Smarter Search with Spring AI and Neo4j

Akmal Chaudhri

Why This Project?

The Problem

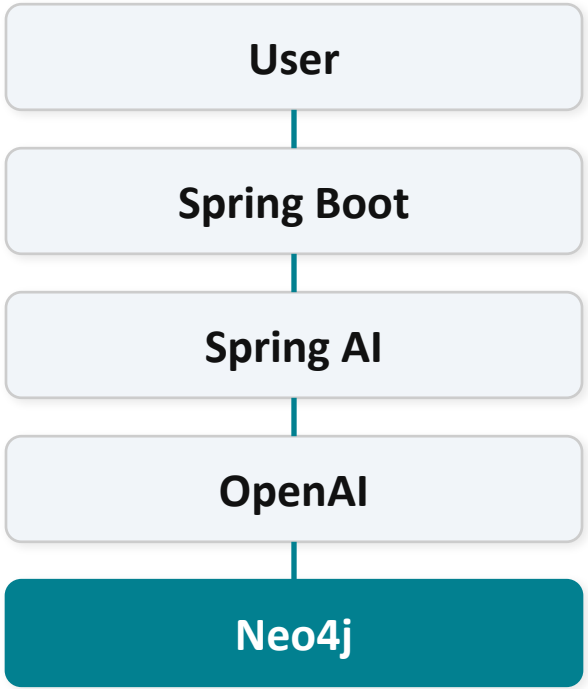
Traditional keyword search struggles with intent and context.

Users say things like:
"I want an uplifting book"
"Books with a happy ending"
"Something encouraging"

Key Questions

- Can an LLM understand the request?
- Can vector search improve results?
- Can graph relationships improve results?

Architecture Overview



Graph search · Vector search · Goodreads data

The Goodreads Dataset

10,000	12,371	69,791	44,827
Books	Authors	Reviews	Users

Graph Model

Reviews also carry a 1,536-dimension OpenAI embedding vector, enabling semantic similarity search.

What is RAG?

Retrieval-Augmented Generation

Why RAG?

- LLMs hallucinate when they don't have the right context.
- RAG retrieves relevant data first, then passes it to the model as context.
- The result is grounded, accurate and explainable - the model answers from real data.
- Neo4j enables two retrieval mechanisms: vector similarity and graph traversal.

VectorRAG vs GraphRAG

VectorRAG /vector

Query: "I want something uplifting"

Phrase > Embedding > Cosine similarity > Review nodes

- Semantic matching
- Handles fuzzy language
- No exact keyword matching needed

GraphRAG /graph

Query: "Find books about encouragement"

Vector search > Review nodes > WRITTEN_FOR > Book nodes

- Explicit relationships
- Structured context
- Explainable retrieval

Spring AI

Spring-style abstractions for AI applications

- Chat Client
- Embeddings
- Vector Store
- RAG Patterns
- Prompt Templates

```
// Simple LLM call - no retrieval
client.prompt()
      .user(question)
      .call()
      .content();
```

Application Structure

```
com.jmhreif.springaigoodreads
├── BookController.java
├── BookRepository.java
├── Book.java
├── Review.java
└── SpringaiGoodreads
    Application.java
```

BookController

REST endpoints - /hello, /llm, /vector, /graph

BookRepository

Spring Data Neo4j - graph query for books by review IDs

Book / Review

Domain model - mapped to Neo4j nodes

Neo4jVectorStore

Spring AI - wraps the Neo4j vector index for similarity search

ChatClient

Spring AI - wraps OpenAI for prompt and response handling

The Four Endpoints

`/hello`	Simple LLM call. No retrieval. Verifies model connectivity. *Baseline - pure model response*
`/llm`	Sends the search phrase to the LLM with no context from Neo4j. *Fast but prone to hallucination*
`/vector`	Embeds the phrase, runs vector similarity search on Review nodes, passes results to LLM. *Semantic matching on review text*
`/graph`	Vector search > retrieve related Book nodes via graph > LLM with full context. *Full GraphRAG pipeline*

Neo4j Vector Index

Each Review node stores a 1,536-dimension embedding generated by OpenAI. Neo4j indexes these vectors for fast cosine similarity search.

```
CREATE VECTOR INDEX `review-text` IF NOT EXISTS
FOR (n:Review) ON (n.embedding)
OPTIONS { indexConfig: {
  `vector.dimensions`: 1536,
  `vector.similarity_function`: 'cosine'
}};
```

How Spring AI uses it

- `vectorStore.similaritySearch(SearchRequest.builder().query(searchPhrase).build())`
- Spring AI embeds the phrase via OpenAI, then queries the "review-text" index automatically.

Vector Search vs Graph Search

Feature	Vector Search	Graph Search
Relationship-aware	✗	✓
Semantic matching	✓	Limited
Explainable	Partial	✓
Handles fuzzy intent	✓	Partial
Structured reasoning	Partial	✓

These two approaches are complementary - the /graph endpoint combines both.

Lessons Learned

Vector index naming matters

Spring AI expects a specific index name. The default is "spring-ai-document-index" but this project requires "review-text". Mismatch causes a runtime error.

Review nodes need id and text properties

Spring AI maps vector search results to Document objects using properties named "id" and "text". Without these, results come back empty even when embeddings exist.

Pre-generated embeddings save time

Generating embeddings at runtime via OpenAI costs tokens and time. Loading pre-computed embeddings from a file is faster and cheaper for demo setup.

Data loading takes patience

69,791 review embeddings take some minutes to load via Cypher. Plan for this in demo prep - it only needs to be done once.

Live Demo

1. Show Neo4j graph - browse books and reviews
2. Run /hello - verify LLM connectivity
3. Run /llm - no context, note limitations
4. Run /vector - vector-only search
5. Run /graph - full GraphRAG pipeline
6. Compare /vector vs /graph outputs

Key Takeaways

- Spring AI makes AI development familiar for Java developers - same patterns, same idioms.
- Neo4j supports both vector similarity and graph traversal in a single database.
- VectorRAG and GraphRAG solve different problems and work best in combination.
- Grounding LLM responses in real data reduces hallucinations and improves reliability.
- **The Goodreads demo is a practical, reusable pattern for enterprise AI architecture.**

Resources

- GitHub: github.com/JMHReif/springai-goodreads
- GraphAcademy: graphacademy.neo4j.com
- AuraDB: neo4j.com/product/auradb/
- Doc: neo4j.com/docs/getting-started/languages-guides/java/spring-data-neo4j/

Thank you!