DOCUMENTACIÓN

Santiago Celis – 202111131

Silvana Sandoval – 202123682

Gabriela Soler – 202123744

Para la documentación del proyecto vamos a tomar como ejemplo una sola entidad, debido a que el procedimiento es similar para las demás. En este caso, la entidad seleccionada es Hotel:

Definición de la entidad en el Modelo:

```java
package uniandes.edu.co.proyecto.Modelo;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.OneToOne;
import jakarta.persistence.Table;
```

Para cada entidad se usaron los *imports* necesarios para asegurar la persistencia de los datos.

```java
@Entity
@Table(name="hoteles")

public class Hotel {
    @Id
    private String nombre;
    private String tipo;
    private String ciudad;
    private String pais;

    @OneToOne
    @JoinColumn(name = "internets_id", referencedColumnName = "id")
    private Internet internets_id;
```

Se definen las variables para cada atributo. A su vez, se establecen las etiquetas necesarias para indicar que atributos son llaves primarias y también, para definir la cardinalidad de las llaves foráneas con su respectivo atributo y tabla de origen.

```java
public Hotel(String nombre, String tipo, String ciudad, String pais, Internet internets_id)
{
    this.nombre = nombre;
    this.tipo = tipo;
    this.ciudad = ciudad;
    this.pais = pais;
    this.internets_id = internets_id;
}
```

Para cada entidad, el constructor incluye no solos los atributos propios, sino que también se reciben como parámetros los atributos que representan llaves foráneas. Al final, cada una se asigna a las variables declaradas anteriormente.

```java
public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getTipo() {
    return tipo;
}
```

Por último, se definen los *getters* y *setters* para la obtención y asignación de valores en las variables.

Definición del Repositorio:

```java
package uniandes.edu.co.proyecto.Repositorios;

import java.util.Collection;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.transaction.annotation.Transactional;

import uniandes.edu.co.proyecto.Modelo.Hotel;
```

Se importan las librerías necesarias para el desarrollo del repositorio y su correcto funcionamiento para cumplir con los requerimientos funcionales.

```java
public interface HotelRepository extends JpaRepository<Hotel, String> {
    @Query(value = "SELECT * FROM hoteles", nativeQuery=true)
    Collection<Hotel> darHoteles();

    @Query(value = "SELECT * FROM hoteles WHERE nombre = :nombre", nativeQuery=true)
    Hotel darHotel(@Param("nombre") String nombre);

    @Modifying
    @Transactional
    @Query(value = "INSERT INTO hoteles (nombre, tipo, ciudad, pais, internet) VALUES(hoteles_sequence.nextval, :nombre, :t
    void insertarHotel(@Param("nombre") String nombre, @Param("tipo") String tipo, @Param("ciudad") String ciudad, @Param("

    @Modifying
    @Transactional
    @Query(value = "UPDATE hoteles SET  tipo = :tipo, ciudad = :ciudad, pais = :pais, internet = : internet WHERE id=:id",
    void actualizarHotel(@Param("tipo") String tipo, @Param("ciudad") String ciudad, @Param("pais") String pais, @Param("in

    @Modifying
    @Transactional
    @Query(value = "DELETE FROM hoteles WHERE id = :id", nativeQuery=true)
    void eliminarHotel(@Param("nombre") String nombre);
}
```

Se definen las consultas necesarias, denotadas con la etiqueta *@Query*, para para extraer la información necesaria para las consultas. Estas incluyen: una invocación a la tabla (SELECT *), uno para insertar valores (INSERT, denotado con las etiquetas *@Modifiy* que indica la modificación de la tabla y *@Transactional* denotando su carácter de transacción en la tabla), uno para modificar dichos valores (UPDATE, denotado con las etiquetas *@Modifiy* que indica la modificación de la tabla y *@Transactional* denotando su carácter de transacción en la tabla) y uno para eliminar los valores insertados (DELETE, denotado con las etiquetas *@Modifiy* que indica la modificación de la tabla y *@Transactional* denotando su carácter de transacción en la tabla). Lo anterior, resolver todos los requerimientos funcionales pedidos para todas las entidades del problema. Cada uno tiene los parámetros necesarios para las consultas.

Update: tiene todos los atributos, menos la llave primaria para modificar.

Delete: solo tiene la llave primaria la cual es la necesaria para eliminar el objeto requerido.

Insert: tiene los atributos a insertar para crear un objeto.

Dar: sirve para conseguir los objetos necesarios

Definición del Controlador:

```java
package uniandes.edu.co.proyecto.Controller;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;

import uniandes.edu.co.proyecto.Modelo.Hotel;
import uniandes.edu.co.proyecto.Repositorios.HotelRepository;
```

En primer lugar, se definieron los *imports* necesarios para el correcto funcionamiento de los datos que se utilizarían para cada entidad, con el fin de garantizar la persistencia de la información.

```java
@Controller
public class HotelesController {
    @Autowired
    private HotelRepository hotelRepository;
```

Después, se establece la clase como un controlador con el fin de gestionar las solicitudes HTTP y controlar la lógica de negocio asociada. *Autowired* se usa para inyectar las dependencias en Spring, es decir, los repositorios de cada clase.

```java
@GetMapping("/hoteles/new")
public String hotelForm(Model model) {
    model.addAttribute("hotel", new Hotel());
    return "hotelNuevo";
}

@PostMapping("/hoteles/new/save")
public String hotelGuardar(@ModelAttribute Hotel hotel) {
    hotelRepository.insertarHotel(hotel.getNombre(), hotel.getTipo(), hotel.getCiudad(), hotel.getPais(), hotel.getInternets_id().getId());
    return "redirect:/hoteles";
}

@GetMapping("/hoteles/{id}/edit")
public String hotelEditarForm(@PathVariable("nombre") String nombre, Model model) {
    Hotel hotel = hotelRepository.darHotel(nombre);
    if (hotel != null) {
        model.addAttribute("hotel", hotel);
        return "hotelEditar";
    } else {
        return "redirect:/hoteles";
    }
}
```

Este código implementa las operaciones CRUD (crear, leer, actualizar, eliminar) para cada entidad. @GetMappin maneja las solicitudes GET a la URL asociada, @PostMapping se utiliza para manejar las solicitudes POST a la URL, es decir, para procesar los datos enviados.

```java
@PostMapping("/hoteles/{id}/edit/save")
public String hotelEditarGuardar(@PathVariable("nombre") String nombre, @ModelAttribute Hotel hotel) {
    hotelRepository.actualizarHotel(hotel.getTipo(), hotel.getCiudad(), hotel.getPais(), hotel.getInternets_id().getId());
    return "redirect:/hoteles";
}

@GetMapping("/hoteles/{id}/delete")
public String hotelEliminar(@PathVariable("nombre") String nombre) {
    hotelRepository.eliminarHotel(nombre);
    return "redirect:/hoteles";
}
```

Finalmente, se utilizan las anotaciones @PostMapping y @GetMappin para procesar los datos enviados desde un formulario y actualizar la entidad correspondiente en el repositorio y para identificar el identificador o la llave de la entidad que se va a eliminar respectivamente.