**Task Sheet:Internship Task: Domain-Specific PDF Summarization & Keyword Extraction Pipeline**

Problem Statement:-

1**. PDF Ingestion & Parsing**

Requirement : The pipeline should be able to process multiple PDFs from a folder on the desktop, i.e. if the path of a folder is provided in the code, it should ingest all the documents in that folder. It must handle documents of varying lengths:

- Short PDFs (1-10 pages)

- Medium PDFs (10-30 pages)

- Long PDFs (30+ pages)

2. **MongoDB Dataset Storage & JSON Updates**

Initial Storage: When each PDF is ingested, its metadata (document name, path, size, etc.) must be stored in a MongoDB collection.

Post-Processing Update: After summarization and keyword extraction, the MongoDB entry for each document must be updated with the JSON output, including the generated summary and extracted keywords.

3. **Summarization & Keyword Extraction**

Summarization: Dynamically generate summaries that are relevant to the domain you have chosen. The summary length and detail should correspond to the document length (e.g., a detailed summary for long documents, concise summaries for short ones).

Keyword Extraction : Extract non-generic , domain-specific keywords that reflect key ideas or themes of the document. Avoid common or irrelevant keywords.

4. **JSON Structure & MongoDB Updates**

JSON Format : Summaries and keywords must be formatted in JSON, which will then be stored in the MongoDB document. Make sure to handle updates efficiently after processing each document.

Error Handling : Log any errors (e.g., for corrupted PDFs or unsupported formats) and ensure that MongoDB records are not affected by such issues.

5. **Concurrency & Performance**

Concurrency : The pipeline should be designed to handle multiple documents simultaneously, leveraging parallel processing to improve speed.

Performance : Provide data on how well the system scales, especially in terms of how quickly it processes large and multiple PDFs concurrently

Before coming to problem statement ,I setup all the required IDEs for ex-Spyder,MongoDB,GitHub.

1st Requirement

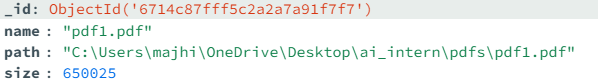
**PDF Ingestion & Parsing:-**

* First of all I Read the Json file programatically, there are 18 total pdf link given .
* From its link I download all the pdf file sequentially.
* Creating a separate folder where all extracted pdf are stored.

2nd Requirement

**MongoDB Dataset Storage & JSON Updates:-**

* Installing MongoDB.
* Setup a new database .
* Setup a new collection.
* Through programmatically get the meta data like name , path , size.
* Store that metadata in newly created database collection.
* It looks like below image reference.

****

3rd Requirement

**Summarization & Keyword Extraction:-**

1. **Text** **Extraction** **from** **PDF**:  
   The extract\_text\_from\_pdf function reads a PDF file and extracts its text content. It utilizes PyPDF2 to read each page of the PDF and combines the extracted text into a single string.
2. **Text** **Cleaning** **and** **Tokenization**:  
   The clean\_and\_tokenize function processes the extracted text by converting it to lowercase, removing punctuation, and eliminating common stopwords. This prepares the text for keyword extraction and summarization by splitting it into individual tokens (words).
3. **Text** **Summarization**:  
   The summarize\_text function generates a summary of the extracted text. It calculates word frequencies and scores sentences based on the importance of their words. The function selects a portion of the most significant sentences to create the summary, determined by the summary\_ratio parameter.
4. **Keyword** **Extraction**:  
   The extract\_keywords function identifies the most frequent and important words in the text. It returns a list of the top n keywords based on their frequency, with a default of the top 10 keywords.
5. **Saving** **Summaries** **and** **Keywords**:  
   The save\_to\_json function stores the generated summary and keywords in a JSON file. The filename is based on the original PDF's name, and the file is saved in the specified output directory.
6. **Batch** **Processing** **of** **Multiple** **PDFs**:  
   The process\_multiple\_pdfs function handles multiple PDF files from a given directory. For each PDF, it extracts text, summarizes the content, identifies keywords, and saves the results in JSON format. It processes each PDF file individually in a loop.

4th Requirement

**JSON Structure & MongoDB Updates:-**

1. **MongoDB Connection Setup**: The script begins by establishing a connection to a MongoDB instance running on localhost. It accesses a specific database (AI\_Intern) and collection (Extracted) where the PDF metadata is stored.
2. **Directory and File Iteration**: It specifies a directory (json\_directory) where the JSON files containing summaries and keywords are stored. The script then iterates through all the files in this directory, looking specifically for those with a .json extension.
3. **Reading and Parsing JSON Files**: For each JSON file, the script reads the file and loads its content into a Python dictionary using the json module. This step allows the script to access the summary and keywords for each PDF.
4. **PDF Name Matching**: The JSON filenames are assumed to correspond to the PDF files, so the script replaces the .json extension in the filename with .pdf to match the correct PDF metadata in the MongoDB collection.
5. **Updating MongoDB**: The script constructs an update query that searches for a MongoDB document with the name field matching the PDF filename. It updates the document by adding or replacing the summary and keywords fields with the data from the JSON file.

5th Requirement

**Concurrency & Performance:-**

**PDF Downloading from JSON (Concurrency and Performance Considerations)**

This section of the code handles downloading PDF files using URLs provided in a JSON file. The key considerations for this process are concurrency and error handling.

* **Concurrency**:
  + The script uses Python's ThreadPoolExecutor from the concurrent.futures module to download multiple PDFs concurrently.
  + This improves speed by allowing up to 5 PDFs to be downloaded at the same time (max\_workers=5), rather than downloading them one by one.
* **Performance**:
  + Concurrency ensures faster processing, especially when dealing with large datasets. The system should handle a substantial number of downloads efficiently.
  + The ThreadPoolExecutor allows the system to scale when processing a larger number of PDFs, distributing tasks across multiple threads.