STRING MATCHING SYSTEM

HIGH LEVEL OVERVIEW AND FUNCTIONS

DATA STORING PROCESS :

- Data chunking in PostgreSQL or python or JavaScript
- Parallel processing the data chunks :
  - Using python's multiprocessing library
  - Using native PostgreSQL parallel processing
  - Using Task queues with Celery
- Pre-Processing the data chunks by data transformation :
  - Normalizing the text
  - Standardizing the text
- Compressing the data before storing in PostgreSQL
- Table partitioning and batch inserts of data in PostgreSQL database
- Handling batch insertions failures using chunk id in PostgreSQL :
  - Checkpointing using chunk ids
  - Using idempotent operations to prevent adverse effects on database health
  - Logging and Monitoring database health using Prometheus or Grafana

DATA RETRIEVAL AND MATCHING PROCESS :

- Indexing and Partitioning of tables :
  - Full-text indexing for text-heavy queries
  - Composite indexing for combining columns which are frequently queried
- Creating Materialized Views for faster query retrieval
- Caching
- Parallel Processing using task queues for processing chunks of data queried
- Implementing Boyer-Moore algorithm for each text chunk retrieved by the query

Doubts :

1. How to use Task queues and Task workers for managing data chunks using Kafka.
2. How to do parallel processing of Queries and Tasks in PostgreSQL.
3. How to handle Concurrency of all the parallel queries working with the database.

ACHIEVABLE PROJECT FUNCTIONALITIES AND WORKFLOW

DATA STORING PROCESS :

- Extracting Text content from pdf or word file using Python libraries.
- Server-Side data chunking (Partial Data chunking)
- Pre-Processing the data chunks by data transformation :
  - Normalizing the text
  - Standardizing the text

- Compressing each chunk of text data using Multithreading
- Using Asyncio Requests to insert each chunk of compressed data in PostgreSQL
- Handling batch insertions failures using chunk id in PostgreSQL :
  - Checkpointing using chunk ids
  - Using idempotent operations to prevent adverse effects on database health
  - Logging and Monitoring database health using Prometheus or Grafana
- Table partitioning and batch inserts of data in PostgreSQL database from Original Table
- Re-Handling batch insertions failures using chunk id in PostgreSQL :
  - Checkpointing using chunk ids
  - Using idempotent operations to prevent adverse effects on database health
  - Logging and Monitoring database health using Prometheus or Grafana
- Indexing on each database record for faster data querying in Server by looping the PostgreSQL procedure.
  - Full-text indexing for text-heavy queries
  - Composite indexing for combining columns which are frequently queried
- Creating Materialized Views for faster query retrieval

DATA RETRIEVAL AND MATCHING PROCESS :

- Using Pagination to retrieve chunks of data using Asynchio Requests in python server
- Using Multithreading to decompress the retrieved data and re-chunking data into smaller chunks
- Implementing Boyer-Moore algorithm for each text chunk