Bar JDK Services Project

A continuacion la documentacion del codigo de los microservicios que fueron generados en Java Spring Boot, para los servicios se manejan 5 carpetas base, entity, repository, service, implement y controller, a continuacion explicaremos uno por uno:

Entity

Las clases de entidad son las clases aquellas que suelen usarse para mapearse directamente a tablas en la base de datos. A continuacion un ejemplo de un entity:

```
package org.barjdk.entity;

import com.fasterxml.jackson.annotation.JsonIgnore;
import jakarta.persistence.*;
import lombok.Data;

// La anotación @Data es parte del proyecto Lombok y genera automáticamente métodos como equals(), hashCode(), toString(), etc.
@Data
// La anotación @Entity indica que esta clase es una entidad JPA.
@Entity
// La anotación @Table especifica el nombre de la tabla en la base de datos a la que se asignará esta entidad.
@Table(name = "EMPLEADO")
public class EmpleadoEntity {

    // La anotación @Id marca el campo como clave primaria de la entidad.
    @Id
    // La anotación @GeneratedValue indica la estrategia de generación de valores para la clave primaria.
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    // La anotación @Column especifica el nombre de la columna en la base de datos.
    @Column(name = "PK_EMPLEADO_ID")
    private Integer pkEmpleadoId;

    @Column(name = "DOCUMENTO")
    private String documento;

    @Column(name = "NOMBRE")
    private String nombre;

    @Column(name = "APELLIDO")
    private String apellido;

    @Column(name = "USUARIO_ACCESO")
    private String usuarioAcceso;

    @Column(name = "CLAVE_ACCESO")
    private String claveAcceso;
```

```
    // La anotación @OneToOne indica una relación uno a uno con otra entidad.
    @OneToOne
    // La anotación @JoinColumn especifica la columna de la clave externa en esta
entidad.
    @JoinColumn(name = "FK_ROL_ID")
    private RolEntity rol;

    @OneToOne
    @JoinColumn(name = "FK_SEDE_ID")
    private SedeEntity sede;
}
```

Repository

En Spring, una clase repository se utiliza para acceder y gestionar datos en una base de datos. La anotación @Repository se usa para indicar que la clase es un bean de repositorio, lo que significa que maneja las operaciones de persistencia, como la lectura y escritura de datos en una base de datos.

Las clases repository en Spring Data se utilizan comúnmente con JPA (Java Persistence API) para interactuar con bases de datos relacionales. Estas clases ofrecen métodos convenientes para realizar operaciones CRUD (Create, Read, Update, Delete) en las entidades de dominio, encapsulando la lógica de acceso a la base de datos. A continuacion un ejemplo de un repository:

```
package org.barjdk.repository;

import org.barjdk.entity.EmpleadoEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

// La anotación @Repository indica que esta interfaz es un componente de repositorio
de Spring y maneja el acceso a la base de datos.
@Repository("EmpleadoRepository")
// La interfaz extiende JpaRepository, que proporciona métodos CRUD predefinidos
para la entidad EmpleadoEntity y su clave primaria (Integer).
public interface EmpleadoRepository extends JpaRepository<EmpleadoEntity, Integer> {
    // No se necesitan métodos adicionales aquí porque JpaRepository ya incluye
métodos como save(), findById(), delete(), etc.
    // Spring Data JPA generará automáticamente las consultas SQL correspondientes
en tiempo de ejecución.
}
```

Services

Esta interfaz define las operaciones básicas que pueden llevarse a cabo con entidades relacionadas con las entidades de la aplicación. Cada método representa una operación específica. Implementar esta interfaz en una clase concreta permitirá realizar estas operaciones utilizando la lógica de negocio real de la aplicación. A continuacion un ejemplo de un service:

```java
package org.barjdk.services;

import org.barjdk.entity.EmpleadoEntity;
import org.barjdk.entity.PermisosEmpleadoEntity;

import java.util.List;

// Interfaz que define los métodos para operaciones relacionadas con la entidad
EmpleadoEntity
public interface EmpleadoService {

    // Método para consultar un empleado por su identificador (pkEmpleadoId)
    EmpleadoEntity consultarPorId(Integer pkEmpleadoId);

    // Método para consultar todos los empleados en la base de datos
    List<EmpleadoEntity> consultarTodos();

    // Método para guardar un empleado en la base de datos
    EmpleadoEntity guardar(EmpleadoEntity empleado);

    // Método para eliminar un empleado de la base de datos
    void eliminar(EmpleadoEntity empleado);

    // Método para eliminar un empleado por su identificador (pkEmpleadoId)
    void eliminarPorId(Integer pkEmpleadoId);

    // Método para validar el acceso de un empleado por usuario y contraseña
    PermisosEmpleadoEntity validarAcceso(String usuarioAcceso, String claveAcceso);
}
```

Implement

En un proyecto Spring, la carpeta implement es utilizada para almacenar las implementaciones concretas de interfaces definidas en el paquete service. Este patrón es comúnmente conocido como "Service Implementation Pattern" y se utiliza para separar claramente la interfaz (contrato) de la implementación real. A continuacion un ejemplo de un implement:

```java
package org.barjdk.implement;

import lombok.extern.slf4j.Slf4j;
import org.barjdk.entity.EmpleadoEntity;
import org.barjdk.entity.PermisosEmpleadoEntity;
import org.barjdk.repository.EmpleadoRepository;
import org.barjdk.repository.PermisosEmpleadoRepository;
import org.barjdk.services.EmpleadoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.security.MessageDigest;
```

```java
import java.security.NoSuchAlgorithmException;
import java.util.List;

// La anotación @Service indica que esta clase es un componente de servicio gestionado por Spring.
@Service
// La anotación @Slf4j es parte de Lombok y proporciona un logger llamado 'log'.
@Slf4j
public class EmpleadoImplement implements EmpleadoService {

    // La anotación @Autowired realiza la inyección de dependencias de EmpleadoRepository.
    @Autowired
    private EmpleadoRepository empleadoRepository;

    // La anotación @Autowired realiza la inyección de dependencias de PermisosEmpleadoRepository.
    @Autowired
    private PermisosEmpleadoRepository permisosEmpleadoRepository;

    // Implementación del método de la interfaz para consultar un empleado por su ID.
    @Override
    public EmpleadoEntity consultarPorId(Integer pkEmpleadoId) {
        return empleadoRepository.findById(pkEmpleadoId).orElse(null);
    }

    // Implementación del método de la interfaz para consultar todos los empleados.
    @Override
    public List<EmpleadoEntity> consultarTodos() {
        return this.empleadoRepository.findAll();
    }

    // Implementación del método de la interfaz para guardar un empleado.
    @Override
    public EmpleadoEntity guardar(EmpleadoEntity empleado) {
        // Encriptar la claveAcceso con SHA-256 antes de guardarla en la base de datos.
        String claveAccesoEncriptada = encriptarConSHA256(empleado.getClaveAcceso());
        empleado.setClaveAcceso(claveAccesoEncriptada);

        // Guardar el empleado en la base de datos y devolver la entidad guardada.
        return empleadoRepository.save(empleado);
    }

    // Método privado para encriptar un texto con el algoritmo SHA-256.
    private String encriptarConSHA256(String texto) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] hashBytes = md.digest(texto.getBytes());
```

```
            // Convertir el array de bytes a una representación hexadecimal.
            StringBuilder hexStringBuilder = new StringBuilder();
            for (byte b : hashBytes) {
                hexStringBuilder.append(String.format("%02x", b));
            }

            return hexStringBuilder.toString();
        } catch (NoSuchAlgorithmException e) {
            // Manejar la excepción apropiadamente.
            e.printStackTrace();
            throw new RuntimeException("Error al encriptar la contraseña");
        }
    }

    // Implementación del método de la interfaz para eliminar un empleado.
    @Override
    public void eliminar(EmpleadoEntity empleado) {
        empleadoRepository.delete(empleado);
    }

    // Implementación del método de la interfaz para eliminar un empleado por su ID.
    @Override
    public void eliminarPorId(Integer pkEmpleadoId) {
        empleadoRepository.deleteById(pkEmpleadoId);
    }

    // Implementación del método de la interfaz para validar el acceso de un
empleado.
    @Override
    public PermisosEmpleadoEntity validarAcceso(String usuarioAcceso, String
claveAcceso) {
        // Validar la entrada.
        if (usuarioAcceso == null || claveAcceso == null) {
            throw new IllegalArgumentException("Usuario de acceso o clave de acceso
nulos");
        }

        // Encriptar la claveAcceso con SHA-256 antes de buscar en la base de datos.
        String claveAccesoEncriptada = encriptarConSHA256(claveAcceso);

        // Buscar en la base de datos y devolver el resultado.
        return
permisosEmpleadoRepository.findByUsuarioAccesoAndClaveAcceso(usuarioAcceso,
claveAccesoEncriptada);
    }
}
```

Controller

En Spring, un controlador (Controller) es una clase encargada de manejar las solicitudes HTTP y coordinar la lógica de presentación o negocio de una aplicación. Los controladores reciben solicitudes del cliente, procesan la entrada,

interactúan con los servicios subyacentes (normalmente a través de inyección de dependencias), y devuelven la respuesta apropiada al cliente. A continuacion un ejemplo de un controller:

```
package org.barjdk.controllers;

import org.barjdk.entity.EmpleadoEntity;
import org.barjdk.entity.PermisosEmpleadoEntity;
import org.barjdk.services.EmpleadoService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/empleado")
@CrossOrigin(origins = "*")
public class EmpleadoController {

    // Inyección de dependencias del servicio EmpleadoService.
    @Autowired
    EmpleadoService empleadoService;

    // Configuración de un logger para registrar información en el sistema.
    private final Logger log = LoggerFactory.getLogger(EmpleadoController.class);

    // Método para manejar solicitudes GET para consultar un empleado por su ID.
    @GetMapping(path = "/consultar/{id}")
    public EmpleadoEntity consultarPorId(@PathVariable(name = "id") Integer
pkEmpleadoId) {
        return empleadoService.consultarPorId(pkEmpleadoId);
    }

    // Método para manejar solicitudes GET para consultar todos los empleados.
    @GetMapping(path = "/consultarTodos")
    public List<EmpleadoEntity> consultarTodos() {
        return this.empleadoService.consultarTodos();
    }

    // Método para manejar solicitudes POST y PUT para guardar un empleado.
    @RequestMapping(path = "/guardar", method = { RequestMethod.POST,
RequestMethod.PUT })
    public EmpleadoEntity guardar(@RequestBody EmpleadoEntity empleadoEntity) {
        // Registrar información sobre el empleado en el sistema.
        log.info(String.format("Empleado: %s", empleadoEntity.toString()));
        // Llamar al servicio para guardar el empleado.
        return empleadoService.guardar(empleadoEntity);
    }

    // Método para manejar solicitudes DELETE para eliminar un empleado por objeto.
```

```
    @DeleteMapping(path = "/eliminar")
    public void eliminar(@RequestBody EmpleadoEntity empleado) {
        empleadoService.eliminar(empleado);
    }

    // Método para manejar solicitudes DELETE para eliminar un empleado por su ID.
    @DeleteMapping("/eliminar/{id}")
    public void eliminarPorId(@PathVariable(name = "id") Integer pkEmpleadoId) {
        empleadoService.eliminarPorId(pkEmpleadoId);
    }

    // Método para manejar solicitudes POST para validar el acceso de un empleado.
    @PostMapping(path = "/validar")
    public PermisosEmpleadoEntity validar(@RequestBody EmpleadoEntity empleado) {
        // Llamar al servicio para validar el acceso y devolver el resultado.
        return empleadoService.validarAcceso(empleado.getUsuarioAcceso(),
empleado.getClaveAcceso());
    }
}
```

Estas son las 5 carpetas principales que se utilizan en un proyecto SpringBoot para lograr el correcto funcionamiento e implementacion de microservicios.