Laboratório 19: RAG Corporativo no Spring AI

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.

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 vectorSto
        // 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.

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.