change runtime type.
    - Select **T4 GPU**.
3. **Install Required Libraries**
   Run in the first Colab cell:
4. !pip install transformers torch gradio PyPDF2 -q
5. **Load IBM Granite Model from Hugging Face**
   Example code snippet in Colab:
6. from transformers import AutoModelForCausalLM, AutoTokenizer
7. 
8. model_name = "ibm-granite/granite-3.2-2b-instruct"
9. tokenizer = AutoTokenizer.from_pretrained(model_name)
10. model = AutoModelForCausalLM.from_pretrained(model_name)
11. **Build the Gradio App**
    Use Gradio to create the UI for uploading PDFs, generating code, and chatting with AI.
12. **Run the Application**
    - Run the notebook cells.
    - Colab will provide a public **Gradio link**.
    - Open the link to interact with the SmartSDLC prototype.
13. **Upload Project to GitHub**
    - Download your Colab .py file via File → Download → Download as .py.
    - Push to GitHub:
    - git init
    - git add .
    - git commit -m "SmartSDLC prototype"

- git remote add origin <your_repo_link>
- git push -u origin main

# 5. Folder Structure

### 1. Enterprise-Grade SmartSDLC (DOCX Version)

```
SmartSDLC/
├── app/                          # Backend (FastAPI)
│   ├── main.py                    # Entry point for FastAPI
│   ├── routes/                    # API route handlers
│   │   ├── ai_routes.py           # AI features (code gen, test gen, bug
fix)
│   │   ├── auth_routes.py         # Authentication & login
│   │   ├── chat_routes.py         # Chatbot endpoints
│   │   ├── feedback_routes.py     # Feedback collection
│   ├── services/                  # Core business logic
│   │   ├── ai_story_generator.py # Requirement analysis
│   │   ├── code_generator.py      # Multilingual code & test generation
│   │   ├── bug_resolver.py        # Bug fixing & optimization
│   │   ├── doc_generator.py       # Code summarization & documentation
│   │   ├── github_service.py      # GitHub integration
│   ├── models/                    # Data models & schemas
│   │   ├── user.py                # User schema
│   │   ├── feedback_model.py      # Feedback schema
│   ├── utils/
│   │   ├── watsonx_service.py     # Watsonx AI model integration
│   │   ├── security.py            # Authentication utilities
│   ├── database/
│   │   ├── feedback_data.json     # Feedback storage
│   ├── config.py                  # Env variables & project configs
│
├── frontend/                      # Streamlit frontend
│   ├── Home.py                    # Dashboard entry
│   ├── pages/                     # Modular feature pages
│   │   ├── Upload_and_Classify.py
│   │   ├── Code_Generator.py
│   │   ├── Test_Generator.py
│   │   ├── Bug_Fixer.py
│   │   ├── Code_Summarizer.py
│   │   ├── Feedback.py
│   ├── api_client.py              # Handles requests to FastAPI backend
│   ├── assets/                    # Lottie animations, CSS, images
│
├── requirements.txt               # Dependencies
├── .env                           # API keys & secrets
├── README.md                      # Documentation
```

---

### 2. Student-Friendly SmartSDLC (PDF Version)

```
SmartSDLC_Colab/
├── SmartSDLC.ipynb                # Google Colab notebook
```

```
├── smart_sdlc.py                  # Exported Python script from
Colab
│
├── modules/                       # AI & helper modules
(optional if modularized)
│   ├── requirement_extractor.py # Extracts text from PDFs
│   ├── code_generator.py        # AI-based code generator
│   ├── bug_fixer.py             # Code debugging utility
│   ├── doc_writer.py            # Documentation &
summarization
│
├── assets/                        # Sample PDFs, test code
snippets
│
├── requirements.txt               # Dependencies (transformers,
torch, gradio, PyPDF2)
├── README.md                      # Setup and usage guide
```

## 6.Running the Applications

**1. Enterprise-Grade SmartSDLC (DOCX Version)**

**Steps to Run Locally**

1. **Start the Backend (FastAPI)**
2. uvicorn app.main:app --reload
    - The backend will start at → http://127.0.0.1:8000
    - Interactive API docs → http://127.0.0.1:8000/docs
3. **Start the Frontend (Streamlit)**
4. streamlit run frontend/Home.py
    - The frontend will launch at → http://localhost:8502
5. **Access the Application**
    - Upload requirements PDF → automatically classified into SDLC phases.
    - Generate code from user stories or prompts.
    - Auto-generate test cases.
    - Paste buggy code → AI fixes and returns optimized version.
    - Summarize or document code.
    - Use chatbot for real-time SDLC assistance.
    - Submit feedback → stored in feedback_data.json.

---

**2. Student-Friendly SmartSDLC (PDF Version)**

**Steps to Run in Google Colab**

1. **Open the Notebook**
    - Go to Google Colab.
    - Upload the SmartSDLC.ipynb notebook or create a new one.
2. **Set Runtime**
    - Navigate to Runtime → Change runtime type.
    - Select **T4 GPU** for faster performance.
3. **Install Dependencies**
   Run this in the first cell:
4. !pip install transformers torch gradio PyPDF2 -q
5. **Load IBM Granite Model**
   Example snippet:
6. from transformers import AutoModelForCausalLM, AutoTokenizer
7. model_name = "ibm-granite/granite-3.2-2b-instruct"
8. tokenizer = AutoTokenizer.from_pretrained(model_name)
9. model = AutoModelForCausalLM.from_pretrained(model_name)
10. **Run the Gradio App**
11. import gradio as gr
12. 
13. def generate_code(prompt):
14.  # model inference logic here
15.  return "Generated code for: " + prompt
16. 
17. iface = gr.Interface(fn=generate_code, inputs="text", outputs="code")
18. iface.launch()
    - Colab will provide a **public link**.
    - Click the link to open the SmartSDLC UI in your browser.
19. **Test Features**
    - Upload PDFs → extract requirements.
    - Enter prompts → generate code.

- Auto-create test cases.
- Paste buggy code → AI fixes errors.
- Get summaries/documentation.
- Chat with the AI assistant.

20. **Save & Upload to GitHub**
    - Download your notebook as .py → File → Download → Download .py.
    - Push it to GitHub for project submission.

## 7.API Documentation

**1. Enterprise-Grade SmartSDLC (DOCX Version)**

The backend is built with **FastAPI**, exposing modular microservices that can be explored interactively via **Swagger UI** at:

👉 http://127.0.0.1:8000/docs

**Core Endpoints**

🔐 **Authentication**

- **POST** /auth/register → Register a new user
- **POST** /auth/login → User login with hashed passwords
- **GET** /auth/me → Get current user profile

📄 **Requirement Analysis**

- **POST** /ai/upload-pdf
    - **Input**: PDF file with requirements
    - **Output**: Structured requirements categorized by SDLC phase (Requirement, Design, Development, Testing, Deployment)

💻 **Code Generation**

- **POST** /ai/generate-code
    - **Input**: Natural language prompt or user story
    - **Output**: Clean, production-ready code (Python, JavaScript, etc.)

🧪 **Test Case Generation**

- **POST** /ai/generate-tests
    - **Input**: User-provided or AI-generated code
    - **Output**: Unit tests / Integration tests

🐞 **Bug Fixing**

- **POST** /ai/fix-bugs
    - **Input**: Buggy code snippet
    - **Output**: Corrected and optimized code with optional explanation

📄 **Code Summarization & Documentation**

- **POST** /ai/summarize-code
    - **Input**: Code snippet
    - **Output**: Documentation-style summary

🤖 **Chatbot Assistance**

- **POST** /chat/chat
    - **Input**: User query
    - **Output**: Conversational AI response (powered by Watsonx + LangChain)

📊 **Feedback Collection**

- **POST** /feedback/submit
    - **Input**: User feedback (rating/comments)
    - **Output**: Stored entry in feedback_data.json

---

**Response Format (Example: Code Generation)**

**Request**

POST /ai/generate-code

```
{
  "prompt": "Create a Python function to calculate factorial"
}
```

**Response**

```
{
  "status": "success",
  "language": "python",
  "generated_code": "def factorial(n):\n    return 1 if n == 0 else n * factorial(n-1)"
}
```

---

**2. Student-Friendly SmartSDLC (PDF Version)**

The **Colab + Gradio version** does not expose REST APIs. Instead:

- It runs as an **interactive Gradio interface**.
- Users directly interact via text inputs, file uploads, and buttons inside the browser.
- Each feature (requirement extraction, code generation, bug fixing) is mapped to a Gradio function instead of an HTTP endpoint.

Example usage in Gradio:

iface = gr.Interface(fn=generate_code, inputs="text", outputs="code")

iface.launch()

# 8.Authentication

**1. Enterprise-Grade SmartSDLC (DOCX Version)**

The enterprise-grade version implements a **secure authentication system** within the **FastAPI backend**.

**Features**

- **User Registration & Login**
    - Users can register with email/username and password.
    - Passwords are **hashed** before storage (never stored as plain text).
- **Session & Token Management**
    - Authentication is verified via session tokens or JWTs.
    - Each API request checks credentials before granting access.
- **Role-Based Access (optional extension)**
    - Certain features (e.g., GitHub integration, feedback analytics) can be restricted to authorized roles.

**Core Modules**

- auth_routes.py → Defines routes for registration, login, and validation.
- security.py → Handles hashing, token generation, and verification.
- user.py → Defines user schema (username, email, hashed password, role).

**Endpoints**

- **POST** /auth/register → Register a new user
    - Input: { "username": "john", "password": "1234" }

    - Output: {"message": "User registered successfully"}
- **POST** /auth/login → Authenticate user
    - Input: { "username": "john", "password": "1234" }
    - Output: { "access_token": "xxxx.yyyy.zzzz", "token_type": "bearer" }
- **GET** /auth/me → Retrieve current user info (requires valid token)
    - Output: { "username": "john", "role": "developer" }

---

**2. Student-Friendly SmartSDLC (PDF Version)**

The Colab + Gradio prototype is **lightweight** and **does not implement authentication** by default.

**Characteristics**

- Runs in **Google Colab** with Gradio, accessible via a temporary public link.
- No built-in user accounts or login system.
- Security relies on Colab/Gradio link sharing permissions.

**Optional Extension for Authentication**

If required for student projects:

- Use **Gradio authentication decorators**:
- iface = gr.Interface(
- fn=generate_code,
- inputs="text",
- outputs="code",
- auth=("username", "password")
- )
- iface.launch()
- Or add a **simple login cell in Colab** that checks credentials before running main code.

---

**3. Comparative View**

| Feature | Enterprise (DOCX) | Student (PDF) |
|---|---|---|
| User Accounts | ✅ Yes (FastAPI + DB) | ❌ No (Colab/Gradio only) |
| Password Security | ✅ Hashed storage | ❌ Not implemented |

| Feature | Enterprise (DOCX) | Student (PDF) |
|---|---|---|
| Token/Session Mgmt | ✅ JWT/Bearer tokens | ❌ Not available |
| Role-based Access | ✅ Possible | ❌ Not supported |
| Target Audience | Professional teams | Students / learners |

# 9.User Interface

**1. Enterprise-Grade SmartSDLC (DOCX Version)**

The **frontend** is built with **Streamlit**, providing a clean, modular dashboard for interacting with AI-powered SDLC features.

**UI Components**

- **Home Dashboard (Home.py)**
    - Hero section with project title, tagline, and animation.
    - Navigation grid linking to core features.
    - Embedded floating chatbot for real-time assistance.
- **Feature-Specific Pages (pages/ directory)**
    - **Upload_and_Classify.py** → Upload PDF → AI classifies requirements into SDLC phases.
    - **Code_Generator.py** → Enter natural language prompts → AI generates code.
    - **Test_Generator.py** → Auto-generate unit/integration tests.
    - **Bug_Fixer.py** → Paste buggy code → AI fixes errors.
    - **Code_Summarizer.py** → Summarize and document code snippets.
    - **Feedback.py** → Collect user feedback with forms.
- **Interactive Elements**
    - **File Upload Widgets** → Upload PDF requirements.
    - **Text Areas** → Enter prompts or paste code snippets.
    - **AI Outputs** → Displayed with syntax highlighting (st.code()), success messages, or markdown.
    - **Chatbot Box** → Conversational assistant powered by LangChain + Watsonx.

**UI Example Workflow**

1. User uploads a PDF requirement.
2. Output displayed → categorized SDLC tasks with user stories.

3. User selects “Generate Code” → enters requirement → AI generates Python code.
4. User runs “Bug Fixer” → pastes buggy code → AI returns corrected version.
5. Chatbot answers questions in real time.

---

**2. Student-Friendly SmartSDLC (PDF Version)**

The **frontend** is implemented with **Gradio** inside **Google Colab**, offering a simple browser-based interface.

**UI Components**

- **Gradio Blocks / Interface**
    - **Requirement Upload** → File upload field for PDFs.
    - **Prompt Input** → Text box to enter natural language requirements.
    - **Code Output** → Syntax-highlighted code block.
    - **Test Generation** → Button to generate test cases.
    - **Bug Fixer** → Input for buggy code + output with corrected version.
    - **Chatbot** → Text-based Q&A interface with AI assistant.
- **Interaction Flow**

1. User uploads a PDF in the Gradio file uploader.
2. Model extracts requirements → displays structured text output.
3. User enters prompt → AI generates corresponding code.
4. User tests code or fixes bugs with one-click buttons.
5. Documentation summaries shown in plain text.

---

**3. Comparative UI View**

| Feature | Enterprise (DOCX) | Student (PDF) |
|---|---|---|
| Framework | Streamlit | Gradio |
| Dashboard | ✅ Rich, multi-page | ❌ Minimal |
| File Upload | ✅ Yes (PDFs) | ✅ Yes (PDFs) |
| Code Display | ✅ Syntax highlighted | ✅ Syntax highlighted |
| Chatbot | ✅ Floating chatbot | ✅ Simple text chat |

| Feature | Enterprise (DOCX) | Student (PDF) |
|---|---|---|
| Feedback | ✅ Integrated form | ❌ Not included |
| Audience | Professionals | Students/learners |

# 10.Testing

**1. Enterprise-Grade SmartSDLC (DOCX Version)**

The enterprise version integrates **AI-driven test generation** and allows **manual + automated testing** of its backend and frontend.

**AI-Powered Test Case Generation**

- **Module**: code_generator.py (test generation feature).
- **Functionality**:
  - Takes generated or user-provided code.
  - Produces **unit tests** and **integration tests** automatically.
  - Uses IBM Watsonx Granite model (granite-20b-code-instruct).

**Example:**

- Input → Python function:
- def add(a, b):
-  return a + b
- Output → AI-generated test:
- import unittest
-
- class TestAdd(unittest.TestCase):
-  def test_add(self):
-  self.assertEqual(add(2, 3), 5)
-  self.assertEqual(add(-1, 1), 0)
-  self.assertEqual(add(0, 0), 0)

**Backend Testing**

- Run FastAPI tests with pytest or unittest.
- Example test for API endpoint (tests/test_api.py):
- from fastapi.testclient import TestClient
- from app.main import app

- 
- client = TestClient(app)
- 
- def test_generate_code():
- response = client.post("/ai/generate-code", json={"prompt": "print hello"})
- assert response.status_code == 200
- assert "generated_code" in response.json()

**Frontend Testing (Streamlit)**

- UI tested by interacting with widgets (PDF upload, text inputs).
- AI outputs validated manually (e.g., correct requirement classification).
- Automated testing possible using **Streamlit testing utilities**.

---

**2. Student-Friendly SmartSDLC (PDF Version)**

The Colab + Gradio version also supports **AI-driven test case generation**, but testing is simpler.

**AI Test Generation in Colab**

- Uses Hugging Face Granite model (granite-3.2-2b-instruct).
- Example prompt →
- Write unit tests for a function that checks if a number is prime.
- Output →
- import unittest
- 
- class TestPrime(unittest.TestCase):
- def test_prime(self):
- self.assertTrue(is_prime(7))
- self.assertFalse(is_prime(10))
- self.assertTrue(is_prime(13))

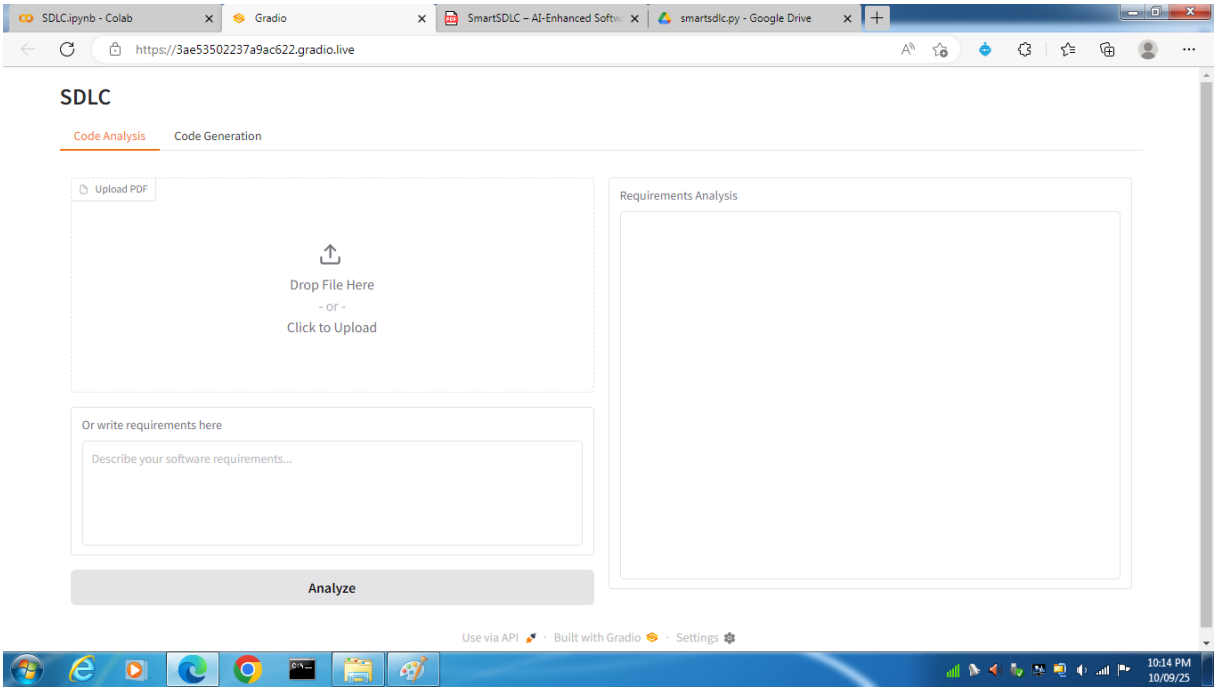
**Notebook Testing**

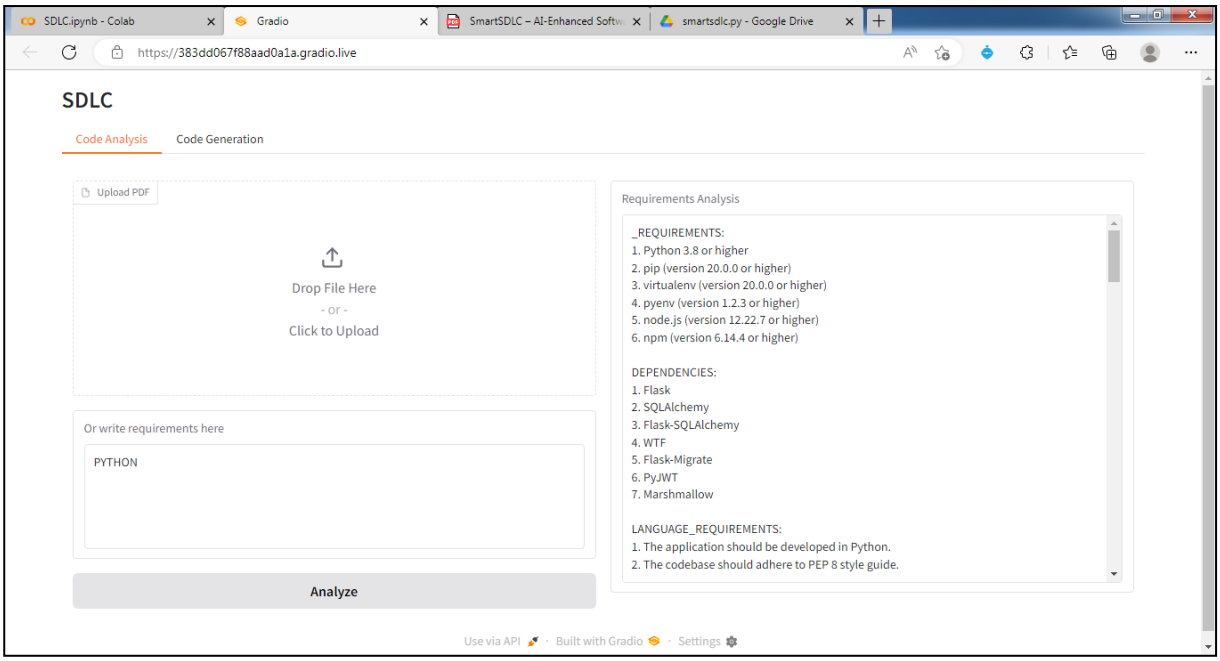
- Users can directly run generated tests inside Colab cells.
- No dedicated backend testing (since it’s not microservice-based).
- Gradio UI tested by running interactions (file upload, prompt input, chatbot Q&A).

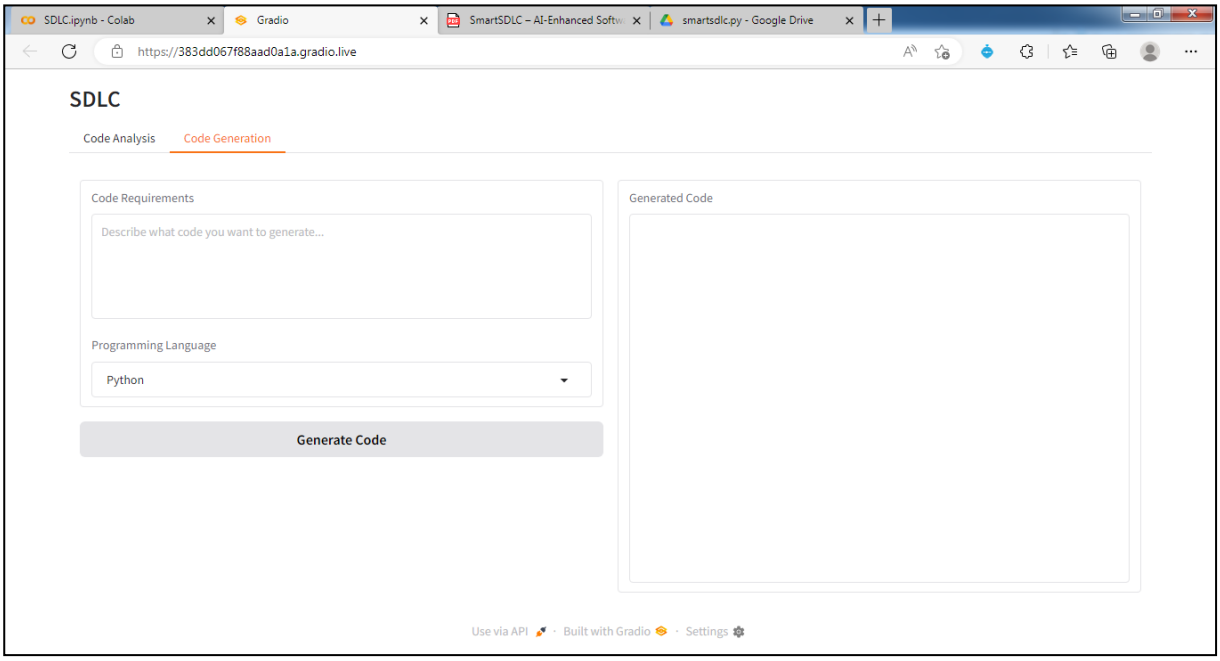
### 3. Comparative Testing View

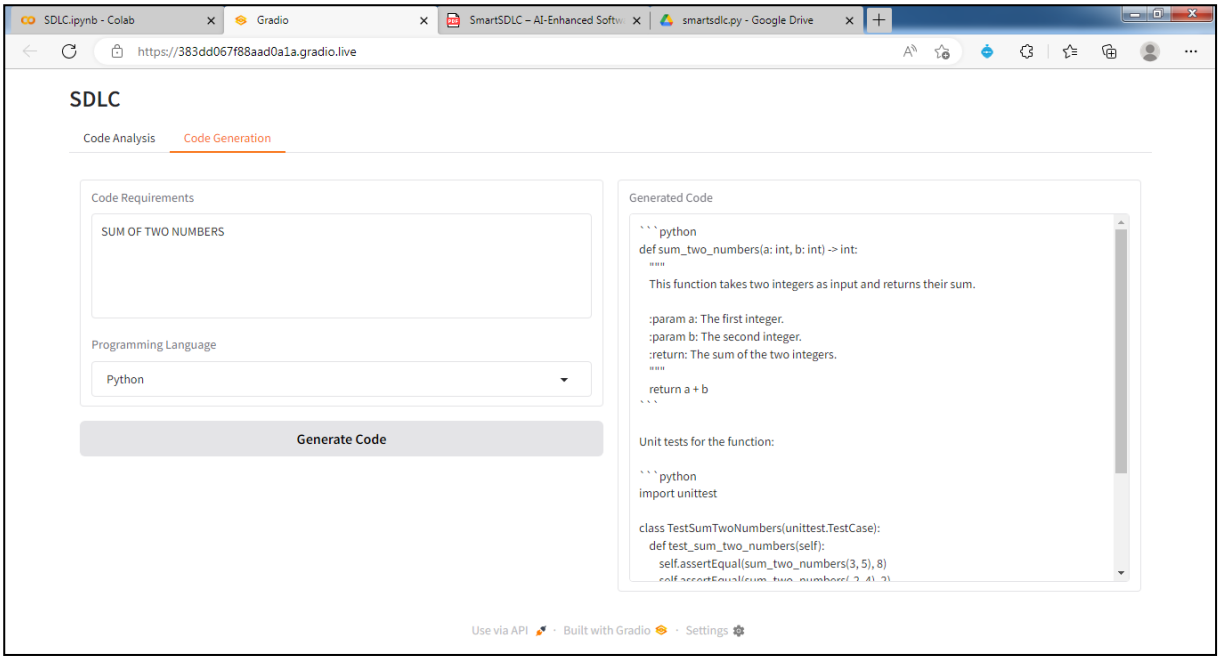
| Testing Aspect | Enterprise (DOCX) | Student (PDF) |
|---|---|---|
| AI Test Generation | ✅ Unit & integration tests | ✅ Unit tests |
| Backend Testing | ✅ FastAPI routes via pytest | ❌ Not applicable |
| Frontend Testing | ✅ Streamlit UI (manual/auto) | ✅ Gradio UI (manual only) |
| Automation | ✅ Full pipeline testing | ❌ Minimal |
| Target Users | Teams, QA engineers | Students, learners |

# 11.Output Screenshots




SDLC.ipynb - Colab
Gradio
SmartSDLC – AI-Enhanced Softw
smartsdlc.py - Google Drive
https://383dd067f88aad0a1a.gradio.live
SDLC
Code Analysis
Code Generation
Upload PDF
Drop File Here
- or -
Click to Upload
Or write requirements here
PYTHON
Analyze
Requirements Analysis
_REQUIREMENTS:
1. Python 3.8 or higher
2. pip (version 20.0.0 or higher)
3. virtualenv (version 20.0.0 or higher)
4. pyenv (version 1.2.3 or higher)
5. node.js (version 12.22.7 or higher)
6. npm (version 6.14.4 or higher)
DEPENDENCIES:
1. Flask
2. SQLAlchemy
3. Flask-SQLAlchemy
4. WTF
5. Flask-Migrate
6. PyJWT
7. Marshmallow
LANGUAGE_REQUIREMENTS:
1. The application should be developed in Python.
2. The codebase should adhere to PEP 8 style guide.
Use via API
Built with Gradio
Settings




# 12.Known Issues

### 1. Enterprise-Grade SmartSDLC (DOCX Version)

- ⚠️ **Model Dependency**

    - Relies on IBM Watsonx Granite models (granite-20b-code-instruct, granite-13b-chat-v1).
    - Performance and accuracy depend on API availability and network stability.
- ⚠️ **Scalability Constraints**
    - Currently designed for **local deployment** (FastAPI + Streamlit).
    - Cloud deployment or CI/CD integration is not yet implemented.
- ⚠️ **Authentication Simplification**
    - Basic login with hashed passwords.
    - No multi-factor authentication or enterprise-grade security (e.g., OAuth2, SSO).
- ⚠️ **Limited Language Coverage**
    - Code generation works best with **Python and JavaScript**.
    - Other languages may produce inconsistent results.
- ⚠️ **PDF Parsing Accuracy**
    - Requirement extraction depends on **PyMuPDF**.
    - Complexly formatted PDFs may lead to misclassification or missing text.
- ⚠️ **Feedback Loop Still Basic**
    - Feedback stored locally in feedback_data.json.
    - No advanced analytics or automated retraining pipeline yet.

---

**2. Student-Friendly SmartSDLC (PDF Version)**

- ⚠️ **No Authentication**
    - Gradio app in Colab is **publicly accessible via a link**.
    - No login system or data security implemented.
- ⚠️ **Temporary Deployment**
    - Runs only in **Google Colab sessions**.
    - Session resets when Colab disconnects, losing state unless manually saved.
- ⚠️ **Lightweight Model Limitations**
    - Uses granite-3.2-2b-instruct (smaller Hugging Face model).
    - Accuracy and code quality are lower compared to Watsonx enterprise models.

- ⚠️ **Manual GitHub Upload**
  - Students must **download and upload files manually** to GitHub.
  - No automated GitHub integration.
- ⚠️ **Performance Constraints**
  - Depends on Colab's free **T4 GPU** availability.
  - May be slow or unavailable during peak usage.
- ⚠️ **UI Simplicity**
  - Gradio interface is minimal, lacking advanced navigation and visualization.

---

**3. Common Issues Across Both Versions**

- AI-generated code may need **manual refinement** before production use.
- Test case generation may not cover **all edge cases**.
- Bug fixer may correct syntax but **miss deeper logical errors**.
- Documentation generator may produce **generic summaries** instead of detailed technical docs.

# 13.Future Enhancements

**1. Enterprise-Grade SmartSDLC (DOCX Version)**

- 🚀 **Cloud Deployment & CI/CD**
  - Deploy backend (FastAPI) and frontend (Streamlit) to **cloud platforms** (AWS, Azure, IBM Cloud, GCP).
  - Integrate **CI/CD pipelines** for continuous testing and deployment.
- 🔐 **Advanced Authentication & Security**
  - Add **OAuth2, JWT refresh tokens, and role-based access control (RBAC)**.
  - Support **SSO (Single Sign-On)** for enterprise teams.
- 🌍 **Expanded Language Support**
  - Extend AI code generation beyond Python/JavaScript to **Java, C#, Go, Kotlin, etc.**
  - Use specialized models for each programming language.
- 🧪 **Enhanced Test Automation**
  - Auto-generate **end-to-end tests** and **API tests**.
  - Integration with **pytest frameworks** and coverage analysis tools.

- 📊 **Feedback Analytics & Learning Loop**
  - Implement **real-time dashboards** for feedback insights.
  - Enable **AI model fine-tuning** using collected feedback.
- 📂 **Version Control & Team Collaboration**
  - Deeper **GitHub/GitLab integration**: pull requests, branch management, issue tracking.
  - Multi-user project workspaces with role permissions.
- 🤖 **Smart Chatbot Evolution**
  - Upgrade to a **context-aware AI assistant** that remembers past interactions across sessions.
  - Integrate **voice-based interaction** for accessibility.

---

**2. Student-Friendly SmartSDLC (PDF Version)**

- 🌐 **Persistent Deployment**
  - Host Gradio app on **Hugging Face Spaces** or **Streamlit Cloud** for longer availability.
  - Eliminate reliance on temporary Colab sessions.
- 🔑 **Basic Authentication**
  - Add lightweight authentication (Gradio login or Firebase Auth).
  - Secure public demo links with access controls.
- ⚡ **Model Upgrade**
  - Move from lightweight granite-3.2-2b-instruct → larger **Watsonx Granite models**.
  - Improve accuracy and reliability of outputs.
- 📤 **Automated GitHub Sync**
  - Direct integration to push code/tests/docs from Colab to GitHub.
  - Enable **auto-deploy pipelines** for students.
- 🎨 **UI Improvements**
  - Add tabbed navigation, better layouts, and result visualization in Gradio.
  - Provide **downloadable reports** of generated outputs.
- 🎓 **Learning Mode**

- Interactive tutorials within the app explaining each SDLC phase.
- "Explain my code" button → AI explains code step-by-step for learners.

---

**3. Common Future Enhancements (Both Versions)**

- **AI Model Fine-Tuning** → Customize Watsonx/Granite models with domain-specific data.
- **Natural Language Query Expansion** → Support multilingual requirements (English, Hindi, Tamil, etc.).
- **Integration with Agile Tools** → Connect with Jira, Trello, or Slack for real-time project tracking.
- **CI/CD Testing Integration** → Auto-generate and run tests during deployment pipelines.
- **Cloud-Native Scalability** → Use Kubernetes/Docker for containerized deployment.