



UFC

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.

Thymeleaf: Template engine para renderização de views (não utilizado diretamente neste projeto, mas configurado no build.gradle).

3. Estrutura do Projeto

3.1. Cliente (PecaClientApi.java)

O cliente é responsável por interagir com o usuário e enviar requisições para o servidor. Ele permite:

Adicionar peças: O usuário pode inserir o nome, código e quantidade de peças, que são enviadas para o servidor via POST.

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

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

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.

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

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.

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.

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).

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.

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

POST para criação.

GET para leitura.

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.

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.