1. Introduction

•**Project Title**: Smart SDLC – AI Requirement Analyzer
•**Team Members**:
Unnam Sneha Siri
Tadaboyina Chandrika
Syed Naseera Banu
Srungarapati Chetana

2. Project Overview

Purpose:
To automate the Software Development Life Cycle (SDLC)requirement analysis process using AI. Users can upload a requirements document, and the system provides insights like functional requirements, modules, and potential features.

Features:
•Upload and analyze .txt requirement files
•AI-powered extraction of requirements
•Interactive UI built with Streamlit
•FastAPI backend for processing requests

3. Architecture

Frontend:

- Built using **Streamlit** for quick UI prototyping
- Provides a simple drag-and-drop interface for uploading files and viewing AI-generated insight

Backend:
•Developed using **FastAPI**
•Handles file input and communicates with AI model APIs (e.g., OpenRouter or Watsonx)

Database:

- **No database used currently**
- All analysis is done in memory during runtime

4. Setup Instructions

Prerequisites:

•Python 3.10+
•pip (Python package installer)
•(Optional) IBM Cloud API key or OpenRouter API key

Installation:

```
git clone https://github.com/Nassi786/sdlc
cd sdlc
# Install backend dependencies
cd backend
pip install -r requirements.txt

# Install frontend dependencies
cd ../frontend
pip install -r requirements.txt
```

5. Folder Structure

•**frontend/**:
•app.py: Streamlit app UI
•.streamlit/: config files

•**backend/**
    •main.py: FastAPI backend
    •ai/: Custom logic for AI model integration (e.g., watsonx_integration.py)

6. Running the Application

Frontend:

```
cd frontend
streamlit run app.py
```

Backend:

```
cd backend
uvicorn main:app --reload
```

7. API Documentation

POST /analyze

- Description: Accepts requirement text and returns insights
- Request Body:

```
{
  "text": "Your requirement text
here"  9. User Interface
}
```

Response Example:

```
{
  "functional_requirements": [...],
  "non_functional_requirements":
[...]
}
```

8. Authentication

- **Current Status**:

Open to future addition of API key-based or token-based security.

9. User Interface

UI Features:

•Upload section for requirement files

•"Analyze" button to trigger processing

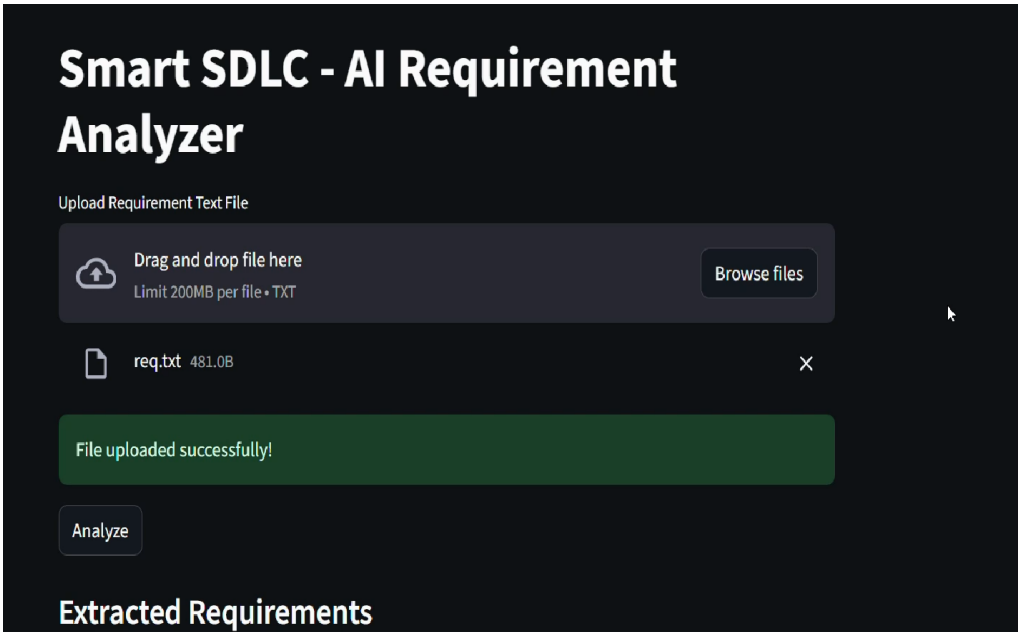
•Insight output with expandable sections

10. Testing

•**Tools**:

- Manual testing through Streamlit interface
- FastAPI endpoints tested using cURL and Postman

11. Screenshots or Demo



12. Known Issues

•Openrouter integration may fail if incorrect API/project ID is used

•Large text files (>1MB) may take longer to analyze or timeout

13. Future Enhancements

- Add user authentication (JWT or OAuth)
- Integrate database (e.g., MongoDB or PostgreSQL) to store analysis results
- Add support for PDF and DOCX formats
- Improve model response with RAG pipeline or embeddings