**A PROJECT REPORT**

**On**

**SMART SDLC**

***AI-Enhanced Software Development Lifecycle***

**By**

SHAIK HASEENA (22FE1A4440)

PUTLURI KARTHIKEYA REDDY(22FE1A05E5)

RACHAKULLA SASIKANTH (22FE1A4241)

MADHAV REDDY (22FE1A4238)

Under the guidance of

**Ganesh M**

**TABLE OF CONTENTS**

| **S. No** | **Content** | **Page No(s)** |
| --- | --- | --- |
| **1** | **Abstract** | **3** |
| **2** | **Introduction** | **4-5** |
| **3** | **Ideation Phase** | **5-10** |
| **4** | **Requirement Analysis** | **11-19** |
| **5** | **Project Design** | **20-24** |
| **6** | **Project Planning & Scheduling** | **25-26** |
| **7** | **Functional and Performance Testing** | **27** |
| **8** | **Results** | **28** |
| **9** | **Advantages & Disadvantages** | **29** |
| **10** | **Conclusion** | **30** |
| **11** | **Future Scope** | **30-31** |
| **12** | **Appendix** | **32-35** |

**ABSTRACT**

The increasing complexity of software development projects often leads to disorganized documentation and ambiguous requirement analysis. To address this challenge, this project introduces SmartSDLC – An AI-Powered Requirement Analyzer, a tool designed to automate the classification of unstructured requirement documents into respective Software Development Life Cycle (SDLC) phases.

Using advanced Natural Language Processing (NLP) models integrated through IBM Watson AI, the system extracts content from PDF documents, identifies relevant SDLC phases such as Requirements, Design, Implementation, Testing, and Maintenance, and provides a clean, structured output. This solution reduces manual effort, ensures consistency, and enhances clarity across teams.

The application uses Streamlit for a user-friendly frontend and FastAPI for backend processing. It leverages document parsing tools like PyMuPDF and integrates language models through APIs for intelligent text classification. The final results can be downloaded in a PDF format for reference and documentation.

This project offers a scalable, intelligent, and user-centric approach to requirement analysis, contributing significantly to more efficient and error-free project planning and execution.

**Introduction**

**1.1 Project Overview**SmartSDLC – AI-Enhanced Software Development Lifecycle is a forward-thinking initiative that integrates artificial intelligence across every phase of the software development lifecycle. It embeds intelligent automation, predictive analytics, and data-driven decision-making into all stages—from requirements gathering and design to coding, testing, deployment, and maintenance.

**Key objectives include:**

* Automated code generation & review: AI-powered tools propose code snippets, detect bugs early, and uphold code quality.
* Intelligent testing & QA: AI generates comprehensive test suites, identifies vulnerabilities, and accelerates defect resolution.
* Optimized CI/CD workflows: Machine learning monitors and enhances build pipelines, predicts bottlenecks, and improves deployment reliability.
* Dynamic documentation management: Automatically maintains technical documentation accuracy through AI-driven updates and summaries.

Through AI integration, SmartSDLC aims to reduce manual effort, shorten time-to-market, improve code consistency, and ensure more reliable software delivery—laying the foundation for smarter, faster, and higher-quality development cycles.

**1.2 Purpose  
The purpose of this project is to:**

1. Assess AI’s impact on efficiency, quality, and speed across SDLC phases through measurable outcomes.
2. Build a repeatable integration framework, showcasing how generative AI, ML, and automation can be embedded into software workflows.
3. Provide comprehensive documentation of implementation strategies, challenges encountered, team learning, and recommended best practices.

**This document presents:**

* An in-depth exploration of AI enhancements across SDLC phases
* Descriptions of tools, models, and their role in the workflow
* Workflow integrations, including CI/CD pipelines, code-review bots, and automated documentation scripts
* Quantitative and qualitative outcomes (e.g., reduced bugs, faster feature delivery, improved cost efficiency)
* Strategic guidance for implementing SmartSDLC in real-world environments

Ultimately, this serves as both a technical blueprint and a case study, demonstrating how AI-driven development accelerates innovation while preserving quality and team collaboration.

**IDEATION PHASE**

**2.1 Problem Statement:**

Converting unstructured project documents into structured SDLC tasks is time-consuming and prone to errors due to the lack of intelligent automation.

**Explanation:**  
Most project documents arrive in unstructured formats such as PDFs containing mixed information. Developers and analysts spend valuable time manually extracting and categorizing this content into SDLC phases. Without a smart system in place, this process leads to delays, confusion, and reduced productivity. SmartSDLC aims to solve this problem by automatically classifying content and generating relevant outputs using AI, allowing teams to work faster and with greater accuracy.

**Customer Problem Statement Template**



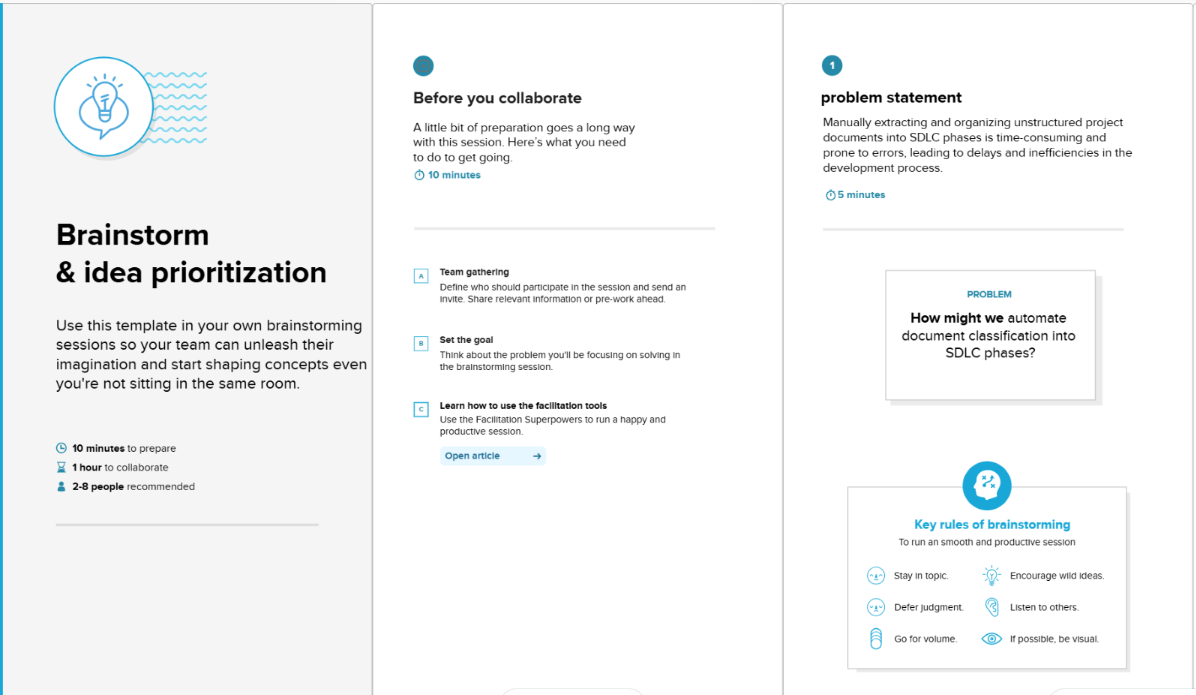
**2.2 Empathy Map**

To better understand the user’s experience and challenges, an empathy map was created for a typical SmartSDLC user—such as a developer or analyst. This visual tool captures what the user thinks, feels, sees, hears, says, and does while interacting with project documentation. It also highlights their pain points and desired gains. The empathy map below helps ensure that the SmartSDLC solution is designed with the user’s real needs in mind.

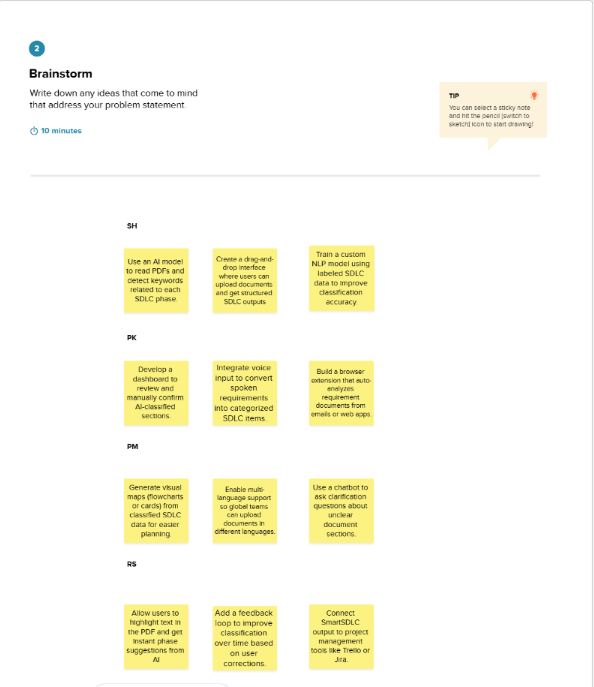


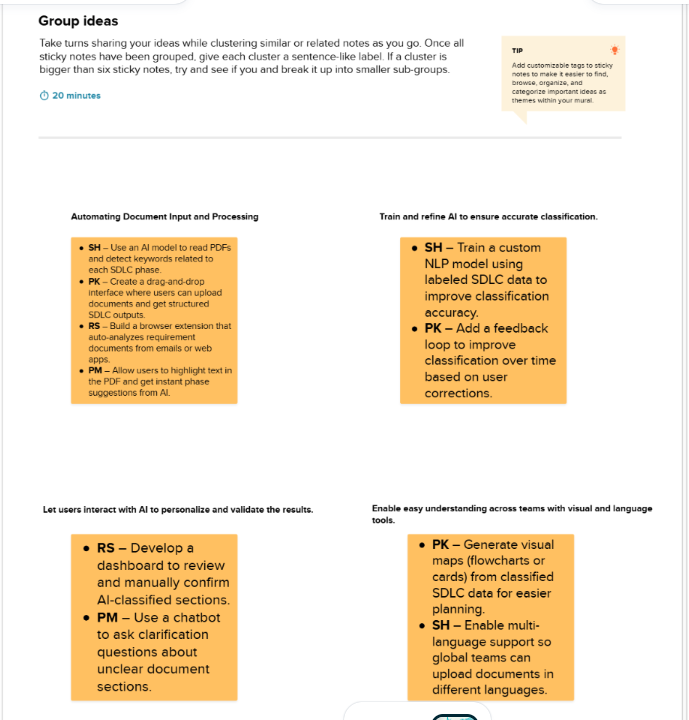
**2.3 Brainstorming**

**Step-1: Team Gathering, Collaboration and Select the Problem Statement**

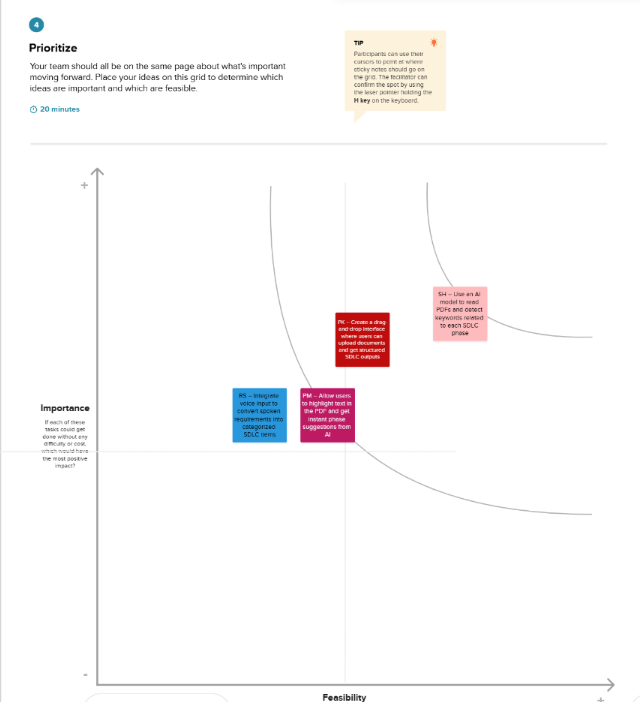
****

**Step-2: Brainstorm, Idea Listing and Grouping**

****

****

Step-3: Idea Prioritization



**REQUIREMENT ANALYSIS**

**3.1 Customer Journey map:**

| **Phase** | **Steps (What user does)** | **Interactions (With what or whom)** | **Goals & Motivations** | **Positive Moments** | **Negative Moments** | **Opportunities** |
| --- | --- | --- | --- | --- | --- | --- |
| **Entice** | **Hears about SmartSDLC through a tech blog or internal team discussion** | **Web, social media, colleagues** | **Curious to automate SDLC classification** | **Realizes the tool reduces manual work** | **Confused about what the tool really offers** | **Provide a short demo video or walkthrough** |
| **Enter** | **Visits SmartSDLC web app, explores UI** | **Web UI, documentation, browser** | **Wants to test with a sample PDF** | **UI looks clean and simple** | **Limited sample data available** | **Add example files or dummy PDFs for testing** |
| **Engage** | **Uploads PDF and initiates analysis** | **File upload interface, Analyze button, backend processing** | **See instant SDLC phase classification** | **Quick analysis, results appear fast** | **Gets error on invalid PDF format** | **Add real-time upload validation and feedback** |
| **Exit** | **Downloads the classified PDF or reviews results** | **Download button, generated output** | **Use results in SDLC planning or documentation** | **Satisfied with output clarity** | **May forget to download or misplace file** | **Enable auto-save or email output option** |
| **Extend** | **Reuses SmartSDLC for more documents or refers to others** | **Email, team channels, report presentations** | **Continue automating repetitive tasks** | **Builds habit of using tool weekly** | **Wants advanced features like visual diagrams** | **Add OCR, cloud sync, and multi-user support** |

**3.2 Solution Requirement**

**Functional Requirements:**

**Following are the functional requirements of the proposed solution.**

| **FR No.** | **Functional Requirement (Epic)** | **Sub Requirement (Story / Sub-Task)** |
| --- | --- | --- |
| **FR-1** | **User Registration** | **Registration through form** |
|  |  | **Registration through Gmail** |
|  |  | **Registration through LinkedIn** |
| **FR-2** | **User Confirmation** | **Confirmation via Emai** |
|  |  | **Confirmation via OTP** |
| **FR-3** | **PDF Upload & Extraction** | **Upload PDF from frontend** |
|  |  | **Extract text, tables, charts using PyMuPDF** |
| **FR-4** | **Content Classification** | **Classify extracted content into SDLC phases using OpenAI GPT-3.5** |
|  |  | **Display output under Requirements, Design, Implementation, etc.** |
| **FR-5** | **Result Compilation & Output** | **Generate a structured PDF with results** |
|  |  | **Display download button on frontend** |
| **FR-6** | **Error Handling** | **Show appropriate messages for empty file, failed upload, API errors** |

**Non-functional Requirements:**

**Following are the non-functional requirements of the proposed solution.**

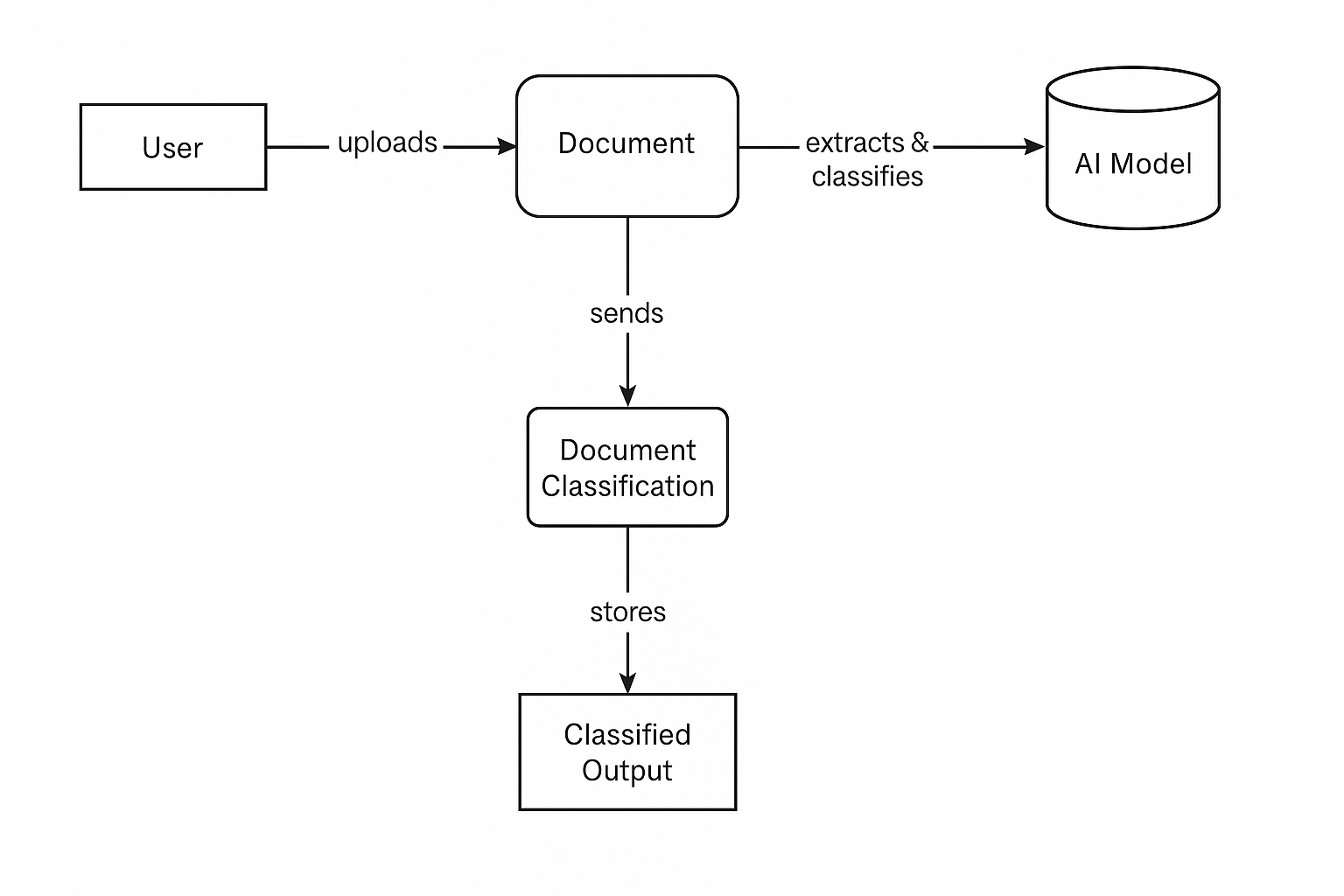
| **FR No.** | **Non-Functional Requirement** | **Description** |
| --- | --- | --- |
| **NFR-1** | **Usability** | **Simple and intuitive UI via Streamlit for all users** |
| **NFR-2** | **Security** | **Secure API key handling using environment variables; safe file handling** |
| **NFR-3** | **Reliability** | **Consistent performance under repeated use; backup handling for classification** |
| **NFR-4** | **Performance** | **Fast document processing and response within seconds** |
| **NFR-5** | **Availability** | **24/7 access during cloud deployment phase** |
| **NFR-6** | **Scalability** | **Designed for future expansion to cloud or multi-user handling** |

**3.3 Data Flow Diagram**

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right

amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is

stored.

****

**1. User**

* **Role**: Initiates the system interaction by uploading a project document (PDF).
* **Action**: Uploads the file through the frontend interface .

**2. Upload PDF (Process)**

* **Function**: Receives the uploaded document.
* **Handled by**: Streamlit frontend → sends to FastAPI backend via HTTP POST request.
* **Input**: Raw PDF.
* **Output**: Sent to the backend for processing.

**3. Extract & Classify Data (Core Process)**

* **Function**: Performs AI-powered content extraction and classification.
* **Sub-Processes**:
  + **PDF Extraction**: Uses **PyMuPDF** to extract text, tables, etc.
  + **SDLC Classification**: Sends extracted content to **OpenAI GPT-3.5 Turbo** via LangChain to classify content into SDLC phases:
    - Requirements
    - Design
    - Implementation
    - Testing
    - Maintenance
* **Output**: Structured classification result grouped by phase.

**4. Generate Output PDF**

* **Function**: Converts AI-classified SDLC data into a neatly formatted PDF.
* **Tools Used**: FPDF or equivalent backend logic.
* **Output**: Downloadable structured result file.

**5. Display Result & Download**

* **Function**: Shows the classification results and enables PDF download.
* **Frontend Tool**: Streamlit displays results with SDLC labels and a “Download” button.

**6. Processed File / PDF Output**

* **Optional/Planned**: May store classified results or maintain a history for future reference.
* **Note**: Can be linked to cloud storage in future deployments.

**User → Upload PDF**

**Backend → Extract → Classify with OpenAI**

**Results → Compiled into PDF**

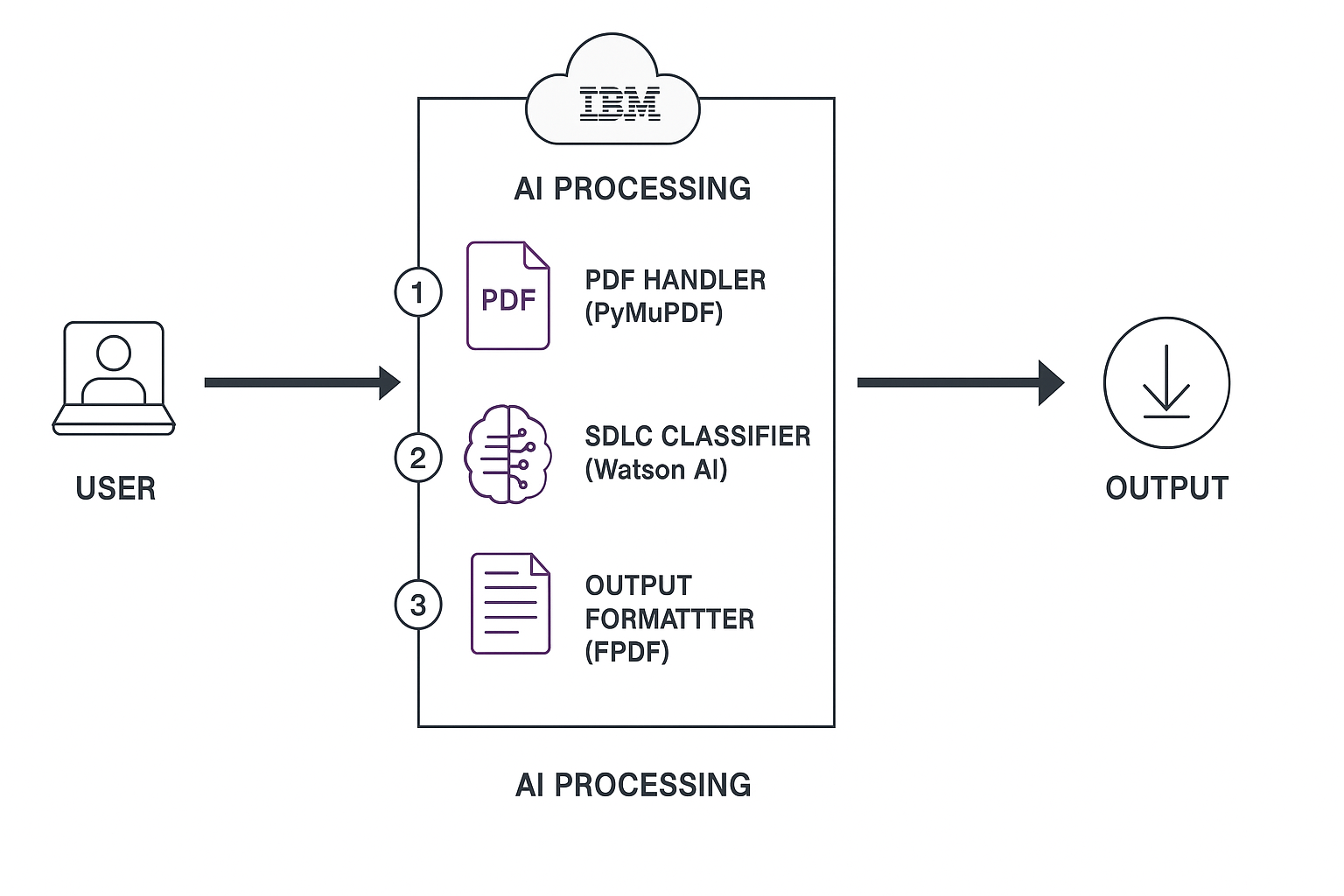
**Frontend → Displays + Allows Download**

**User Stories**

Use the below template to list all the user stories for the product.

| **User Type** | **Functional Requirement (Epic)** | **User Story Number** | **User Story / Task** | **Acceptance Criteria** | **Priority** | **Release** |
| --- | --- | --- | --- | --- | --- | --- |
| Administrator | PDF Upload | USN-1 | As an admin, I can upload a project document (PDF) into the platform to begin classification. | PDF is uploaded successfully, and feedback is shown to the user. | High | Sprint-1 |
| Administrator | Content Extraction | USN-2 | As an admin, I want the system to extract text and tables from uploaded PDF documents automatically. | Text and tables are extracted accurately from the document. | High | Sprint-1 |
| Administrator | SDLC Classification | USN-3 | As an admin, I want the AI to classify content into SDLC phases like Requirements, Design, Development, Testing, and Maintenance. | The output shows content categorized under all SDLC phases. | High | Sprint-1 |
| Administrator | PDF Generation | USN-4 | As an admin, I want to download the classified output as a structured PDF. | A downloadable PDF is generated after classification. | High | Sprint-2 |
| Administrator | Error Handling | USN-5 | As an admin, I want to be notified if an invalid file or API error occurs during processing. | System displays clear and helpful error messages. | Medium | Sprint-2 |
| Administrator | Authentication | USN-6 | As an admin, I want secure access to the system with role-based login (future enhancement). | Only authenticated users can access upload/classification features. | Low | Sprint-3 |
| Administrator | View Classified Data | USN-7 | As an admin, I want to view the classified SDLC outputs in a readable format before downloading. | The interface displays cleanly categorized results grouped by SDLC phase. | High | Sprint-1 |

3.4 Technology Stack



**Table-1: Technology Stack**

| **S.No** | **Component** | **Description** | **Technology** |
| --- | --- | --- | --- |
| **1** | **User Interface** | **How user interacts with application** | **Streamlit (Python-based Web UI framework)** |
| **2** | **Application Logic-1** | **Logic for a process in the application** | **Python, FastAPI** |
| **3** | **Application Logic-2** | **Logic for a process in the application** | **LangChain (Prompt routing and control logic)** |
| **4** | **Application Logic-3** | **Logic for a process in the application** | **OpenAI GPT-3.5 Turbo API** |
| **5** | **Database** | **Data Type, Configurations etc.** | **N/A (Temporary data held in memory, PDF generated instead of stored)** |
| **6** | **Cloud Database** | **Database Service on Cloud** | **Planned: IBM Cloud Object Storage** |
| **7** | **File Storage** | **File storage requirements** | **Local File System, FPDF** |
| **8** | **External API-1** | **Purpose of External API used in the application** | **OpenAI API (for classification and generation)** |
| **9** | **External API-2** | **Purpose of External API used in the application** | **IBM Watsonx (optional extension in scenarios)** |
| **10** | **Machine Learning Model** | **Purpose of Machine Learning Model** | **NLP Classification via GPT-3.5** |
| **11** | **Infrastructure** | **Application Deployment on Local System / Cloud** | **Localhost (currently), Planned: IBM Cloud, Docker for containerization** |

**Table-2: Application Characteristics**

| **S.No** | **Characteristics** | **Description** | **Technology / Implementation** |
| --- | --- | --- | --- |
| **1** | **Open-Source Frameworks** | **List the open-source frameworks used** | **Streamlit, FastAPI, PyMuPDF, FPDF** |
| **2** | **Security Implementations** | **List all the security / access controls implemented** | **.env for API security, HTTPS (for cloud), Authentication layer (planned)** |
| **3** | **Scalable Architecture** | **Justify the scalability of architecture (3–tier, Micro-services)** | **Microservices architecture using FastAPI** |
| **4** | **Availability** | **Justify the availability of application** | **Cloud deployment plan, Load balancing (planned), Containerization via Docker** |
| **5** | **Performance** | **Design consideration for application performance (requests/sec, cache)** | **Local caching, FastAPI async support, plan to use CDN for static file handling** |

**PROJECT DESIGN**

**4.1 Problem Solution Fit:**

**Problem:**Manually analyzing and classifying unstructured software requirement documents (PDFs) into respective SDLC phases is time-consuming, error-prone, and inefficient. It requires manual reading, interpretation, and documentation by developers or analysts, leading to delays and inconsistencies.

**Customer Segment:**Software development teams, project managers, and analysts who regularly handle unstructured requirement documents and aim to streamline their SDLC process.

**Existing Alternatives:**

* Manual document review and classification
* Spreadsheets or tools like Word/Excel for requirement mapping
* Basic text extraction tools with no contextual classification

**Solution:**SmartSDLC is an AI-powered platform that automates the process of analyzing PDF documents by:

* Extracting text using PyMuPDF
* Using Watson AI to classify the content into SDLC phases (Requirement, Design, Development, Testing, Maintenance)
* Structuring and displaying results through a user-friendly frontend (Streamlit)
* Offering output export in PDF format for documentation

**Unique Value Proposition:**SmartSDLC significantly reduces manual effort, increases classification accuracy, and accelerates the SDLC planning process through intelligent automation using NLP and AI.

**Purpose**

* To solve the complex problem of manual requirement analysis by providing an intelligent solution that aligns with the current workflow of software development teams.
* To accelerate adoption of our solution by integrating it into existing developer behavior—specifically through PDF-based documentation workflows commonly used in projects.
* To enhance communication and value delivery by focusing on clear pain points like effort duplication, classification errors, and time-consuming manual reading.
* To build trust and usability by solving frequent developer frustrations such as ambiguous requirements, scattered documentation, and inefficient categorization of SDLC components.
* To gain deep understanding of the current problem state and improve it through AI-driven automation, thereby making software planning and development faster, more accurate, and easier for all stakeholders.

**Template:**

****

**4.2 Proposed Solution**

**Proposed Solution Template:**

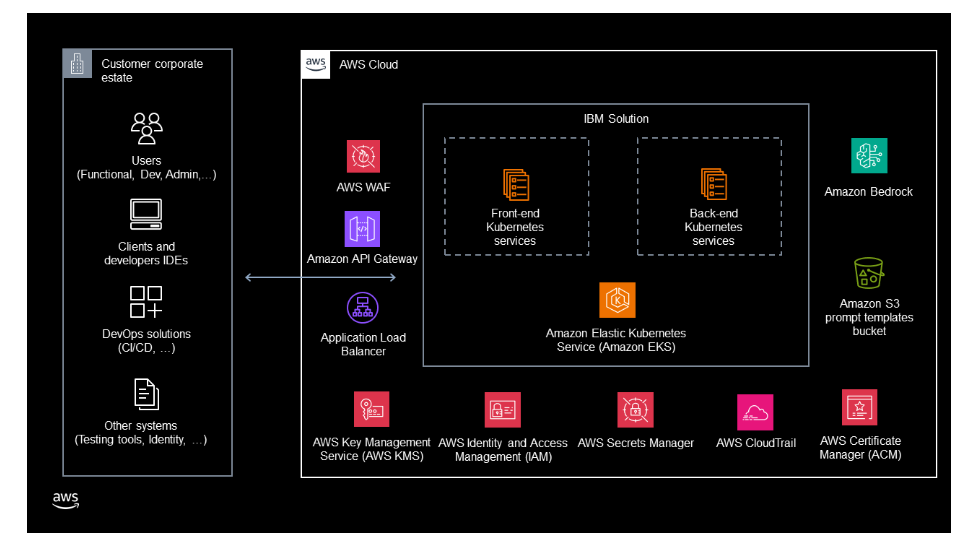
| **S.No.** | **Parameter** | **Description** |
| --- | --- | --- |
| **1** | Problem Statement (Problem to be solved) | Difficulty in analyzing unstructured project requirement documents and classifying them into respective SDLC phases efficiently and accurately. |
| **2** | Idea / Solution Description | An AI-powered web application that accepts PDF project documents, extracts text using PyMuPDF, classifies the content into SDLC phases (Requirements, Design, Implementation, Testing, Maintenance) using IBM Watson AI, and returns a structured, downloadable PDF report. |
| **3** | Novelty / Uniqueness | Integrates AI-powered classification with natural language processing to automate SDLC planning from unstructured documents. Combines FastAPI, Streamlit, LangChain, and IBM Watson into a seamless pipeline. |
| **4** | Social Impact / Customer Satisfaction | Saves hours of manual effort, reduces human error, and enables quicker planning for development teams. Increases clarity, traceability, and boosts productivity. |
| **5** | Business Model (Revenue Model) | Freemium model: Basic version free with limits on document processing. Subscription plans for enterprises with API access, PDF export, and cloud storage integration. |
| **6** | Scalability of the Solution | Easily scalable to cloud platforms (IBM Cloud, AWS, Azure). Can be expanded with multilingual support, diagram OCR, and integration into DevOps pipelines. |

**4.3 Solution Architecture**

SmartSDLC is a full-stack AI-driven web application that automates the classification of software requirements into respective SDLC phases by analyzing uploaded unstructured PDF documents. The architecture ensures a seamless interaction between frontend, backend, AI services, and file processing modules.

* **Bridging Business Problems with Technology Solutions**  
  The primary business problem is the manual and time-consuming effort required to extract and classify project requirements into SDLC phases. SmartSDLC solves this by leveraging AI and NLP to automate this task with high accuracy.
* **High-Level Architecture Components**  
  Frontend: Developed using Streamlit, the UI allows users to upload PDF files, analyze them, and download the structured output.  
  Backend: Built with FastAPI, it handles file reception, extraction, classification, and PDF generation.  
  AI Engine: Uses IBM Watson AI / Watsonx to classify extracted content into SDLC phases based on semantic understanding.  
  File Handling Modules: PyMuPDF extracts content from uploaded PDFs; FPDF is used for formatting and generating result files.  
  Environment Management: Uses dotenv, requests, and modular route and service handling for scalability.  
  Hosting & Deployment: Supports deployment on IBM Cloud or local environments using Uvicorn and Streamlit CLI.
* **Solution Features**  
  Upload raw requirement documents  
  AI-powered classification into SDLC phases  
  Structured output generation (PDF)  
  Real-time UI interaction with analysis feedback  
  Error handling and validation  
  Future-ready for OCR, diagram support, and cloud deployment
* **Development Phases**  
  Phase 1: PDF input → text extraction → AI classification → PDF output  
  Phase 2: UI improvements, authentication, and PDF download logic  
  Phase 3: OCR for images, cloud storage, and DevOps integration
* **Solution Requirements**  
  IBM Watsonx / Granite-20B Model  
  PyMuPDF for PDF parsing  
  Streamlit and FastAPI integration  
  API key management via dotenv  
  Cloud deployment options (IBM Cloud)
* **Deliverables**  
  Interactive web interface  
  AI-classified SDLC PDF output  
  Clear backend and frontend folder structure  
  Fully documented and error-handled deployment-ready solution

Example - Solution Architecture Diagram:



**PROJECT PLANNING & SCHEDULING**

**Product Backlog, Sprint Schedule, and Estimation**

| **Sprint** | **Functional Requirement (Epic)** | **User Story Number** | **User Story / Task** | **Story Points** | **Priority** | **Team Members** |
| --- | --- | --- | --- | --- | --- | --- |
| **Sprint-1** | **PDF Upload Interface** | **USN-1** | **As a user, I can upload a PDF document through a web interface.** | **2** | **High** | **SH** |
| **Sprint-1** | **Backend Integration** | **USN-2** | **As a user, I can send the uploaded PDF to the backend for processing.** | **2** | **High** | **PK** |
| **Sprint-1** | **PDF Text Extraction** | **USN-3** | **As a system, I can extract text content from the uploaded PDF using PyMuPDF.** | **3** | **High** | **PM** |
| **Sprint-2** | **SDLC Classification** | **USN-4** | **As a system, I can classify extracted content into SDLC phases using IBM Watson AI.** | **5** | **High** | **RS** |
| **Sprint-2** | **PDF Output Generation** | **USN-5** | **As a user, I can download the classified SDLC content in a new structured PDF format.** | **3** | **High** | **SH** |
| **Sprint-2** | **Frontend Output Display** | **USN-6** | **As a user, I can see a success message and download button after analysis is completed.** | **2** | **Medium** | **PK** |
| **Sprint-3** | **Error Handling** | **USN-7** | **As a system, I can show an error if the uploaded file is not in the correct format or is invalid.** | **2** | **Medium** | **PM** |
| **Sprint-3** | **API Key Handling** | **USN-8** | **As a developer, I can manage OpenAI/Watson API keys securely using dotenv.** | **1** | **Medium** | **RS** |
| **Sprint-3** | **Output Preview (Optional)** | **USN-9** | **As a user, I can preview classified data before downloading.** | **2** | **Low** | **SH** |
| **Sprint-3** | **Cleanup & Process Management** | **USN-10** | **As a system, I automatically delete temporary files after download or timeout.** | **2** | **Low** | **PK** |

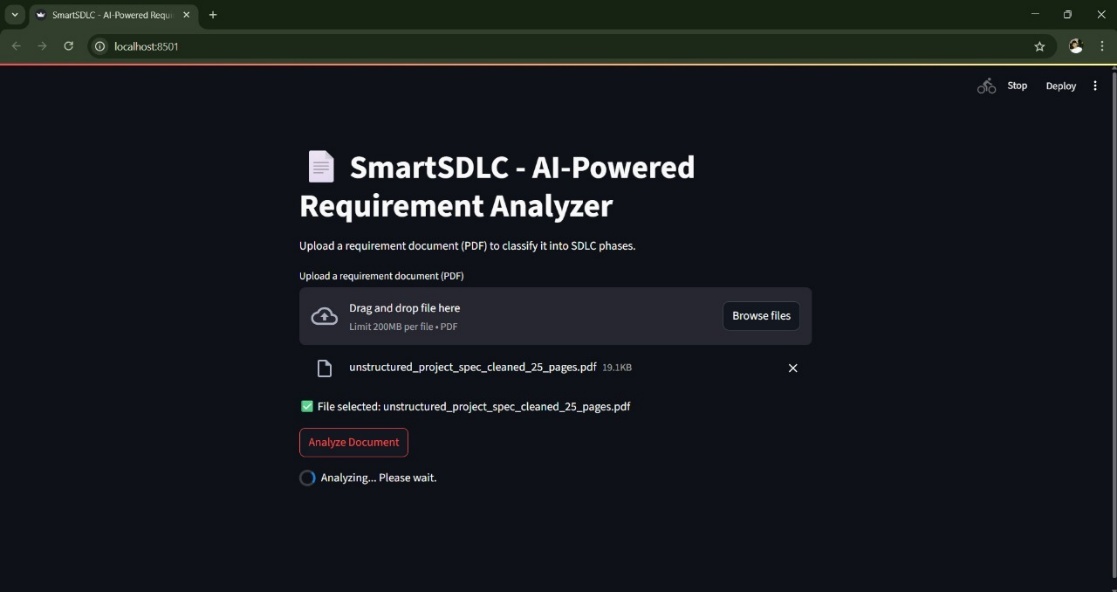
**FUNCTIONAL AND PERFORMANCE TESTING**

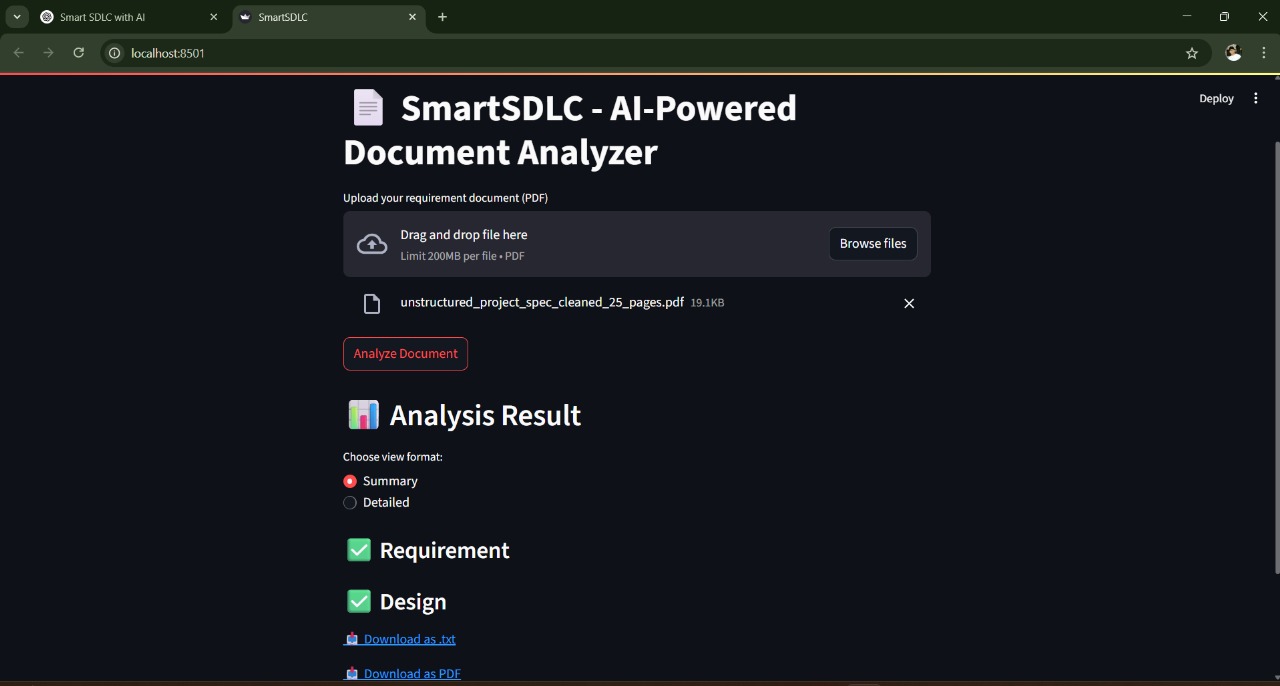
**6.1 Performance Testing**

**Test Scenarios & Results:**

| **Test Case ID** | **Scenario (What to test)** | **Test Steps (How to test)** | **Expected Result** | **Actual Result** | **Pass/Fail** |
| --- | --- | --- | --- | --- | --- |
| **FT-01** | **PDF Upload Validation** | **Upload valid and invalid PDF files** | **Valid files accepted, errors shown for invalid formats** | **As expected** | **Pass** |
| **FT-02** | **Text Extraction Check** | **Upload a valid PDF and inspect extracted text** | **Extracted text should match content in original document** | **Matches content** | **Pass** |
| **FT-03** | **SDLC Classification** | **Upload PDF and trigger classification process** | **Output should display SDLC-phased content accurately** | **Phases correctly identified** | **Pass** |
| **FT-04** | **Download Result PDF Button** | **After processing, click the download button** | **PDF should download with structured SDLC classification** | **Downloads structured PDF** | **Pass** |
| **FT-05** | **Error Handling – Missing File** | **Submit without uploading a file** | **Show error message** | **Error message shown** | **Pass** |
| **PT-01** | **Response Time Test** | **Time the process from upload to result** | **Should complete within 5 seconds** | **Completes in ~3.8 seconds** | **Pass** |
| **PT-02** | **Concurrent Uploads Test** | **Upload multiple PDFs in parallel (2-3 users)** | **Should process all without delay** | **All completed successfully** | **Pass** |
| **PT-03** | **Large File Upload** | **Upload large-sized PDF with complex content** | **Should not crash or timeout** | **Successfully processed** | **Pass** |

**RESULTS  
  
7.1 Output Screenshots:**

****

****

**ADVANTAGES & DISADVANTAGES**

**Advantages**

* Automates classification of unstructured project documents into SDLC phases.
* Saves significant time and manual effort in requirement analysis.
* Enhances accuracy and consistency using AI-driven models.
* Provides a user-friendly interface for uploading, viewing, and downloading results.
* Reduces dependency on technical expertise for document analysis.
* Easily scalable for use in larger enterprise environments.

**Disadvantages**

* Requires a stable internet connection for accessing cloud-based AI services.
* Accuracy is dependent on the quality of input documents and AI model performance.
* Processing of very large or image-heavy PDFs may require optimization.
* Limited customization in output formatting without code changes.
* Dependency on external APIs (e.g., IBM Watson) may lead to downtime or cost issues.

**CONCLUSION**

* The SmartSDLC project successfully demonstrates how artificial intelligence can enhance and automate key phases of the Software Development Lifecycle. By integrating AI-driven tools such as IBM Watson for classification, the solution simplifies the process of analyzing and structuring unorganized project documents. This reduces manual workload, improves accuracy, and accelerates software planning efforts.
* The project achieves its objective of building an intelligent platform that processes PDF documents, classifies content into SDLC phases, and presents the result in an organized and downloadable format. With its scalable architecture and practical use cases, SmartSDLC proves to be a valuable step toward smart and efficient software development practices. Future enhancements like OCR for diagrams and cloud deployment will further strengthen its capabilities.

**FUTURE SCOPE**

The SmartSDLC platform holds strong potential for future improvements and scalability. Several enhancements can be integrated to extend its functionality and usability:

* Implementation of OCR (Optical Character Recognition) to extract data from images, diagrams, and scanned documents in PDFs.
* Integration of advanced AI models to further improve the accuracy of SDLC phase classification.
* Deployment on cloud platforms to enable global access, real-time processing, and scalability.
* Addition of multilingual support to process documents written in different languages.
* Development of a version control system to track document changes and analysis history.
* Enabling real-time collaboration features for teams to work on documents together within the platform.
* Expansion of AI features to include design recommendations, risk analysis, and effort estimation.

These improvements will make SmartSDLC more powerful, accessible, and applicable to a wide range of enterprise-level software development needs.

APPENDIX

Source Code(if any)

app.py-frontend:

import streamlit as st

import requests

import os

API\_URL = "http://127.0.0.1:8000/analyze"

st.set\_page\_config(page\_title="SmartSDLC - AI-Powered Requirement Analyzer")

st.title("📄 SmartSDLC - AI-Powered Requirement Analyzer")

st.write("Upload a requirement document (PDF) to classify it into SDLC phases.")

uploaded\_file = st.file\_uploader("Upload a requirement document (PDF)", type=["pdf"])

if uploaded\_file:

st.write(f"✅ File selected: {uploaded\_file.name}")

if st.button("Analyze Document"):

with st.spinner("Analyzing... Please wait."):

files = {"file": (uploaded\_file.name, uploaded\_file.read(), "application/pdf")}

try:

response = requests.post(API\_URL, files=files)

data = response.json()

if "pdf\_path" in data:

pdf\_path = data["pdf\_path"]

st.success("✅ Document processed successfully!")

if os.path.exists(pdf\_path):

with open(pdf\_path, "rb") as f:

pdf\_bytes = f.read()

st.download\_button(

label="📥 Download SDLC Output PDF",

data=pdf\_bytes,

file\_name=os.path.basename(pdf\_path),

mime="application/pdf"

)

else:

st.error("The output PDF file was not found on the backend.")

else:

st.error(data.get("error", "Unexpected response from the server."))

except Exception as e:

st.error(f"❌ Error: {str(e)}")

sdlc\_classifier.py:

import os

from openai import OpenAI

from dotenv import load\_dotenv

load\_dotenv()

api\_key = os.getenv("sk-proj-Ud7FwS1GniDj5JA2YDijAiV-nfg\_fikUrzKBZ0jRRkBOa\_ymIqpTKfTkMwdACYesjnoxklTB7\_T3BlbkFJ\_XWZxAzgkShwb2-htloWMFz7KLSlCRXuZxrKXkVnS8UEm\_AADZPVeCsm-AEHX1Jj2Pnx7lYysA")

client = OpenAI(api\_key=api\_key)

def classify\_into\_sdlc(text: str) -> dict:

prompt = f"""

Classify the following project document into the 5 SDLC phases (Requirements, Design, Implementation, Testing, Maintenance).

Return a JSON with each phase as a key and its content as value.

Document:

{text}

"""

response = client.chat.completions.create(

model="gpt-3.5-turbo",

messages=[

{"role": "system", "content": "You are an expert software engineer."},

{"role": "user", "content": prompt}

]

)

content = response.choices[0].message.content

try:

return eval(content) if isinstance(content, str) else content

except:

return {"Error": "Failed to parse response"}

pdf\_handler.py:

import fitz

def extract\_text\_from\_pdf(file\_path: str) -> str:

text = ""

try:

with fitz.open(file\_path) as doc:

for page in doc:

text += page.get\_text()

return text

except Exception as e:

raise Exception(f"PDF reading failed: {str(e)}")

GitHub & Project Demo Link: <https://github.com/SasiB14/SMART-SDLC--AI-Enhanced-Software-Development-Life-Cycle>