Sistema de Cadastro e Consulta de Funcionários — API

Instituição: UFC
Curso: Engenharia da Computação
Disciplina: Sistemas Distribuídos
Professor: Rafael Braga
Aluno: Jeferson Alves e David SIlva

1) Descrição

Este trabalho consiste na implementação do serviço do Trabalho 2, agora usando uma API REST, com comunicação cliente-servidor via requisição/resposta (sem sockets ou RMI). A dupla implementou cada um dos componentes (cliente ou servidor), sendo que o cliente foi feito em ao menos duas linguagens diferentes da usada no servidor.

2) Estrutura do Projeto

SD-TRABALHO-3/

```
├── README.md              ← Documentação geral do projeto
└── sd/                    ← Diretório principal da aplicação
    ├── cliente.py         ← Cliente Python
    ├── ClienteJava.java   ← Cliente Java
    ├── cliente.js         ← Cliente Node.js
    ├── pom.xml            ← Configuração do projeto Maven (Java)
    └── src/
        └── main/
            └── java/
                └── com/
                    └── exemplo/
                        ├── SupermercadoApplication.java
                        ├── HomeController.java
                        ├── FuncionarioController.java
                        ├── FuncionarioService.java
```

├── Funcionario.java

├── Gerente.java

├── Vendedor.java

├── Caixa.java

└── Balconista.java

3) Backend Java (Spring Boot)

Gerenciado com Maven (`pom.xml`) e construído dentro de uma estrutura Spring padrão, inicialmente começamos reaproveitando os códigos anteriores do trabalho, onde pegamos a camada de modelo, entidades principais do sistema, ou seja, os dados que a aplicação manipula. Tendo a classe Base “funcionario.java” e as subclasses de Herança Gerente, Caixa, Vendedor e Balconista. Uma vez já implementadas, o próximo passo foi criar a camada de Serviço, responsável por implementar a lógica de negócio da aplicação. Ou seja, é onde ficam as regras que determinam como os dados devem ser manipulados, validados e tratados antes de serem enviados à camada de controle ou salvos na base de dados (ou estrutura em memória, se não houver banco, como nosso caso).

Foi feita uma nova forma de servidor, visando o manter uma lista interna de funcionários e fornecer dois métodos principais: um para registrar novos funcionários (`registrarFuncionario`) e outro para listar todos os funcionários cadastrados (`listarFuncionarios`). A classe está anotada com `@Service`, permitindo que o Spring a reconheça como um componente de serviço e injete automaticamente onde for necessário.

Por fim, fazemos a camada de Controle, A classe `FuncionarioController` é responsável por expor a API REST para operações relacionadas aos funcionários.

Com a anotação `@RequestMapping("/funcionarios")`, ela define o caminho base da API. A classe injeta a dependência de `FuncionarioService` com `@Autowired`, delegando a ele a lógica de negócio. Ela possui dois endpoints principais: `@PostMapping`: recebe um objeto `Funcionario` via corpo da requisição (`@RequestBody`), registra-o usando o serviço e retorna uma mensagem de confirmação. `@GetMapping`: retorna uma lista com todos os funcionários cadastrados.

Sobre a questão levantada em sala de aula: A aplicação não trabalha com objetos distribuídos, Eles são locais ao servidor onde a aplicação está rodando Atualmente, o sistema desenvolvido segue uma arquitetura monolítica, executando em um único servidor ou container. Os componentes como `FuncionarioController` e `FuncionarioService` existem apenas na memória desse servidor, o que significa que, mesmo com múltiplos clientes acessando o sistema (seja por navegadores ou outras aplicações), todos se conectam ao mesmo ambiente centralizado e compartilham os mesmos objetos Java em memória. Apesar de haver múltiplos acessos simultâneos, isso não caracteriza uma arquitetura distribuída, pois toda a lógica e o estado do sistema estão

concentrados em um único ponto. Mas como estamos trabalhando com um servidor sendo requisitado, então podemos garantir que as demais requisições sejam atribuídas com sucesso.

4) Clientes

Objetivo dos Clientes

Os clientes em Python, Java e JavaScript (Node.js) foram desenvolvidos para permitir a interação remota com a API RESTful implementada no servidor Spring Boot. Esses clientes ilustram o conceito de serviço distribuído, onde a lógica da aplicação é dividida entre cliente e servidor por meio de requisições HTTP.

Serviço Remoto REST

O servidor Spring Boot expõe um serviço web no endpoint:

http://localhost:8080/funcionarios

Este serviço segue o padrão REST e permite:

- GET /funcionarios → Listar todos os funcionários
- POST /funcionarios → Cadastrar um novo funcionário

A comunicação é feita utilizando o protocolo HTTP e dados no formato **JSON**, o que garante portabilidade entre diferentes linguagens e plataformas.

Clientes Implementados

1. Cliente em Python (cliente.py)

- Biblioteca usada: requests

- Funções principais:
  Envia requisição GET para obter a lista de funcionários
  Envia POST com dados JSON para cadastrar um novo funcionário

- Interação via terminal (CLI)

- Exemplo de uso:

```
res = requests.get('http://localhost:8080/funcionarios')
res = requests.post('http://localhost:8080/funcionarios', json=funcionario)
```

Demonstra simplicidade e rapidez de prototipação com Python.

2. Cliente em Java (ClienteJava.java)

- Biblioteca usada: HttpClient (Java 17+)
- Funções principais:
  Requisição GET com HttpRequest e tratamento da resposta
  Envio de JSON via POST para cadastrar funcionário
- Entrada por Scanner, menu interativo
- Exemplo de uso:

```
HttpRequest request = HttpRequest.newBuilder()
    .uri(URI.create(URL))
    .GET()
    .build();
```

Demonstra integração robusta e segura com serviço REST usando Java moderno.

3. Cliente em JavaScript (cliente.js)

- Biblioteca usada: node-fetch (v2)
- Funções principais:
  Utiliza fetch() para realizar GET e POST
  Interface de texto via readline
- Exemplo de uso:

```
const res = await fetch(URL, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(funcionario)
});
```

Demonstra a leveza da interação com APIs REST usando JavaScript do lado do cliente.