# Assignment1:Content Extraction Using Open-Source and Enterprise Solutions

## Madhura Adadande

## Vemana Anilkumar

## Ashwin Badamikar

## 

## 

## **2. Introduction**

In this project, we focus on developing a prototype AI application that extracts, processes, and organizes data from unstructured sources such as PDFs and web pages. The goal is to evaluate the feasibility of using open-source tools versus enterprise solutions for data extraction and transformation.

### **Technologies Involved:**

* **1. PDF Extraction:** 
  + **Open Source: pymupdf, pdfplumber, Fitz**
  + **Enterprise: Adobe API Extract**
* **2. Website Scraping:**
  + **Open Source: BeuatifulSoup, requests**
  + **Enterprise: Apify**
* **Cloud Storage:** AWS S3 for structured organization and retrieval of processed files.
* **API Development:** FastAPI for managing file processing and data retrieval.
* **User Interface:** Streamlit for building an interactive front-end application.

### **Goal:**

The final objective is to implement and compare multiple approaches for extracting, processing, and structuring data from PDFs and web pages while ensuring extraction efficiency.

## **3. Problem Statement**

### **Challenge Overview**

Handling large volumes of unstructured data, such as PDFs and web pages, presents significant challenges in extraction, processing, and organization. Extracting meaningful information—especially structured elements like tables, images, and charts—requires efficient tools and methodologies. Additionally, determining whether open-source tools or enterprise solutions offer the best balance of cost, accuracy, and scalability is essential for making informed decisions.

### **Expected Outcome**

This project aims to develop a prototype system that:

* **Extracts data** from PDFs and web pages using both open-source and enterprise tools.
* **Converts and structures** the extracted content into Markdown format while preserving key elements.
* **Stores processed data** in AWS S3 using a well-organized and searchable structure.
* **Provides API endpoints** via FastAPI for automated processing.
* **Develops a user-friendly interface** using Streamlit for uploading and retrieving processed files.
* **Evaluates different tools**, comparing performance, cost, and usability to determine the most effective solution.

### **Constraints and Requirements**

* Both open-source tools (PyPDF2, pdfplumber, BeautifulSoup) and enterprise service (Adobe pdf extract, apify) must be utilized.
* Extracted content must retain important elements such as text, tables, and images.
* The system should implement a structured file organization method within AWS S3 for efficient storage and retrieval.
* API and front-end components must be developed using FastAPI and Streamlit, respectively.
* A detailed comparison of the tools must be conducted, highlighting key trade-offs in terms of cost, scalability, and ease of use.

## **4. Proof of Concept**

### **Our Approach and Why We Chose It**

To tackle the challenge of extracting, processing, and organizing unstructured data from PDFs and web pages, we are combining **open-source tools**, an **enterprise solution**, and **cloud storage**. Each technology was selected based on its ability to efficiently handle data extraction, formatting, and storage:

* **Extracting Data from PDFs and Webpages:**
  + *PyPDF2 & pdfplumber:* Help extract text, tables, and images from PDFs.
  + *BeautifulSoup :* Used for parsing web pages and extracting relevant content.
  + *Adobe Pdf Extract:* A powerful enterprise tool offering advanced OCR and structured data extraction.
* **Storing and Organizing Data:**
  + *AWS S3:* Used for storing and organizing processed files efficiently, ensuring easy retrieval.
* **Building the Application:**
  + *FastAPI:* Handles file uploads and data processing through API endpoints.
  + *Streamlit:* Provides an easy-to-use interface where users can upload PDFs and URLs and view results.

### **How We Tested Our Approach**

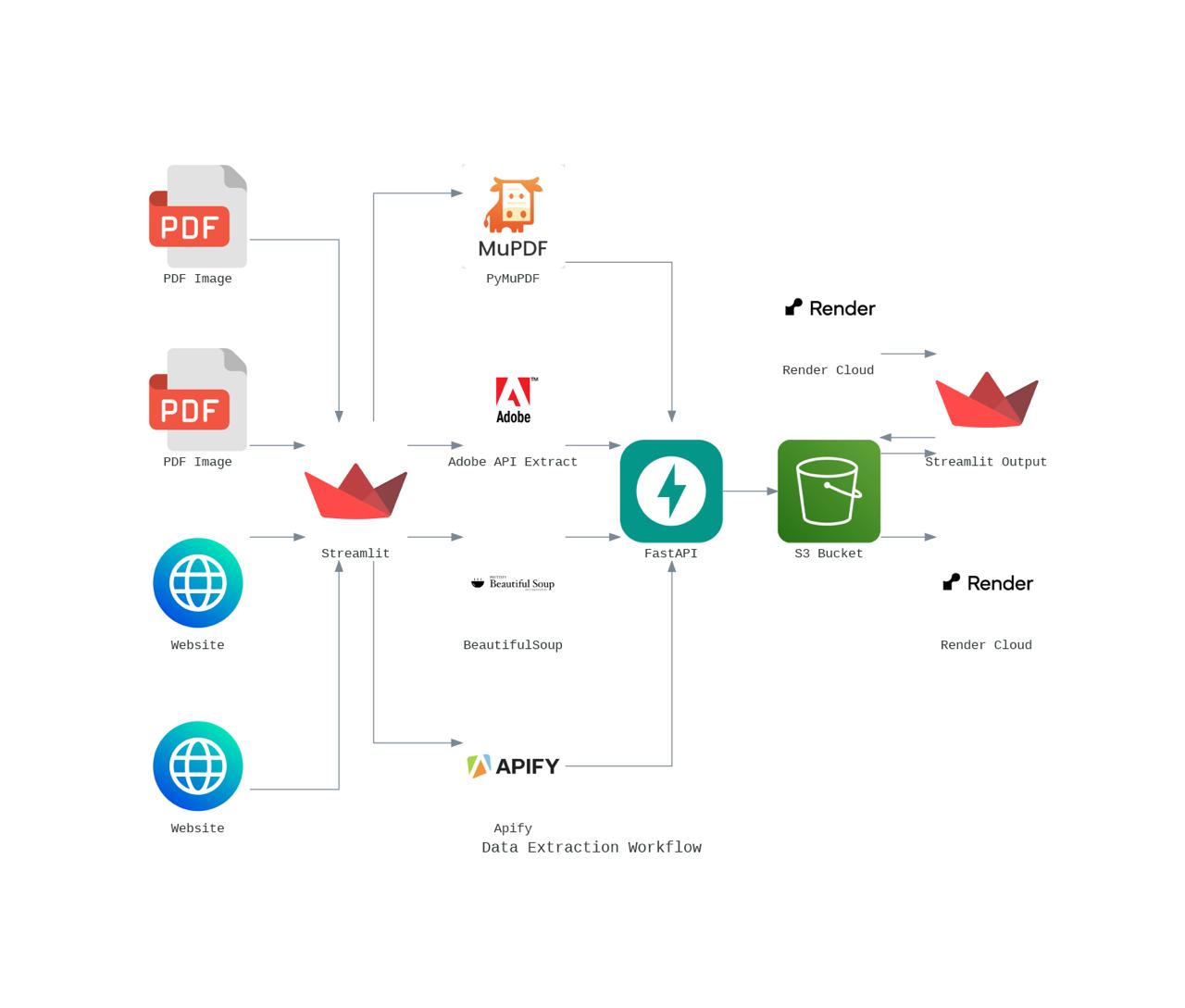
To make sure our solution works as expected, we ran a few initial tests:

1. **PDF Extraction:** We used PyPDF2 and pdfplumber to extract text and tables from sample PDFs and compared them with Microsoft Document Intelligence to check for accuracy.
2. **Web Scraping:** We extracted HTML content using BeautifulSoup and validated the results against Microsoft Document Intelligence.
3. **File Storage:** We set up AWS S3 with organized folders and tested metadata tagging for easy search.
4. **API Functionality:** We built a basic FastAPI endpoint for file uploads and processing, testing its response time and reliability.
5. **User Interface:** We developed a simple Streamlit UI where users can upload documents and see extracted content in Markdown format.

### **Challenges We Expect and How We Plan to Solve Them**

* **Extracting Complex Data:** Some PDFs and web pages may have intricate layouts that open-source tools struggle with. *Solution:* Use a mix of open-source and enterprise tools to get the best results.
* **Handling Large Data Volumes:** Storing and retrieving large amounts of processed data can slow down performance. *Solution:* Use structured storage and metadata tagging in S3 for efficient organization.
* **Integrating Different Tools:** Each tool outputs data differently, making integration tricky. *Solution:* Create a standard pipeline that formats extracted content consistently.
* **API Performance Issues:** Processing large files could lead to slow response times. *Solution:* Implement asynchronous processing in FastAPI to handle tasks efficiently.

## **5. Architecture Diagram**



## **6. Walkthrough of the Application**

This section walks you through how the application works, step by step, with snapshots illustrating each stage.

### **Step 1: Uploading a PDF or Webpage URL**

* Open the **Streamlit web app** in your browser.
* Choose one of the following options:
  + Upload a **PDF file** for processing.
  + Enter a **webpage URL** to extract content.
* Click the **Submit** button to start the extraction process.

*Screenshot:* Upload page showing file selection and URL input.

### **Step 2: Processing the Document**

* Once submitted, the file or URL is sent to the **FastAPI backend** for processing.
* The backend extracts data using:
  + **PyPDF2 & pdfplumber** for PDFs.
  + **BeautifulSoup** for web pages.
  + **Adobe API Extract** for enterprise-level processing.

*Screenshot:* Processing status in the API response or terminal output.

### **Step 3: Storing Processed Data**

* The extracted content is automatically saved in **AWS S3**, following a structured organization:
  + PDFs, images, and Markdown files are stored in separate folders.
  + Metadata is added for easy search and retrieval.

*Screenshot:* AWS S3 folder structure with stored files.

### **Step 4: Viewing Extracted Data**

* Once processing is complete, users can:
  + View the **extracted text and tables** in Markdown format.
  + Download the processed files.
  + Access images and other extracted elements.

*Screenshot:* User interface displaying extracted content in Markdown.

### **Step 5: Using the API (For Developers)**

* Developers can interact with the **FastAPI backend** to automate file uploads and retrieval.

Example API call using **cURL**:  
 curl -X POST "http://localhost:8000/upload\_pdf" -F "file=@sample.pdf"

* The response includes extracted text, metadata, and file locations.

*Screenshot:* Postman request and API response.

### **Step 6: Comparing Open-Source and Enterprise Tools**

**PDF Extraction: Adobe PDF Extract vs. PyMuPDF**

When extracting data from PDFs, **Adobe PDF Extract (Enterprise)** performed significantly better than **PyMuPDF (Open Source)**. Adobe extracted **text, images, and tables in the correct order**, making it highly reliable. In contrast, PyMuPDF had issues where **the extracted content was out of order** and contained **a lot of unnecessary elements**. Additionally, **table extraction was poor** in PyMuPDF, requiring **manual cleanup**. While Adobe is a **paid** tool, it offers **better accuracy and structured output**, whereas PyMuPDF, despite being free, needs **extra effort** to make the extracted content usable.

For website data extraction, **Apify (Enterprise)** provided **well-structured data** including **text, images, and links**. However, it did **not extract tables separately**, which was a minor limitation. On the other hand, **BeautifulSoup (Open Source)** extracted content but lacked **proper formatting and structure**, making the output **hard to read** without further processing. BeautifulSoup is free, but it **requires additional work** to clean and organize the extracted data, while Apify, though a **paid** service, delivers **higher-quality results** with less effort.

**Step 7: Application Workflow (Data Engineering & Backend Architecture)**

1. User Input (Frontend - Streamlit)
   * User uploads a PDF file or provides a website URL.
   * User selects the extraction method (open-source or enterprise).
2. API Request (Frontend → Backend)
   * Streamlit sends an API request to FastAPI (/extract/pdf/ or /extract/website/).
   * The request contains:
     + Uploaded PDF file (if PDF extraction).
     + Website URL (if website extraction).
     + Extraction method selection.
3. Backend Processing (FastAPI)
   * The request is processed based on the selected method:
     + Open-source method:
       - Uses pdfplumber & PyMuPDF for PDF text, images, and table extraction.
       - Uses BeautifulSoup for website text scraping.
     + Enterprise method:
       - Uses Adobe PDF Services for PDF content extraction.
       - Uses Apify Scraper for advanced website extraction.
4. Data Transformation & Markdown Generation
   * Extracted text, images, and tables are formatted into a Markdown (.md) file.
   * Extracted images are stored separately in S3.
5. Storage in AWS S3
   * Markdown files and images are uploaded to S3 (big-data-extraction-amv-assn-1)

**8. References**

Here is a comprehensive list of references for the tools, technologies, and resources that were used during the development of this project:

**Technologies and Tools**

1. **FastAPI**:  
   * [Official Documentation](https://fastapi.tiangolo.com/)
   * [Getting Started with FastAPI](https://fastapi.tiangolo.com/tutorial/)
   * [FastAPI GitHub Repository](https://github.com/tiangolo/fastapi)
2. **Streamlit**:  
   * [Official Documentation](https://docs.streamlit.io/)
   * [Streamlit Tutorials](https://docs.streamlit.io/library/get-started/create-an-app)
   * [Streamlit GitHub Repository](https://github.com/streamlit/streamlit)
3. **PDF Processing Tools**:  
   * PyMuPDF (fitz):
     + [Official Documentation](https://pymupdf.readthedocs.io/)
     + [PyMuPDF GitHub Repository](https://github.com/pymupdf/PyMuPDF)
   * pdfplumber:
     + [Official Documentation](https://pdfplumber.readthedocs.io/)
     + [pdfplumber GitHub Repository](https://github.com/jsvine/pdfplumber)
4. **Web Scraping Tools**:  
   * BeautifulSoup:
     + [BeautifulSoup Documentation](https://www.crummy.com/software/BeautifulSoup/bs4/doc/)
     + [Python Requests Library](https://docs.python-requests.org/en/latest/)
   * [HTML Parsing with BeautifulSoup (Tutorial)](https://realpython.com/beautiful-soup-web-scraper-python/)
5. **AWS S3**:  
   * [Boto3 Documentation](https://boto3.amazonaws.com/v1/documentation/api/latest/index.html)
   * [Amazon S3 Developer Guide](https://docs.aws.amazon.com/AmazonS3/latest/dev/Introduction.html)
   * [Boto3 GitHub Repository](https://github.com/boto/boto3)

#### **Additional Resources**

1. **Community Resources**:  
   * [FastAPI Discussions (GitHub)](https://github.com/tiangolo/fastapi/discussions)

## **9. Disclosures**

### **Team Member Contribution Attestation**

This project was collaboratively developed by our team members, with each member contributing to different aspects of the implementation. Below is a breakdown of responsibilities:

| **Team Member** | **Contributions** |
| --- | --- |
| **All** | Project planning, system architecture design, and overall coordination of the development process. |
| **Vemana, Ashwin** | Development of the FastAPI backend, including API endpoints for PDF and website processing (Open Source & Enterprise). |
| **Madhura** | Implementation of **Open Source PDF extraction** using PymuPDF and pdfplumber, handling text, tables, and images. |
| **Vemana** | Implementation of **Enterprise PDF extraction**, using Adobe pdf extract for structured data extraction. |
| **Madhura, Ashwin** | Development of **Open Source Website scraping** functionality using BeautifulSoup, ensuring proper content extraction. |
| **Vemana** | Implementation of **Enterprise Website scraping using Apify**, handling structured website data using enterprise-grade APIs. |
| **All** | AWS S3 integration, managing cloud storage for extracted files, and ensuring structured data organization for both Open Source & Enterprise. |
| **Madhura,  Vemana** | Documentation, report writing, workflow diagrams, and technical explanations for assignment submission.  Contributions Team Member 1: 33% Team Member 2: 33% Team Member 3: 33% |

All team members actively participated in discussions, debugging sessions, and review meetings to ensure the successful completion of the project.We, the members of this team, confirm that this submission is our original work and that we have accurately represented our contributions.

### 

### **10 AI Usage Disclosure**

While working on this project, we used a few different tools to help streamline the development process and ensure accuracy. ChatGPT was a great resource for writing and refining code, fixing errors, setting up API endpoints, and improving the integration between FastAPI and Streamlit. I also used Perplexity AI to research technical solutions, cross-check information, and gather relevant resources to make sure everything was on the right track. Additionally, DeepSeek helped with analyzing technical documentation and enhancing the data extraction process. These tools made the development process smoother and more efficient, helping me troubleshoot issues faster and build a more reliable application.