Para todo eso utilice el proyecto de MERCADO VILLA , traendo productos de ropa  
  
  
Controlador de manipulación de datos en los end point

package **com.constructora.maquinarias\_pesada.controller**;  
  
  
import **com.constructora.maquinarias\_pesada.dto.ProductoDTO**;  
import **com.constructora.maquinarias\_pesada.dto.ProductoResponse**;  
import **com.constructora.maquinarias\_pesada.service.ProductoService**;  
import **org.springframework.beans.factory.annotation.**Autowired;  
import **org.springframework.http.ResponseEntity**;  
import **org.springframework.web.bind.annotation.**GetMapping;  
import **org.springframework.web.bind.annotation.**RequestMapping;  
import **org.springframework.web.bind.annotation.**RestController;  
  
@RestController  
@RequestMapping("/producto")  
public class **ProductoController** {  
  
 @Autowired  
 public **ProductoService** productoService;  
  
  
 @GetMapping(produces = "application/json")  
 public **ResponseEntity**<**ProductoResponse**> allEmpleado(){  
 try{  
 return productoService.productoDatosGson();  
 }catch (**Exception** e){  
 return **ResponseEntity**.*internalServerError*().build();  
 }  
 }  
  
  
}

serviceImpl

package **com.constructora.maquinarias\_pesada.service.serviceImpl**;  
  
import **com.constructora.maquinarias\_pesada.dto.Producto**;  
import **com.constructora.maquinarias\_pesada.dto.ProductoDTO**;  
import **com.constructora.maquinarias\_pesada.dto.ProductoResponse**;  
import **com.constructora.maquinarias\_pesada.dto.RawProductoResponse**;  
import **com.constructora.maquinarias\_pesada.service.ProductoService**;  
import **org.springframework.http.HttpStatus**;  
import **org.springframework.http.ResponseEntity**;  
import **org.springframework.stereotype.**Service;  
import **org.springframework.web.client.RestTemplate**;  
  
  
import **com.fasterxml.jackson.databind.JsonNode**;  
import **com.fasterxml.jackson.databind.ObjectMapper**;  
  
import **java.util.ArrayList**;  
import **java.util.Collections**;  
import **java.util.List**;  
import **java.util.Map**;  
  
@Service  
public class **ProductoServiceImpl** implements **ProductoService** {  
  
  
 private final **RestTemplate** restTemplate;  
  
 public ProductoServiceImpl(**RestTemplate** \_restTemplate){  
 this.restTemplate = \_restTemplate;  
 }  
  
  
  
 @Override  
 public **ResponseEntity**<**ProductoResponse**> productoDatosGson() {  
 // ✅ Endpoint a consumir  
 **String** url = "http://localhost:5147/api/Prenda";  
  
 // Obtenemos la respuesta inicial  
 **RawProductoResponse** body = restTemplate.getForObject(url, **RawProductoResponse**.class);  
 **List**<**ProductoDTO**> productos = new ArrayList<>();  
  
 if (body == null) {  
 **ProductoResponse** response = new ProductoResponse(  
 **String**.*valueOf*(**HttpStatus**.*NOT\_FOUND*.value()),  
 null,  
 **Collections**.<**ProductoDTO**>*emptyList*()  
 );  
 return **ResponseEntity**.*status*(**HttpStatus**.*NOT\_FOUND*).body(response);  
 }  
  
 try {  
 **ObjectMapper** objectMapper = new ObjectMapper();  
 **JsonNode** jsonNode = objectMapper.valueToTree(body);  
  
 // ✅ Campos principales  
 **String** estado = jsonNode.path("estado").asText("error");  
 **String** fecha = jsonNode.path("fecha").asText("");  
  
 // ✅ Procesamos el arreglo anidado de productos  
 **JsonNode** productoArray = jsonNode.path("producto");  
 if (productoArray.isArray()) {  
 for (**JsonNode** listaInterna : productoArray) {  
 for (**JsonNode** nodo : listaInterna) {  
 **String** nombre = nodo.path("nombre").asText();  
 **String** marca = nodo.path("marca").asText();  
 int cantidad = nodo.path("cantidad").asInt();  
 double precio = nodo.path("precio").asDouble();  
  
 **String** cantidadDescripcion = stockDisponible(cantidad);  
 productos.add(new ProductoDTO(nombre, marca, cantidadDescripcion, precio));  
 }  
 }  
 }  
  
 // ✅ Construimos la respuesta final  
 **ProductoResponse** response = new ProductoResponse(estado, fecha, productos);  
 return **ResponseEntity**.*ok*(response);  
  
 } catch (**Exception** e) {  
 // En caso de error inesperado al parsear  
 **ProductoResponse** errorResponse = new ProductoResponse(  
 "error",  
 null,  
 **Collections**.*emptyList*()  
 );  
 return **ResponseEntity**.*status*(**HttpStatus**.*INTERNAL\_SERVER\_ERROR*).body(errorResponse);  
 }  
 }  
  
 // 🔹 Método auxiliar: descripción del stock  
 public **String** stockDisponible(int cantidad) {  
 if (cantidad < 10) return "quedan pocos";  
 if (cantidad <= 30) return "estás a tiempo";  
 return "stock disponible";  
 }

}

Proyecto principal :  
package **com.constructora.maquinarias\_pesada**;  
  
import **org.springframework.boot.SpringApplication**;  
import **org.springframework.boot.autoconfigure.**SpringBootApplication;  
import **org.springframework.cloud.openfeign.**EnableFeignClients;  
import **org.springframework.context.annotation.**Bean;  
import **org.springframework.web.client.RestTemplate**;  
  
@EnableFeignClients  
@SpringBootApplication  
public class **MaquinariasPesadaApplication** {  
  
 public static void main(**String**[] args) {  
 **SpringApplication**.*run*(**MaquinariasPesadaApplication**.class, args);  
 }  
  
  
 @Bean  
 public **RestTemplate** restTemplate() {  
 return new RestTemplate();  
 }  
}

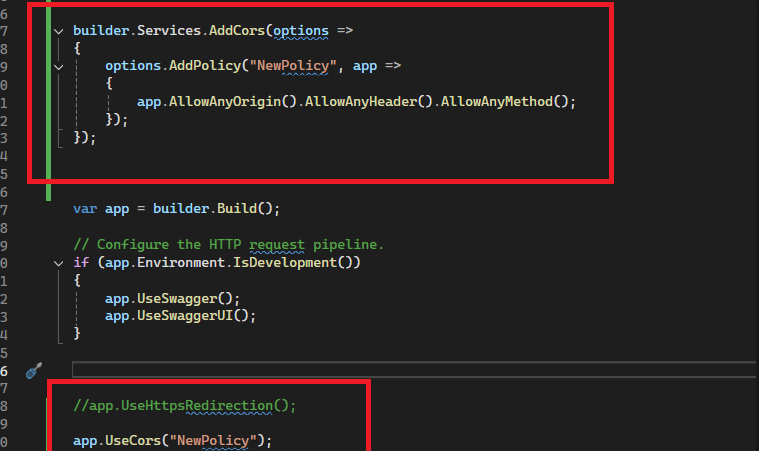
Application.properties:  
spring.application.name=maquinarias-pesada  
server.port=8088  
  
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration  
  
  
  
  
  
  
#Esto desactiva por completo la autoconfiguración de la base de datos.  
#Con esto, tu aplicación inicia correctamente aunque no tengas MySQL, PostgreSQL, etc  
  
  
#si SÍ quieres usar base de datos (más adelante)  
#  
#Asegúrate de tener en pom.xml las dependencias correctas.  
#Por ejemplo, si usas MySQL:  
  
  
#spring.datasource.url=jdbc:mysql://localhost:3306/tu\_base  
#spring.datasource.username=root  
#spring.datasource.password=  
#spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
#  
#spring.jpa.hibernate.ddl-auto=update  
#spring.jpa.show-sql=true

En los modelos, se tuvo que usar los geter y los seters

Como fue un proyecto el cual consumía apis de otro proyecto con apis desarrollado en .NET , se tenia que avilitar unos permisos desde .NET … en .net se utiliza https/ - MIENTRAS UQE EN SPRING HTTP/ … muy aparte que en launchSettings(en la carpeta, dentro de .net) el endPoint tiene que considir con el proyecto.http(carpetas)   
  
.NET en Program.cs TIENE QUE ESTAr desactivado

//app.UseHttpsRedirection();

Tan bien activamos el cors en .NET 🡪



En el tema de las dependencias , dentro de spring “pom.xml” encontramos unos cuantos errores , como:  
la versión fue una errónea que trataba de 2025 (aun cuenta con errores) entonces le puse esta (mas estable)

<version>3.3.4</version>

Tan bien me falto unas dependencias o otras que estaban de mas   
obvio no use conexión a BD ya que era consumo de tora apis, por ende desidi no instalar esa dependencia ya que era innecesaria.

Tan bien tuve otro problema en

<spring-cloud.version>2023.0.3</spring-cloud.version>

Por ende copíe esa línea , ya que anteriormente había una que no servia del todo

Dependencias necesarias

Spring Web  
✅ Lombok  
✅ Spring Boot DevTools (opcional)  
✅ Spring Data JPA (si usarás BD)  
✅ MySQL Driver (si usarás MySQL)  
✅ Spring Cloud OpenFeign (si vas a consumir APIs externas)

Captura de pantalla de computadora

Descripción generada automáticamente