Day 3: Embeddings & Vector Storage

Transforming Text into Searchable Vectors

Day 3 focuses on converting the processed document chunks into numerical representations called embeddings and storing them in a vector database. This step is essential for enabling semantic search capabilities.

1. Understanding Embeddings

Embeddings are dense vector representations of text that capture semantic meaning. Words or phrases with similar meanings will have embeddings that are closer to each other in a multi-dimensional space.

Key Concepts:

- **Vector Representations:** Text is transformed into a list of numbers (a vector).
- **Semantic Similarity:** The distance between vectors indicates the semantic similarity between the original text pieces.
- **Dimensionality:** Embeddings have a fixed size (e.g., 1536 dimensions for OpenAI's `text-embedding-ada-002`).

How Embeddings Work in RAG:

- Document chunks are converted into embeddings.
- User queries are also converted into embeddings.
- The system searches for document chunk embeddings that are most similar to the query embedding.

2. Vector Database Setup

Elasticsearch is used as the vector database in this project to store and index the generated embeddings.

Elasticsearch Configuration:

- Ensure your Elasticsearch instance is running (as set up in Day 1 using Docker Compose).
- You need to create an index in Elasticsearch specifically for storing your document embeddings.

Index Mapping:

- The index mapping defines the structure of the documents stored in the index.
- A key field in the mapping will be of type `dense_vector` to store the embeddings.
- Other fields will store the original text chunk, metadata (like document ID, title, project ID), etc.

Example Index Mapping (Conceptual):

```
PUT /document_embeddings
{
  "mappings": {
    "properties": {
      "embedding": {
```

```
          "type": "dense_vector",
          "dims": 1536, // Example dimension, should match your embedding model
          "index": true,
          "similarity": "cosine"
        },
        "content": {
          "type": "text"
        },
        "document_id": {
          "type": "keyword"
        },
        "metadata": {
          "type": "object"
        }
      }
    }
}
```

Code Reference:

- Look for code that interacts with the Elasticsearch API, particularly for index creation and mapping (likely in a service provider, a dedicated setup command, or the RagService).

3. Implementation: Embedding Generation and Storage

This section covers the process of generating embeddings from the processed text chunks and storing them in Elasticsearch.

Embedding Generation:

- The system uses the OpenAI API to generate embeddings.
- For each text chunk, an API call is made to the OpenAI embedding endpoint.
- The API key is retrieved from the `.env` file.

Code Example (Conceptual - RagService):

```php
<?php

namespace App\Services;

use OpenAI\Client;
use Elasticsearch\Client as ElasticsearchClient;

class RagService
{
    protected $openai;
    protected $elasticsearch;

    public function __construct(Client $openai, ElasticsearchClient
$elasticsearch)
    {
```

```
        $this->openai = $openai;
        $this->elasticsearch = $elasticsearch;
    }

    public function generateEmbedding(string $text)
    {
        $response = $this->openai->embeddings->create([
            'model' => 'text-embedding-ada-002',
            'input' => $text,
        ]);

        return $response->embeddings[0]->embedding;
    }

    public function storeEmbedding(string $documentId, array $chunk, array
$embedding)
    {
        $this->elasticsearch->index([
            'index' => 'document_embeddings',
            'id'    => $documentId . '_' . $chunk['chunk_id'], // Unique ID for
the chunk
            'body'  => [
                'document_id' => $documentId,
                'content'     => $chunk['text'],
                'embedding'   => $embedding,
                'metadata'    => $chunk['metadata'],
            ]
        ]);
    }

    // ... other RagService methods
}
```

Vector Storage:

- Once an embedding is generated for a chunk, it is sent to Elasticsearch.
- Each chunk's data (original text, embedding, metadata) is indexed as a document in the configured Elasticsearch index.

Process Flow:

- The document processing pipeline (from Day 2) feeds chunks to the embedding generation process.
- Embeddings are generated for each chunk.
- Embeddings and associated data are sent to Elasticsearch for indexing.

By the end of Day 3, your system will be able to transform processed document content into searchable vectors and store them efficiently in Elasticsearch, laying the groundwork for the retrieval system.