Laboratório 18: Olá Mundo com Spring AI

Objetivo do Laboratório

Migraremos nosso escopo do Python para o ecossistema corporativo Java. O objetivo é inicializar um projeto Spring Boot e utilizar o Spring AI para executar um prompt simples, atestando a injeção de dependências e a conectividade com o modelo de nuvem via Java.

Atividade Prática

1. Inicializando o Projeto

1 Acesse `https://start.spring.io/` (Spring Initializr).

2 Configure as opções do Projeto:

- **Project:** Maven
- **Language:** Java (versão 17 ou superior)
- **Spring Boot:** Versão 3.2.x ou superior
- **Group:** `br.edu.instituicao`
- **Artifact:** `spring-ai-demo`

3 Clique em **Add Dependencies** e busque por:

- **Spring Web** (Para construir REST APIs futuramente)
- **OpenAI** (Na seção de AI, ou Gemini, conforme escolha).

4 Gere o projeto (botão GENERATE), extraia o arquivo `.zip` e abra na sua IDE (IntelliJ, Eclipse ou VSCode).

2. Configuração das Credenciais

No diretório `src/main/resources/`, abra o arquivo `application.properties` e adicione as configurações de conexão:

```
# Desabilitando a necessidade de uma chave na compilação do Spring (Fallback)
spring.ai.openai.api-key=${OPENAI_API_KEY}
spring.ai.openai.chat.options.model=gpt-3.5-turbo
```

Nota: Exporte a variável `OPENAI_API_KEY` no seu terminal ou na configuração de Run da sua IDE antes de executar a aplicação.

3. O Código: O Controlador REST

No mesmo pacote da sua classe principal (a que possui `@SpringBootApplication`), crie a classe `ChatController.java`:

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

import org.springframework.ai.chat.client.ChatClient;
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;

    // A mágica da Injeção de Dependência do Spring
    public ChatController(ChatClient.Builder chatClientBuilder) {
        this.chatClient = chatClientBuilder.build();
    }

    @GetMapping("/api/pergunta")
    public String fazerPergunta(@RequestParam(value = "msg", defaultValue = "Me diga u
        return chatClient.prompt()
                .user(msg)
                .call()
                .content();
    }
}
```

4. Execução e Entrega via Git

1 Rode a aplicação (no IntelliJ, execute o método `main` da classe base).

2 Abra o navegador ou o Postman na URL: `http://localhost:8080/api/pergunta?msg=Como a internet funciona`

3 Você deverá receber o texto gerado pela IA no navegador.

4 Copie a pasta do projeto para o seu repositório Git local, realize o rastreamento (`git add .`) assegurando que os arquivos compilados (`target/`) estão no `.gitignore`.

5 Salve e empurre para a nuvem: `git commit -m "Cria esqueleto Spring AI e expõe endpoint generativo"`.

Critério de Avaliação

Presença do código Java válido no GitLab, onde se constate o uso nativo do `ChatClient` para orquestrar a chamada generativa, encapsulado adequadamente num `@RestController`.