# Capítulo 19 – O RAG Corporativo (Spring AI na Prática)

Aléssio Miranda Jr.

Outubro - 2026/2

**Capítulo em Revisão**

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

## O que você verá neste capítulo

Neste capítulo, construiremos o sistema RAG completo dentro do ecossistema Spring: desde a ingestão de documentos PDF no Vector Store, passando pela configuração do `QuestionAnswerAdvisor` do Spring AI, até a exposição de uma API REST que responde perguntas fundamentadas em documentos internos da empresa. Formalizaremos o conceito de *Advisors* como middleware de enriquecimento de prompt, e discutiremos as estratégias de reranking para melhorar a precisão dos resultados.

## 1 De Python Artesanal para Java Corporativo

Na Aula 17, construímos um RAG artesanal em Python: buscamos documentos no ChromaDB, concatenamos manualmente ao prompt, e chamamos a API. Funcionou, mas era frágil: sem tipagem, sem tratamento de erros, sem injeção de dependências.

O Spring AI automatiza todo esse fluxo com o conceito de **Advisors** — componentes que interceptam e enriquecem o prompt **antes** de enviá-lo ao LLM:

**▶ Prática: title=RAG com QuestionAnswerAdvisor**

```java
@RestController
public class RagController {

    private final ChatClient chatClient;

    public RagController(ChatClient.Builder builder,
                         VectorStore vectorStore) {
        this.chatClient = builder
            .defaultAdvisors(
                new QuestionAnswerAdvisor(vectorStore)
            )
            .build();
    }

    @PostMapping("/perguntar")
    public String perguntar(@RequestBody String pergunta) {
        return chatClient.prompt()
            .user(pergunta)
            .call()
            .content();
    }
}
```

O `QuestionAnswerAdvisor` faz **automaticamente**: (1) vetoriza a pergunta, (2) busca os Top-K documentos no `VectorStore`, (3) injeta o contexto no prompt, e (4) instrui o LLM a responder com base apenas no contexto.

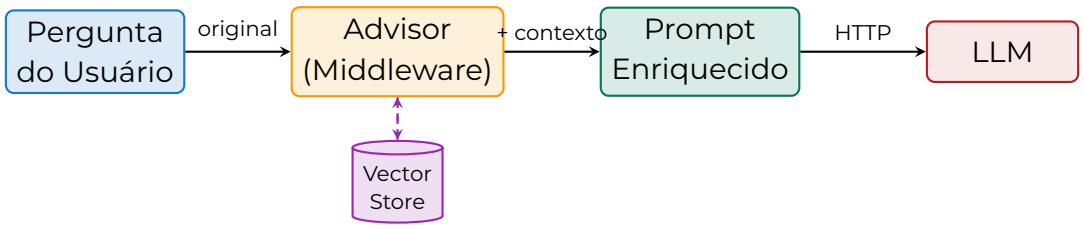


Figura 1: O Advisor intercepta a pergunta do usuário, busca documentos relevantes no Vector Store, e injeta o contexto no prompt antes de enviá-lo ao LLM. Funciona como um "middleware" de enriquecimento.

## 2 Ingestão de Documentos no Vector Store

Antes de perguntar, precisamos popular o Vector Store com os documentos da empresa. O Spring AI oferece `DocumentReader`s e `DocumentTransformer`s para essa pipeline:

```java
@Bean
CommandLineRunner ingestao(VectorStore vectorStore) {
    return args -> {
        // 1. Ler o PDF
```

```
        var reader = new PagePdfDocumentReader(
            "classpath:manual-tecnico.pdf"
        );
        List<Document> documentos = reader.get();

        // 2. Fatiar em chunks (Chunking)
        var splitter = new TokenTextSplitter();
        List<Document> chunks = splitter.apply(documentos);

        // 3. Inserir no Vector Store (auto-embedding)
        vectorStore.add(chunks);
        System.out.println("Ingeridos: " + chunks.size());
    };
}
```

## 3 Reranking: Refinando a Qualidade

A busca vetorial retorna os Top-K documentos por similaridade de cosseno, mas nem sempre o mais "similar" vetorialmente é o mais **relevante** para a pergunta. O **Reranking** é uma segunda etapa que reordena os candidatos usando um modelo especializado:

**• Teoria: title=Two-Stage Retrieval**

1. **Stage 1 — Retrieval:** Busca rápida no Vector Store (ANN). Retorna Top-20 candidatos em milissegundos.
2. **Stage 2 — Reranking:** Um modelo Cross-Encoder (ex: `ms-marco-MiniLM`) avalia a relevância de cada par (pergunta, documento) individualmente. Reordena e seleciona os Top-3 finais.

O Stage 1 prioriza **velocidade** (busca aproximada). O Stage 2 prioriza **precisão** (avaliação exata). A combinação gera o melhor trade-off entre latência e qualidade.

## 4 Configuração do Vector Store no Spring

O Spring AI suporta múltiplos Vector Stores via starters:

- **ChromaDB:** `spring-ai-chroma-store-spring-boot-starter`
- **PGVector (PostgreSQL):** `spring-ai-pgvector-store-spring-boot-starter` — Ideal para quem já usa PostgreSQL.
- **Qdrant:** `spring-ai-qdrant-store-spring-boot-starter`
- **Redis:** `spring-ai-redis-store-spring-boot-starter`

No `application.properties`:

```
# Usar ChromaDB local
spring.ai.vectorstore.chroma.url=http://localhost:8000
spring.ai.vectorstore.chroma.collection-name=documentos

# Modelo de Embedding
spring.ai.openai.embedding.options.model=text-embedding-3-small
```

## 5 Conclusão

O RAG corporativo com Spring AI transforma a infraestrutura de busca vetorial e geração de texto em um **componente Spring gerenciado**: testável, injetável, configurável via properties e integrável com o restante do ecossistema (Spring Security, Spring Data, Spring Cloud). Na próxima aula, adicionaremos o "Fator Uau!" — a **interface de usuário** que consome essa API e apresenta os resultados de forma interativa.

### ✔ Takeaways

- O `QuestionAnswerAdvisor` do Spring AI automatiza o fluxo RAG completo: vetoriza a pergunta, busca Top-K, injeta contexto e instrui o LLM — tudo em uma única linha de configuração.
- **Advisors** são o equivalente Spring a "middlewares de enriquecimento de prompt": interceptam a pergunta antes de enviá-la ao LLM.
- A **ingestão de documentos** segue o pipeline: `DocumentReader` → `TokenTextSplitter` (Chunking) → `VectorStore.add()` (auto-embedding).
- O **Reranking** (Two-Stage Retrieval) combina busca rápida ANN (Top-20) com avaliação precisa Cross-Encoder (Top-3), otimizando o trade-off latência vs. qualidade.
- O Spring AI suporta múltiplos Vector Stores via starters: ChromaDB, PGVector, Qdrant, Redis — trocáveis com uma linha no `application.properties`.
- A configuração por properties externaliza modelo de embedding, URL do Vector Store e collection name, separando infraestrutura de lógica de negócio.

## ✎ Questões: Autoavaliação

1 **[Direta]** Explique o papel do `QuestionAnswerAdvisor` no Spring AI. Quais etapas do RAG ele automatiza internamente?

2 **[Direta]** Descreva o pipeline de ingestão de documentos no Spring AI: quais classes são usadas para ler PDFs, fatiar texto e inserir no Vector Store?

3 **[Direta]** Compare a estratégia de Retrieval simples (busca vetorial direta) com o Two-Stage Retrieval (busca + reranking). Quando o reranking é indispensável?

4 **[Reflexiva]** O `TokenTextSplitter` fatia documentos por contagem de tokens. Discuta cenários em que essa estratégia falha (ex: tabelas, listas, código-fonte) e proponha alternativas.

5 **[Reflexiva]** Uma empresa migrou seu RAG de ChromaDB local para PGVector na nuvem. Quais trade-offs de custo, latência e persistência ela deve considerar?

# Lab. 19: RAG Corporativo no Spring AI

**Capítulo em Revisão**

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

## ⋆ Objetivo do Laboratório

O objetivo é reproduzir a mágica da injeção de contexto (RAG) da Aula 17, mas utilizando a poderosa estrutura de Injeção de Dependências e *Advisors* do Java Spring Boot.

## 6 Atividade Prática

### 1. Adicionando Dependências

No projeto criado na aula anterior (`spring-ai-demo`), adicione as dependências do **Vector Store** e **Embeddings** ao seu `pom.xml`. Usaremos o SimpleVectorStore (em memória) para simular o banco de forma didática.

```
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-openai-spring-boot-starter</artifactId>
</dependency>
```

### 2. Configuração do Bean de VectorStore

Crie uma classe de configuração `RagConfig.java` para ensinar o Spring a popular o banco de memória durante o *startup* da aplicação:

```
package br.edu.instituicao.springaidemo;

import org.springframework.ai.embedding.EmbeddingModel;
import org.springframework.ai.vectorstore.SimpleVectorStore;
import org.springframework.ai.vectorstore.VectorStore;
import org.springframework.ai.document.Document;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.List;
```

```
@Configuration
public class RagConfig {

    @Bean
    public VectorStore vectorStore(EmbeddingModel embeddingModel) {
        SimpleVectorStore vectorStore = new SimpleVectorStore(embeddingModel);

        // Populando nosso "Banco de Dados" vetorial com um documento interno
        Document contrato = new Document(
            "O benefício de vale-refeição da nossa empresa " +
            "cobre o valor de R$ 50,00 diários para todos os funcionários."
        );
        vectorStore.add(List.of(contrato));

        return vectorStore;
    }
}
```

## 3. O Controlador de RAG com Advisor

No `ChatController`, injete o `VectorStore` e aplique o interceptador automático do Spring AI.

```
package br.edu.instituicao.springaidemo;

import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.chat.client.advisor.QuestionAnswerAdvisor;
import org.springframework.ai.vectorstore.VectorStore;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ChatController {

    private final ChatClient chatClient;

    public ChatController(ChatClient.Builder chatClientBuilder, VectorStore vectorStor
        // O Advisor intercepta o prompt, vai ao VectorStore e injeta o contexto!
        this.chatClient = chatClientBuilder
                .defaultAdvisors(new QuestionAnswerAdvisor(vectorStore))
                .build();
    }
```

```
    @GetMapping("/api/rag")
    public String perguntaEmpresa(@RequestParam String msg) {
        return chatClient.prompt()
                .user(msg)
                .call()
                .content();
    }
}
```

### 4. Execução e Validação

1. Rode o servidor Spring Boot local.
2. No navegador, acesse: `http://localhost:8080/api/rag?msg=Qual é o valor do vale-refeição?`
3. O framework fará a transformação do texto em *embeddings*, buscará no `SimpleVectorStore`, construirá o prompt rígido nos bastidores e entregará a resposta correta e sem alucinações.
4. Tente uma pergunta aleatória fora do contexto e veja-o recusar educadamente.
5. Comite as modificações no Git e propague para o servidor remoto.

## 7 Critério de Avaliação

Aplicação Spring funcional no repositório demonstrando sucesso na implementação do Bean `VectorStore` integrado ao `QuestionAnswerAdvisor` via Injeção de Dependências, operando como uma REST API.

## ✔ Takeaways

- Você construiu o **RAG corporativo completo em Spring AI**: documentos vetorizados + busca semântica + geração fundamentada, tudo via Injeção de Dependências.
- O `QuestionAnswerAdvisor` faz **tudo automaticamente**: vetoriza a pergunta, busca no VectorStore, injeta o contexto no prompt e instrui o LLM.
- A classe `RagConfig` encapsula a configuração do VectorStore como um `@Bean` Spring — substituível e testável.
- O teste de "pergunta fora do domínio" comprova o guardrail: o LLM recusa educadamente quando o contexto não contém a resposta.

### ◇ Informação

**Gancho para a Próxima Aula:** O RAG está funcionando no back-end, mas ele é invisível — uma API que só engenheiros com Postman conseguem usar. Na próxima aula, daremos ao sistema o "**Fator Uau!**": uma interface web HTML+JavaScript que consome a API e apresenta as respostas visualmente — como o ChatGPT.