CONCEPTOS

DE

MICROSERVICIOS

Y EJEMPLO.

Autor: Ildefonso Albares García
Fecha Actualización 29-07-2024

Índice

1. Partes de un Microservicio

Un microservicio lo vamos a definir como una caja a la que le llega una petición de forma de URL, normalmente, y con esa información el microservicio le devuelve la información solicitada. Para ello se definen varios paquetes en el proyecto separado

1.1. Modelo de datos o entidad

En esta parte del código se debe definir los datos que serán los representativos dentro de la base de datos (BBDD) a la que acceda el microservicio.

Este archivo o archivos no sólo definen el modelo de acceso a la Base de datos, también pueden contener otro tipo de programación como definir un mensaje, lo veremos en un ejemplo más adelante, pero se define como la entidad.

1.2. Capa de acceso a datos

La capa de acceso a los datos es el archivo encargado de hacer toda la lógica de forma abstraída para devolvernos los datos solicitados de la BBDD. Esto se consigue con las dependencias JPA y de esta manera evitamos el realizar de forma manual la conexión a la base de datos. Lo rápido es usar la interfaz JpaRepository de Spring Data JPA.

La diferencia es que JpaRepository es una extensión específica del Repositorio **JPA (Java Persistence API)** . Contiene la API completa de **CrudRepository** y **PagingAndSortingRepository** . Por lo tanto, contiene API para operaciones CRUD básicas y también API para paginación y clasificación. Dependiendo de lo que necesitemos implementamos una u otra.

1.3. Capa lógica

En esta parte se implementa toda la lógica posible para reducir lo máximo el código en la parte del controlador.

Aquí se realizan definiciones de métodos, validaciones o cualquier otro cálculo complejo. Esta parte se implementa con dos archivos, uno es una interfaz y el segundo es un tipo “Class” identificado con @Service.

En la interfaz del servicio definimos que métodos tendrá el servicio, y en el propio archivo del servicio, implementamos los métodos anteriores.

1.4. El controlador

Aquí se definen las llamadas de la URL´s y lo que debe devolver en dichas solicitudes. Aquí se integra básicamente la lógica del controlador del @RestController. En este caso hay que definir que hace cada solicitud (GET, PUT, POST y DELETE)

2. Nomenclatura y anotaciones del Microservicio Springboot.

- @Controller: que registrará el controlador para Spring MVC
- @RequestMapping: anotación que se encarga de relacionar un método con una petición http
- @RequestBody: contiene la información del cuerpo (normalmente en JSON)
- @SpringBootApplication: indica donde empieza el código.
- @RestController: indica que es el controlador.

- @Autowired: le indica a Spring que busque un bean (un objeto gestionado por Spring) de un tipo específico y lo inyecte en el lugar donde se encuentra la anotación.
- @GetMapping("/request"): para realizar una consulta a base de datos
- @PostMapping("/request"): para realizar un insert en la base de datos.
- @PutMapping: para modificar la base de datos.
- @DeleteMapping("/request"): para realizar operaciones de borrado en la base de datos
- @PathVariable: indica que hay una variable en el Url que se envía.
- @Entity: indica que es la entidad que corresponde con la tabla en base de datos.
- @Table (name="marcas"): da el nombre a la tabla.
- @Id: indica que es el identificador.
- @GeneratedValue(strategy = GenerationType.IDENTITY): indica la forma que se genera el identificador único.
- @Service: indica que es el servicio del micro.

3. Ejemplo de Microservicio

Archivo de inicio o arranque del micro.

IProductoServicio.java | ProductoServicioImpl.java | SprintbootServicioSaludoApplication.java

```java
package com.example.producto.saludo;

import org.springframework.boot.SpringApplication;

@EnableEurekaClient
@SpringBootApplication
public class SprintbootServicioSaludoApplication {

    public static void main(String[] args) {
        SpringApplication.run(SprintbootServicioSaludoApplication.class, args);
    }

}

```

3.2. Capa de acceso a datos

Archivo correspondiente al gestor del repositorio para las consultas.

```java
package com.example.producto.saludo.dao;

import org.springframework.data.repository.CrudRepository;

public interface ProductoDao extends CrudRepository<ProductoMarca, String>{

}
```

3.1. Modelo de datos o entidad.

Archivo que representa la base de datos y debe ser igual a la tabla en BBDD.

```java
package com.example.producto.saludo.mensaje;

import java.io.Serializable;

@Entity
@Table (name="marcas")
public class ProductoMarca implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nombre;
    private Integer precio;
    private String articulo;

    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public Integer getPrecio() {
        return precio;
    }
    public void setPrecio(Integer precio) {
        this.precio = precio;
    }
    public String getArticulo() {
        return articulo;
    }
    public void setArticulo(String articulo) {
        this.articulo = articulo;
    }

    private static final long serialVersionUID = 704674473724120359L;

}

```

3.3. Capa lógica

Archivo que contiene la mayor parte de la lógica.

```java
package com.example.producto.saludo.servicio;

import java.util.List;



@Service
public class ProductoServicioImpl implements IProductoServicio{

    @Autowired
    private ProductoDao productodao;

    @Override
    public ProductoMensaje mensajeback() {
        ProductoMensaje mensaje=new ProductoMensaje();
        mensaje.setMensaje("Hola Mundo");

        return mensaje;
    }

    @Override
    @Transactional(readOnly = true)
    public List<ProductoMarca> findAll() {
        return (List<ProductoMarca>) productodao.findAll();
    }

}

```

Archivo para la interfaz donde se deben definir todos los métodos que contenga el archivo del Servicio.

```java
package com.example.producto.saludo.servicio;

import java.util.List;

public interface IProductoServicio {

    public ProductoMensaje mensajeback();
    public List<ProductoMarca> findAll();
}

```

3.4. El controlador

Archivo que contiene las peticiones Rest.

```java
package com.example.producto.saludo.controller;

import java.util.List;

@RestController
public class SaludoController {

    @Autowired
    private IProductoServicio productoService;

    @GetMapping("/saludos")
    public ProductoMensaje mostrarmensaje() {
        return productoService.mensajeback();
      }

    @GetMapping("/listar")
    public List<ProductoMarca> seleccion(){
        return productoService.findAll();
    }

}
```

4. Bibliografía y otros.

- Curso Microservicios Sprint Boot, SpringCloud, Netflix Eureka 2024. Udemy Andrés Guzman.
- Roldán, D.,Valderas P.J. & Torres V. Microservicios. Un enfoque integrado.
- https://sinbugs.com/como-crear-un-microservicio-o-servicio-web-rest-con-spring-boot-1/
- https://ifgeekthen.nttdata.com/s/post/introduccion-a-spring-boot-creacion-de-un-microservicio-MCPYCV7SLWINDK5POHKL463LJPTI?language=es