UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

PROGRAMA DE GRADUAÇÃO EM REDES DE COMPUTADORES

JOÃO FERREIRA LIMA NETO

VINICIUS GABRIEL RODRIGUES DO NASCIMENTO

RELATÓRIO DA ÚLTIMA ENTREGA DO TRABALHO FINAL DE SISTEMAS DISTRIBUÍDOS COM O TEMA DE “LOJA DE PEÇAS DE VEÍCULOS”

QUIXADÁ

Relatório do Projeto: Sistema de Gerenciamento de Peças com API REST

1. Introdução

Este projeto consiste em uma aplicação Java que permite o gerenciamento de peças através de uma API REST. A aplicação é composta por dois módulos principais: um cliente (PecaClientApi) que interage com o servidor para adicionar e consultar peças, e um servidor (PecaServerApi) que gerencia as peças e fornece endpoints para operações CRUD (Create, Read, Update, Delete).

2. Tecnologias Utilizadas

Java 17: Linguagem de programação utilizada para desenvolver a aplicação.

Spring Boot 3.2.0: Framework para desenvolvimento de aplicações Java, facilitando a criação de APIs REST.

Gradle: Ferramenta de automação de build utilizada para gerenciar dependências e compilar o projeto.

HTTP Client (Java 11+): Utilizado no cliente para fazer requisições HTTP para o servidor.

JSON: Formato de dados utilizado para troca de informações entre cliente e servidor.

3.1. Cliente (PecaClientApi.java)

O cliente é responsável por interagir com o usuário e enviar requisições para o servidor. Ele oferece um menu interativo com as seguintes opções:

Adicionar Peça: O usuário insere o nome, código e quantidade da peça, que são enviados para o servidor via POST.

Listar Peças: O cliente faz uma requisição GET para obter a lista de todas as peças cadastradas.

Atualizar Peça: O usuário insere o código da peça e os novos dados (nome e quantidade), que são enviados para o servidor via PUT.

Deletar Peça: O usuário insere o código da peça, que é enviado para o servidor via DELETE.

Sair: Encerra a execução do cliente.

O cliente utiliza a classe HttpClient para enviar requisições HTTP e processar as respostas do servidor.

3.2. Servidor (PecaServerApi.java)

O servidor é uma aplicação Spring Boot que expõe endpoints REST para gerenciar peças. Ele possui os seguintes endpoints:

GET /pecas: Retorna a lista de todas as peças cadastradas.

POST /pecas: Adiciona uma nova peça à lista.

GET /pecas/{codigo}: Retorna uma peça específica com base no código.

PUT /pecas/{codigo}: Atualiza uma peça existente com base no código.

DELETE /pecas/{codigo}: Remove uma peça com base no código.

GET /pecas/quantidade-total: Retorna a quantidade total de peças cadastradas.

O servidor utiliza uma lista em memória para armazenar as peças e sincroniza o acesso a essa lista para evitar problemas de concorrência.

3.3. Modelo de Dados (Peca.java)

A classe Peca representa o modelo de dados utilizado para armazenar informações sobre as peças. Ela contém os seguintes atributos:

nome: Nome da peça.

codigo: Código único da peça.

quantidade: Quantidade disponível da peça.

A classe também inclui métodos getters e setters para acessar e modificar esses atributos.

3.4. Configuração do Projeto (build.gradle)

O arquivo build.gradle configura as dependências do projeto e define a classe principal para execução. As principais dependências incluem:

spring-boot-starter-web: Para criação de APIs REST.

spring-boot-starter-thymeleaf: Para renderização de views (não utilizado diretamente neste projeto).

spring-boot-starter-test: Para testes.

3.5. Configuração de Propriedades (application.properties)

O arquivo application.properties contém configurações para desabilitar a configuração automática do DataSource e do JPA, já que o projeto não utiliza banco de dados.

4. Funcionamento do Projeto

4.1. Execução do Servidor

Para executar o servidor, basta rodar a classe PecaServerApi como uma aplicação Spring Boot. O servidor estará disponível em http://localhost:8080.

4.2. Execução do Cliente
O cliente pode ser executado diretamente a partir da classe PecaClientApi. Ele interage com o usuário via console, permitindo a adição de peças e a consulta de informações.

4.3. Fluxo de Operações
Adicionar Peça: O usuário insere os dados da peça (nome, código e quantidade), que são enviados para o servidor via POST.

Listar Peças: O cliente faz uma requisição GET para obter a lista de todas as peças cadastradas.

Atualizar Peça: O usuário insere o código da peça e os novos dados (nome e quantidade), que são enviados para o servidor via PUT.

Deletar Peça: O usuário insere o código da peça, que é enviado para o servidor via DELETE.

Consultar Quantidade Total: O cliente faz uma requisição GET para obter a quantidade total de peças cadastradas.

5. Como o projeto utiliza uma API REST?
Neste projeto, o servidor (PecaServerApi) expõe endpoints RESTful que seguem os princípios de uma API REST. Esses endpoints permitem que o cliente (PecaClientApi) interaja com o servidor para realizar operações sobre o recurso "peças". Vamos analisar os principais elementos que caracterizam o uso de uma **API REST**:

a) Endpoints RESTful
O servidor define endpoints que seguem as convenções REST:

GET /pecas: Retorna a lista de todas as peças (operação de leitura).

POST /pecas: Adiciona uma nova peça (operação de criação).

PUT /pecas/{codigo}: Atualiza uma peça existente com base no código (operação de atualização).

DELETE /pecas/{codigo}: Remove uma peça com base no código (operação de exclusão).

GET /pecas/{codigo}: Retorna uma peça específica com base no código (operação de leitura).

GET /pecas/quantidade-total: Retorna a quantidade total de peças (operação de leitura).

Esses endpoints são acessados via métodos HTTP, que é uma característica fundamental de uma **API REST**.

b) Uso de Métodos HTTP

O cliente faz requisições HTTP para interagir com o servidor:

POST: Para enviar dados de uma nova peça.

GET: Para recuperar informações sobre as peças.
PUT: Para atualizar uma peça existente.

DELETE: Para remover uma peça.

Isso está alinhado com as práticas **REST**, onde cada método *HTTP* tem um significado específico:

POST para criação.

GET para leitura.

PUT para atualização.

DELETE para exclusão.

c) Formato de Dados (JSON)

O cliente e o servidor trocam dados no formato JSON, que é amplamente utilizado em APIs REST. Por exemplo:

O cliente envia um JSON no corpo da requisição POST:

```
{
  "nome": "Parafuso",
  "codigo": "123",
  "quantidade": 10
}
```

O servidor responde com uma lista de peças ou a quantidade total em formato JSON.

d) Stateless

A API segue o princípio stateless do REST, ou seja, cada requisição contém todas as informações necessárias para o servidor processá-la. O servidor não mantém estado entre requisições.

e) Sincronização de Recursos Compartilhados

O servidor utiliza sincronização (synchronized) para garantir que operações concorrentes (como adicionar, atualizar ou remover peças) não causem inconsistências na lista de peças. Isso é especialmente importante em ambientes onde múltiplos clientes podem acessar o servidor simultaneamente.

f) Respostas HTTP Descritivas

O servidor retorna respostas HTTP descritivas para indicar o resultado das operações:

200 OK: Para operações bem-sucedidas (como adicionar, atualizar ou listar peças).

404 Not Found: Quando uma peça não é encontrada (por exemplo, ao tentar atualizar ou deletar uma peça com um código inexistente).

6. Conclusão

Este projeto demonstra a criação de uma API REST simples utilizando Spring Boot e a interação com essa API através de um cliente Java. Ele pode ser expandido para incluir mais funcionalidades, como atualização e remoção de peças, além de integração com um banco de dados para persistência dos dados.

O código dos arquivos se encontra no GitHub, e o vídeo explicativo se encontra no Moodle.