Day 2: Document Processing

Handling and Preparing Your Data for RAG

On Day 2, we focus on the critical first step of any RAG system: processing your documents. This involves getting your data into a usable format, extracting the relevant text, and preparing it for embedding.

1. Document Upload System

The project includes a robust system for uploading various document types. This is the entry point for your external knowledge.

Key Aspects:

- **Supported Formats:** The system supports PDF, DOCX, and TXT files.
- **User Interface:** A web interface allows users to easily upload documents.
- **Storage:** Uploaded files are stored securely on the server (typically in the `storage` directory).
- **Real-time Progress:** Users get real-time feedback on the upload progress.

Implementation Details:

- File uploads are handled by Laravel controllers (likely in `app/Http/Controllers/DocumentController.php`).
- Storage configuration is managed in `config/filesystems.php`.
- Frontend components (likely in `resources/js/Components/`) handle the file selection and progress display.

2. Text Extraction

Once a document is uploaded, the system needs to extract the raw text content, regardless of the original file format.

Methods Used:

- **PDF:** Utilizes libraries like `pdftotext` (via command line) or native PHP PDF parsers.
- **DOCX:** Employs the `PhpWord` library, with a fallback mechanism using `ZipArchive` for compatibility.
- **TXT:** Direct reading of the file content.

Process:

- The system identifies the file type.
- It selects the appropriate extraction method.
- Extracts the text content.
- Handles potential errors during extraction.

Code Reference:

- Look for logic related to file type detection and text extraction in service classes (e.g., `app/Services/DocumentProcessor.php` or similar) or controllers.

3. Content Preprocessing

Raw text needs to be cleaned and prepared before it can be effectively used for embeddings and retrieval. This often involves cleaning and chunking.

Steps Involved:

- **Text Cleaning:** Removing unnecessary characters, whitespace, headers, footers, or other noise that doesn't contribute to the document's core content.
- **Chunking:** Breaking down the extracted text into smaller, manageable pieces (chunks). This is crucial because embedding models and LLMs have input token limits. Effective chunking ensures relevant context is kept together.
    - **Strategies:** Chunking can be done by fixed token size, sentence boundaries, paragraph breaks, or even more sophisticated methods that preserve semantic meaning.
- **Metadata Extraction:** Identifying and storing relevant metadata about the document (e.g., title, author, source, project association). This metadata can be useful for filtering and improving retrieval.

Implementation Details:

- Preprocessing logic is likely within service classes responsible for document handling.
- Chunking strategies are implemented based on the chosen method.
- Metadata is stored alongside the document content, possibly in the database or Elasticsearch.

Code Reference:

- Examine service classes for text cleaning and chunking functions.
- Check database migration files for metadata fields in the documents table.

By the end of Day 2, you will have a system capable of ingesting various document types, extracting their text content, cleaning it, and breaking it into suitable chunks, ready for the next step: generating embeddings and storing them in a vector database.