Day 4: Retrieval System

Finding Relevant Information with Semantic Search

Day 4 is dedicated to building the core retrieval system, which allows the RAG application to find the most relevant document chunks based on a user's query. This is where the power of the vector database comes into play.

1. Search Implementation

The retrieval system's primary function is to search the vector database (Elasticsearch) for document chunks that are semantically similar to the user's query.

Types of Search:

- **Semantic Search:** This is the main method, utilizing the embeddings to find content with similar meaning, even if the exact keywords aren't present.
- **Keyword Search:** Traditional keyword-based search can be used in conjunction with semantic search for hybrid approaches.
- **Hybrid Search:** Combining semantic search (vector search) and keyword search (text search) can often yield more accurate and comprehensive results.

How Semantic Search Works:

1. The user's query is converted into an embedding (similar to how document chunks were embedded in Day 3).
2. A search query is constructed for Elasticsearch to find documents with `dense_vector` fields (embeddings) that are closest in vector space to the query embedding.
3. Elasticsearch returns the top-N most similar document chunks.

Code Reference:

- Look for search query construction and execution logic within the `RagService` or a dedicated search service class.

2. Query Processing

Processing the user's query before performing the search is crucial for effective retrieval.

Steps Involved:

- **Query Embedding:** The user's natural language query is sent to the embedding model (OpenAI) to generate its vector representation.
- **Context Retrieval:** The retrieved document chunks (from the search) provide the context that will be used by the LLM for generating a response.
- **Result Ranking:** The search results from Elasticsearch are typically ranked by similarity score. The system selects the top-ranked chunks to form the context.

Code Example (Conceptual - RagService search method):

```
<?php

namespace App\Services;

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

class RagService
{
    protected $openai;
    protected $elasticsearch;

    // ... constructor and generateEmbedding, storeEmbedding methods ...

    public function search(string $query, int $k = 5)
    {
        // 1. Generate query embedding
        $queryEmbedding = $this->generateEmbedding($query);

        // 2. Construct Elasticsearch search query for vector search
        $searchParams = [
            'index' => 'document_embeddings',
            'body'  => [
                'query' => [
                    'knn' => [
                        'embedding' => [
                            'vector' => $queryEmbedding,
                            'k' => $k,
                            'num_candidates' => $k * 10 // Or a larger number for
better recall
                        ]
                    ]
                ],
                '_source' => ['content', 'metadata'] // Retrieve necessary fields
            ]
        ];

        // Add keyword search part for hybrid search (optional)
        // if (!empty($query)) {
        //     $searchParams['body']['query'] = [
        //         'bool' => [
        //             'must' => [
        //                 ['match' => ['content' => $query]] // Keyword search
        //             ],
        //             'should' => [
        //                 $searchParams['body']['query'] // Vector search
        //             ]
        //         ]
        //     ];
        // }

        // 3. Execute search and retrieve results
        $response = $this->elasticsearch->search($searchParams);

        // Extract and return the relevant document chunks (context)
        $context = [];
        foreach ($response['hits']['hits'] as $hit) {
            $context[] = $hit['_source'];
        }

        return $context;
    }

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

3. Performance Optimization

Optimizing the retrieval system's performance is crucial for a responsive RAG application.

Strategies:

- **Elasticsearch Index Optimization:** Proper index mapping, shard allocation, and replica settings can improve search speed.
- **Query Optimization:** Fine-tuning Elasticsearch queries, including the `k` and `num_candidates` parameters for k-NN search.
- **Caching:** Caching frequently accessed document chunks or query results.
- **Hardware:** Ensuring sufficient resources (CPU, RAM, storage) for both the application server and Elasticsearch.

Code Reference:

- Configuration for Elasticsearch connection pooling and timeouts.
- Logic for implementing caching mechanisms.

By the end of Day 4, your RAG system will be able to effectively retrieve relevant document chunks based on user queries, providing the necessary context for the language model to generate informed responses.