Student Performance Analyzer & Career Recommendation System

Design & Implementation Report

Prepared By:

Shubhangi Vilas Jeve

Organization:

ProExtern

Date:

February 20, 2025

Table of Contents

1. INTRODUCTION 3

1.1 PURPOSE AND SCOPE 3

2 TECHNOLOGY STACK 4

3 System Architecture 6

4 Object-Oriented Design (OOD) Diagrams 7

4.1 Use Case Diagram 7

4.2 USE CASE DIAGRAM 8

4.3 SEQUENCE DIAGRAM 9

4.4 Activity Diagram 10

5 Features & Functionalities 12

6 Challenges & Next Steps 14

6.1 Challenges Faced: 14

6.2 Next Steps in Development: 14

1. INTRODUCTION

The Student Performance Analyzer & Career Recommendation System is a comprehensive platform designed to empower students by providing deep insights into their academic performance and aligning those insights with personalized career recommendations. At its core, the system leverages advanced document processing capabilities to extract, clean, and structure data from a variety of document formats, forming the foundational input for further analysis and recommendation algorithms.

In today's fast-paced academic environment, students are often required to submit a wide range of documents such as academic transcripts, certificates, handwritten notes, and other supporting materials. Manually extracting and analyzing the information contained in these documents can be both time-consuming and error-prone. To address these challenges, our system is built with a robust document processing module that automates the extraction of textual content from uploaded files.

1.1 PURPOSE AND SCOPE

The Smart Document Processor is a web-based application designed to convert unstructured documents into a structured JSON format. Its core purpose is to allow users to upload various types of documents—such as images, digital PDFs, scanned (handwritten) PDFs, and Word documents—and process them to extract and organize text content. This enables downstream applications (like data indexing, archival, or further natural language processing) to work with a clean and structured representation of the document.

SCOPE:

- Support for multiple file types using specialized processing methods.
- Utilization of Optical Character Recognition (OCR) for image-based documents.
- Extraction of textual content from both digital and scanned PDFs.
- Parsing and formatting Word documents using appropriate libraries and tools.
- A user-friendly frontend interface that facilitates file selection, upload, and displays the processed output.

2 TECHNOLOGY STACK

The project leverages a robust and diverse set of tools, frameworks, and programming languages to ensure efficiency, scalability, and accuracy in processing student documents and generating career recommendations:

- Programming Language:
    - Python: Chosen for its rich ecosystem, ease of use, and extensive libraries for both document processing and backend development.
- Web Framework:
    - FastAPI: A modern, high-performance framework for building RESTful APIs that simplifies endpoint creation, request handling, and asynchronous processing. FastAPI also provides built-in support for CORS, enabling secure cross-origin requests.
- OCR & Image Processing:
    - Tesseract OCR (via pytesseract): Utilized to extract text from image files and scanned PDFs. Tesseract is a well-established OCR engine that works well with diverse fonts and image qualities.
    - Pillow (PIL): Used for basic image manipulations like converting images to grayscale, a key preprocessing step for enhancing OCR accuracy.
    - Potential Enhancements: Future iterations may incorporate advanced libraries (e.g., OpenCV) for noise reduction, thresholding, and image enhancement.
- PDF & Document Handling:
    - PyPDF2: Handles digital PDF documents by extracting embedded text.
    - PyMuPDF (fitz): Processes scanned PDFs by converting pages into images for OCR.
    - python-docx & antiword: Manage both DOCX and older DOC formats, ensuring a wide range of document compatibility.

- Frontend Technologies:
    - HTML & Tailwind CSS: Create a responsive and visually appealing web interface for file uploads and displaying results.
    - JavaScript: Provides interactivity for file selection, form submission, and dynamic display of processed results.
    - Static Files Management: The FastAPI application serves static files (HTML, CSS, and JS) seamlessly.
- Utility Libraries:
    - Regular Expressions (re), shutil, subprocess, time, datetime, os, io: Assist with tasks like text cleaning, file manipulation, logging, and temporary file storage.
- Server & Deployment:
    - Uvicorn: An ASGI server used to run the FastAPI application, ensuring high performance and scalability.

Layer	Technologies
Backend	FastAPI, PyTesseract, PyPDF2, PyMuPDF
Frontend	HTML5, Tailwind CSS, JavaScript
OCR	Tesseract OCR Engine (v5.3)
PDF	Poppler Utilities (v23.08)
Deployment	Uvicorn ASGI Server

3 System Architecture

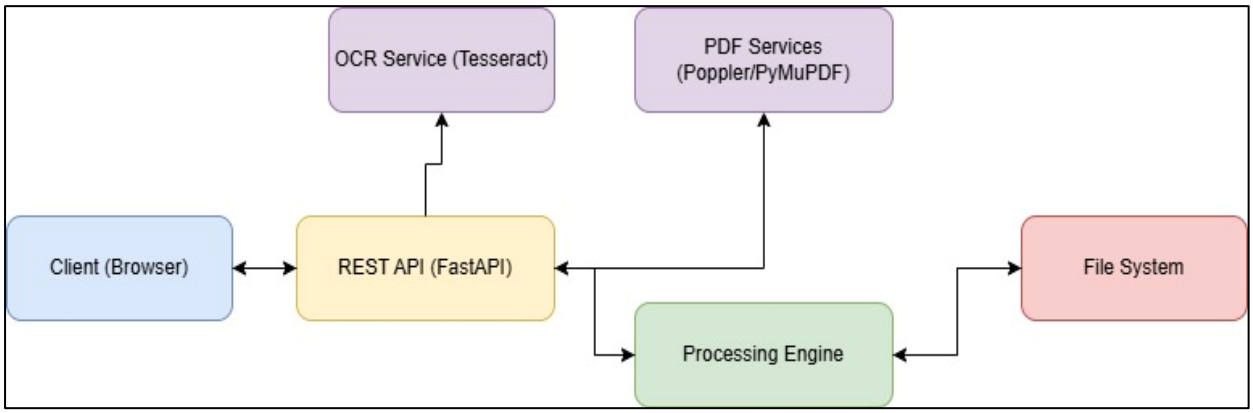


Fig : System Architecture

Data Flow:

1. File upload via HTTP POST
2. Type-based routing to processing modules
3. Parallel text extraction pipelines
4. Multi-stage text normalization:

   Raw Text → Cleaning → Semantic Parsing → JSON Structuring

5. API response with structured data payload

4 Object-Oriented Design (OOD) Diagrams

4.1 Use Case Diagram

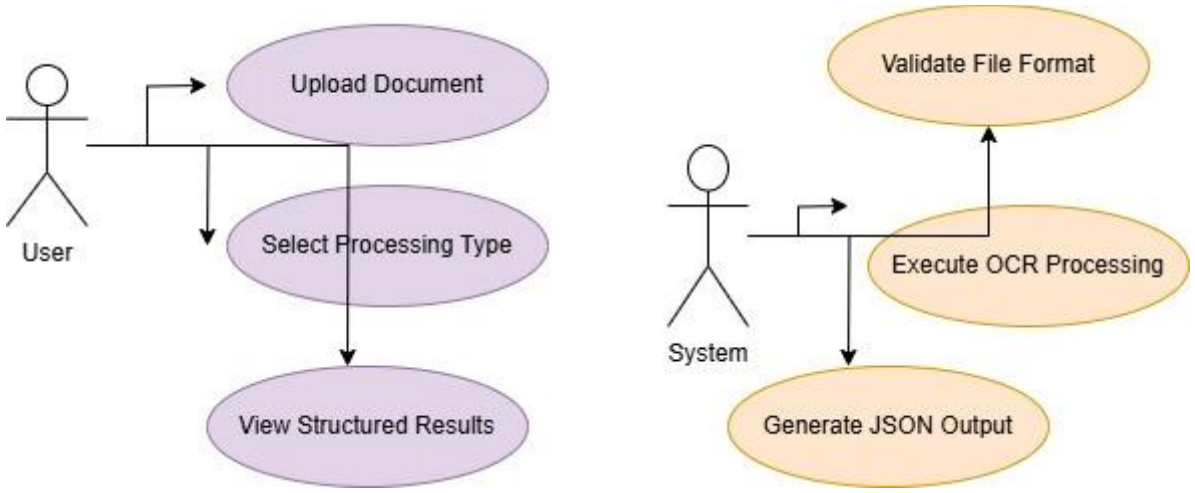


Fig : Use Case Diagram

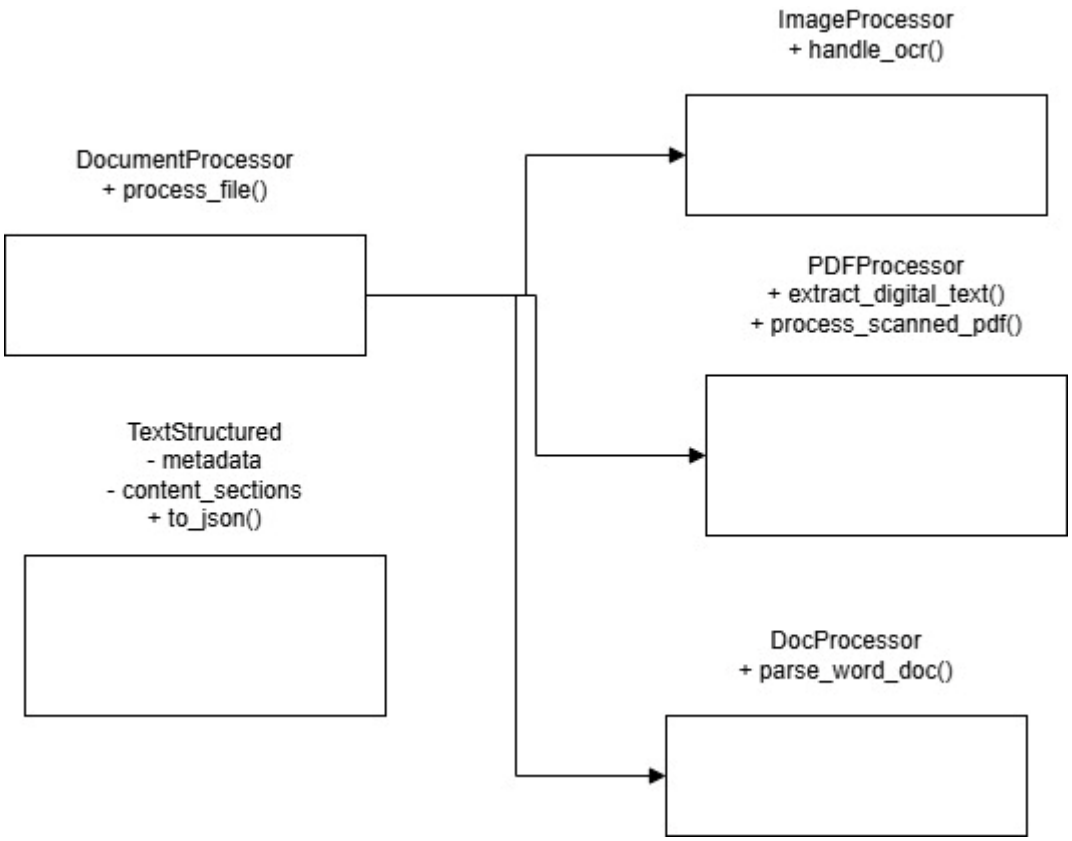
User Actor:

- Upload Document
- Select Processing Type
- View Structured Results

System Actor:

- Validate File Format
- Execute OCR Processing
- Generate JSON Output

4.2 USE CASE DIAGRAM



4.3 SEQUENCE DIAGRAM

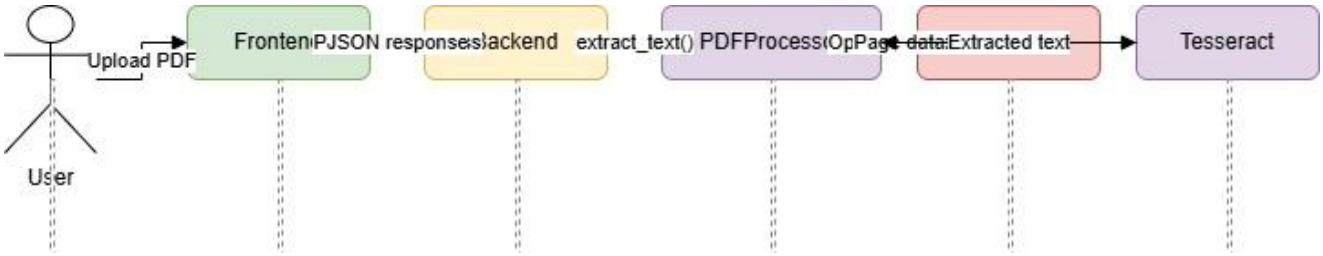


Fig : Sequence Diagram

This diagram illustrates the complete flow:

1. **User → Frontend:** Upload PDF
2. **Frontend → Backend:** POST /api/process
3. **Backend → PDFProcessor:** extract_text()
4. **PDFProcessor → PyMuPDF:** Open document
5. **PyMuPDF → PDFProcessor:** Page data
6. **PDFProcessor → Tesseract:** OCR request
7. **Tesseract → PDFProcessor:** Extracted text
8. **Backend → Frontend:** JSON response

4.4 Activity Diagram

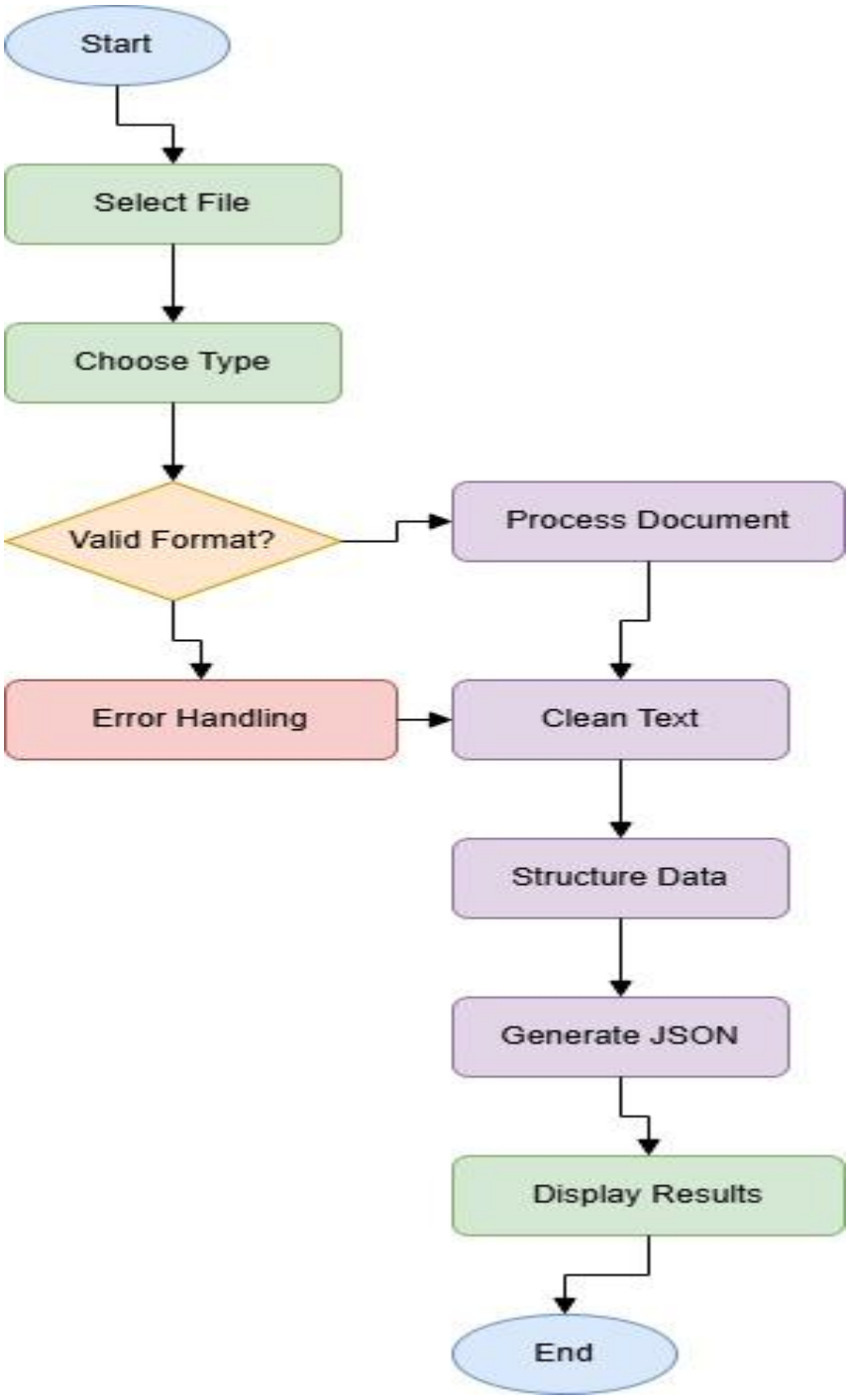


Fig : Activity Diagram

Activity Diagram Flow:

1. **Start**
   ↓
2. **Select File**
   ↓
3. **Choose Type**
   ↓
4. **Decision: Valid Format?**
   ├── Yes → **Process Document**
   └── No → **Error Handling**
5. **Clean Text**
   ↓
6. **Structure Data**
   ↓
7. **Generate JSON**
   ↓
8. **Display Results**
   ↓
9. **End**

5 Features & Functionalities

Features & Functionalities

- **Multi-Format File Upload:**
    - Support for various file types such as JPEG, PNG, PDF, and DOCX.
    - File type and size validations to ensure only supported documents are processed.
    - Secure file handling with dedicated storage for uploaded files.
- **Advanced Text Extraction & OCR:**
    - Integration of Tesseract OCR to extract text from image-based and scanned documents.
    - Use of PyPDF2 and PyMuPDF to handle digital and scanned PDFs respectively.
    - Conversion of non-machine-readable content into structured text.
- **Preprocessing & Data Cleaning:**
    - Preprocessing steps such as converting images to grayscale.
    - Future enhancements may include noise reduction and thresholding techniques to improve OCR accuracy.
    - Cleaning and normalizing extracted text to remove unwanted characters and spaces.
- **Structured Data Output:**
    - Organizing the processed text into a JSON format.
    - Inclusion of metadata (timestamp, version) and segmentation into meaningful sections.
    - Facilitates downstream processes like performance analysis and career recommendations.

- **User-Friendly Interface:**
    - Intuitive web interface for file uploads, processing feedback, and results display.
    - Real-time error handling and user notifications.
    - Responsive design using Tailwind CSS for a consistent experience across devices.
- **Performance Analysis & Career Recommendations (Future Integration):**
    - Modules to analyze student performance based on the structured data extracted.
    - Algorithms to match performance metrics with personalized career paths.
    - Generation of actionable insights and recommendations to guide academic and professional development.

6 Challenges & Next Steps

6.1 Challenges Faced:

- OCR Accuracy:

  Handling low-quality images or complex handwritten content can reduce OCR accuracy. Variations in font style, size, and document quality present a continuous challenge.

- File Format Variability:

  Supporting multiple file formats and ensuring consistent output requires integrating various libraries, each with its own limitations.

- Performance Optimization:

  Processing large documents or a high volume of concurrent uploads may impact performance. Optimizing I/O operations and processing pipelines is essential.

- Error Handling & Security:

  Ensuring robust validation and error handling to prevent processing errors or malicious file uploads is critical.

- Integration with Analysis Modules:

  Future integration with performance analysis and recommendation modules may require additional data processing, machine learning models, and API coordination.

6.2 Next Steps in Development:

- Enhance OCR and Preprocessing:

  Integrate advanced image processing techniques (e.g., noise reduction, thresholding, deskewing) to improve text extraction accuracy.

- Implement Comprehensive Validation:

  Add robust file type and size validations, and implement security measures to prevent malicious uploads.

- Database Integration:
- Implement a database (such as PostgreSQL) to store processed documents, user data, and analysis results for persistent storage.
- User Management & Authentication:

  Develop a secure user authentication and authorization system to enable personalized user sessions and document history.
- Scalability & Performance Optimization:

  Explore asynchronous processing and scaling techniques (such as using task queues like Celery) to handle high loads and improve responsiveness.
- Advanced Analysis and Recommendation Engine:

  Integrate performance analysis modules and develop algorithms to generate personalized career recommendations based on the extracted data.
- UI/UX Enhancements:

  Refine the frontend design for improved usability, incorporating feedback mechanisms, progress indicators, and more intuitive navigation.