**Construcción de aplicaciones web en JSP.**

**Competencia:** Determinar especificaciones y requerimientos de la aplicación web.

Una clase contiene atributos y métodos.

* Los atributos definen el objeto.

Cada clase tiene una instancia y lo que representa un objeto único.

**Palabras:**

* **Bufferedreader:** para capturar datos a través del teclado como Scanner.
* **Integer.parseInt():** es para convertir una línea de código o lo que se recibe como a un carácter legible y intero.
* **Double.parseDouble():** similar a Integer.parseInt(); con la diferencia que este es para decimales.

**DESARROLLO DE APLICACIONES.**

**Esquema de comunicación y flujo**

**//pendiente**

**<https://drive.google.com/file/d/1Dx74ywd6_l8191UTr_keSbDF2kBVFb_L/view?usp=sharing>**

1. **Crear y configurar proyecto con spring boot.**
2. **Ir a la página Spring Initializr <https://start.spring.io/>**
3. **Seleccionar las opciones del Project, Language y Spring boot.**

* **Maven es más robusto.**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**

1. **Configurar:**
   * **Group:** es el paquete base. Se debe colocar el dominio del negocio; no es recomendable dejar example.
   * **Artifact:** es el nombre de la aplicación.

**Tabla

El contenido generado por IA puede ser incorrecto.**

1. **Y agregar las dependencias siguientes. Dar clic en “GENERATE”**

* **Spring web:** trae las librerías que corren la API.

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

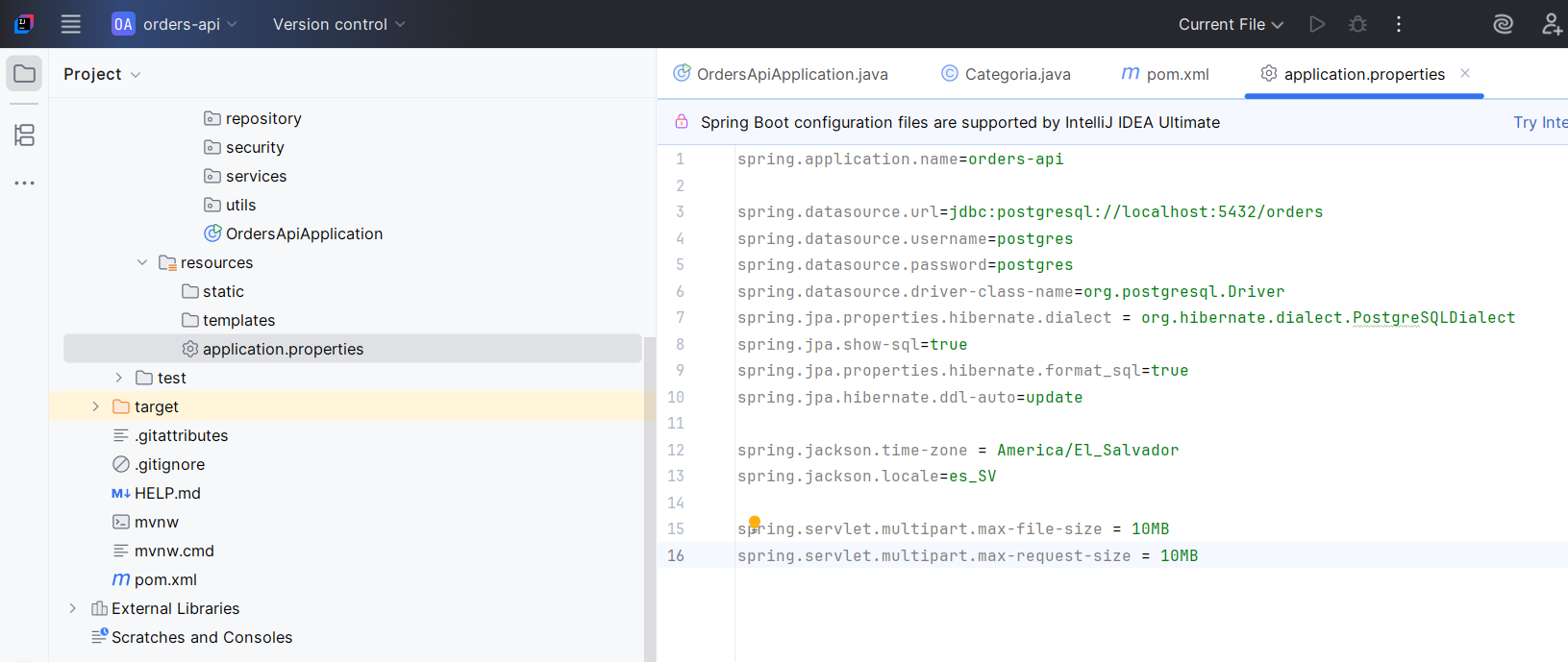
El contenido generado por IA puede ser incorrecto.**

1. **Crear una carpeta para guardar el archivo descomprimido.**
2. **Se crearon los Packages en el package principal:** com.devsoft.orders\_api

**Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.**

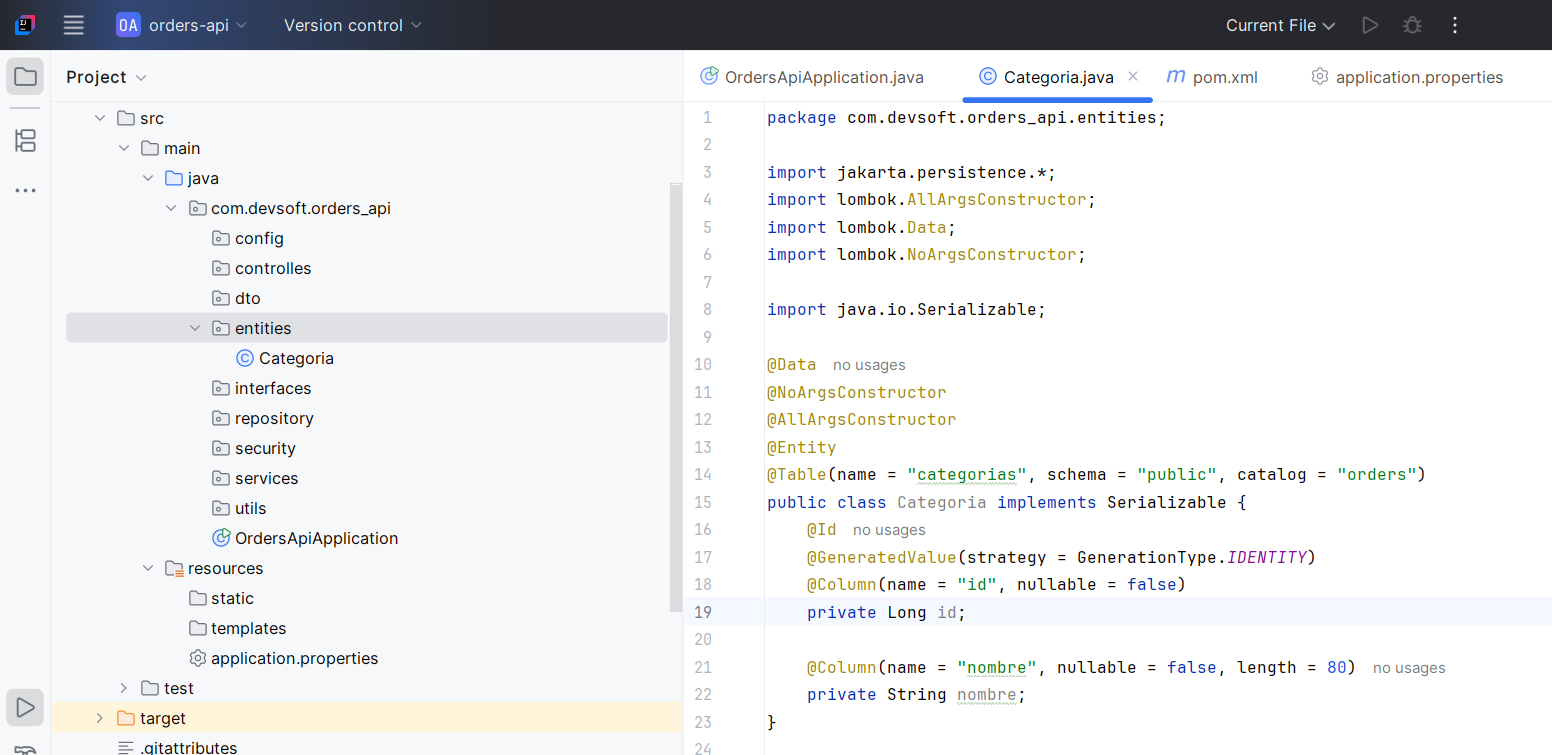
1. **Configurar el archivo “application.properties”.**

****

1. **Se debe cambiar el nombre de la base de datos “/orders”**

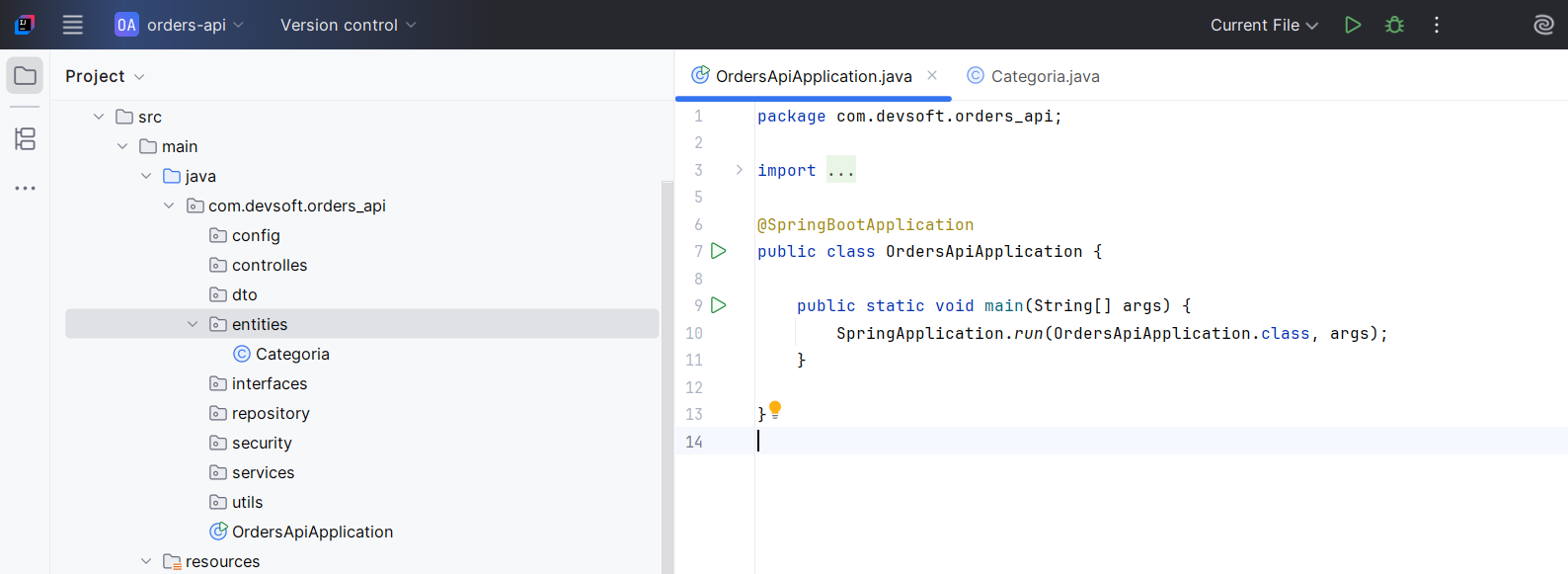
spring.application.name=orders-api  
  
spring.datasource.url=jdbc:postgresql://localhost:5432/orders  
spring.datasource.username=postgres  
spring.datasource.password=postgres  
spring.datasource.driver-class-name=org.postgresql.Driver  
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.format\_sql=true  
spring.jpa.hibernate.ddl-auto=update  
  
spring.jackson.time-zone = America/El\_Salvador  
spring.jackson.locale=es\_SV  
  
spring.servlet.multipart.max-file-size = 10MB  
spring.servlet.multipart.max-request-size = 10MB

1. ***Crear una clase (Java Class) en este caso se llamó Categoria en la carpeta entities (package):*** ESTE CÓDIGO HA SIDO MODIFICADO EN EL PASO 13

****

package com.devsoft.orders\_api.entities;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.io.Serializable;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "categorias", schema = "public", catalog = "orders")  
public class Categoria implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 @Column(name = "id", nullable = false)  
 private Long id;  
  
 @Column(name = "nombre", nullable = false, length = 80)  
 private String nombre;  
}

1. **PARA EJECUTAR EL PROGRAMA SE DEBE UBICAR EN EL ARCHIVO (nombreDelProyecto)…ApiApplication.java**

****

1. **Verificar que esté instalado “Lombok”**:

**Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.**

1. **SE HIZO MODIFICACIÓN EN LA CLASE “Categoria.java”**

**@Setter**

**@Getter**

**@NoArgsConstructor**

**@AllArgsConstructor**

**@Entity**

**Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.**

package com.devsoft.orders\_api.entities;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.io.Serial;  
import java.io.Serializable;  
  
@Setter  
@Getter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "categorias", schema = "public", catalog = "orders")  
public class Categoria implements Serializable {  
 @Serial  
 private static final long *serialVersionUID* = 1L;  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "nombre", nullable = false, length = 80)  
 private String nombre;  
 @Column(name = "descripcion", nullable = false, length = 100)  
 private String descripcion;  
}

1. **Se creó la clase “Menu.java”**

package com.devsoft.orders\_api.entities;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.io.Serial;  
import java.io.Serializable;  
import java.math.BigDecimal;  
  
@Setter  
@Getter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "menus", schema = "public", catalog = "orders")  
public class Menu implements Serializable {  
 @Serial  
 private static final long *serialVersionUID* = 1L;  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "nombre", nullable = false, length = 80)  
 private String nombre;  
 @Column(name = "descripcion", nullable = false, length = 100)  
 private String descripcion;  
 @Column(name = "tipo", nullable = false, length = 1)  
 private String tipo; *//Producto o Platillo* @Column(name = "precio\_unitario", nullable = false, precision = 8, scale = 2)  
 private BigDecimal precioUnitario;  
 @Column(name = "url\_imagen", nullable = true, length = 100)  
 private String urlImagen;  
 @Column(name = "activo", columnDefinition = "boolean default true")  
 private boolean activo;  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "categoria\_id", referencedColumnName = "id", nullable = false)  
 private Categoria categoria;  
}

1. **Se creó la clase “Mesa.java”**

package com.devsoft.orders\_api.entities;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.io.Serial;  
import java.io.Serializable;  
  
@Setter  
@Getter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "mesas", schema = "public", catalog = "orders")  
public class Mesa implements Serializable {  
 @Serial  
 private static final long *serialVersionUID* = 1L;  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "numero", nullable = false)  
 private int numero;  
 @Column(name = "ubicacion", nullable = false, length = 15)  
 private String ubicacion;  
}

1. **Se creó la clase “Role.java”**
2. package com.devsoft.orders\_api.entities;  
     
   import jakarta.persistence.\*;  
   import lombok.AllArgsConstructor;  
   import lombok.Getter;  
   import lombok.NoArgsConstructor;  
   import lombok.Setter;  
     
   import java.io.Serial;  
   import java.io.Serializable;  
     
   @Setter  
   @Getter  
   @NoArgsConstructor  
   @AllArgsConstructor  
   @Entity  
   @Table(name = "roles", schema = "public", catalog = "orders")  
   public class Role implements Serializable {  
    @Serial  
    private static final long *serialVersionUID* = 1L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
    private Long id;  
    @Column(name = "nombre", nullable = false, length = 50)  
    private String nombre;  
   }
3. **Se creó la clase “Usuario.java”**

package com.devsoft.orders\_api.entities;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.io.Serial;  
import java.io.Serializable;  
  
@Setter  
@Getter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "usuarios", schema = "public", catalog = "orders")  
public class Usuario implements Serializable {  
 @Serial  
 private static final long *serialVersionUID* = 1L;  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "nombre", nullable = false, length = 80)  
 private String nombre;  
 @Column(name = "usuario", nullable = false, length = 30)  
 private String usuario;  
 @Column(name = "password", nullable = false, length = 100)  
 private String password;  
 @Column(name = "activo", columnDefinition = "boolean default true")  
 private boolean activo;  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "role\_id", referencedColumnName = "id", nullable = false)  
 private Role role;  
}

1. **Se creó la clase “Cliente.java”**

package com.devsoft.orders\_api.entities;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.io.Serial;  
import java.io.Serializable;  
  
@Setter  
@Getter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "clientes", schema = "public", catalog = "orders")  
public class Cliente implements Serializable {  
 @Serial  
 private static final long *serialVersionUID* = 1L;  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)private Long id;  
 @Column(name = "nombre", nullable = false, length = 50)  
 private String nombre;  
 @Column(name = "direccion", nullable = false, length = 150)  
 private String direccion;  
 @Column(name = "telefono", nullable = true, length = 12)  
 private String telefono;  
 @Column(name = "email", nullable = true, length = 50)  
 private String email;  
 @Column(name = "tipo\_cliente", nullable = false, length = 1)  
 private String tipoCliente;  
}

1. **Se creó la clase “Orden.java”**

package com.devsoft.orders\_api.entities;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.io.Serializable;  
import java.math.BigDecimal;  
import java.time.LocalDate;  
import java.time.LocalTime;  
  
@Setter  
@Getter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "ordenes", schema = "public", catalog = "orders")  
public class Orden implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 @Column(name = "correlativo", nullable = false, length = 10)  
 private String correlativo;  
 @Column(name = "fecha", nullable = false)  
 private LocalDate fecha;  
 @Column(name = "hora", nullable = false)  
 private LocalTime hora;  
 @Column(name = "estado", nullable = false, length = 1)  
 private String estado;  
 @Column(name = "total", nullable = false, precision = 10, scale = 2)  
 private BigDecimal total;  
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "cliente\_id", referencedColumnName = "id", nullable = false)  
 private Cliente cliente;  
 private Mesa mesa;  
 private Usuario usuario;  
  
 *//relación con detalleOrden*}

1. **Se creó la clase “DetalleOrden.java”**

package com.devsoft.orders\_api.entities;  
  
import com.fasterxml.jackson.annotation.JsonBackReference;  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.io.Serial;  
import java.io.Serializable;  
import java.math.BigDecimal;  
  
@Setter  
@Getter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "detalle\_ordenes", schema = "public", catalog = "orders")  
public class DetalleOrden implements Serializable {  
 @Serial  
 private static final long *serialVersionUID* = 1L;  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "cantidad", nullable = false)  
 private int cantidad;  
 @Column(name = "precio", nullable = false, precision = 8, scale = 2)  
 private BigDecimal precio;  
 @Column(name = "subtotal", nullable = false, precision = 10, scale = 2)  
 private BigDecimal subtotal;  
  
 *//relaciones con Orden y Menu* @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "menu\_id", referencedColumnName = "id", nullable = false)  
 private Menu menu;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "orden\_id", referencedColumnName = "id", nullable = false)  
 @JsonBackReference *//para evitar cargar la consulta de la orden* private Orden orden;  
}

1. **Modificaciones en Orden.java**

package com.devsoft.orders\_api.entities;  
  
import com.fasterxml.jackson.annotation.JsonManagedReference;  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.io.Serial;  
import java.io.Serializable;  
import java.math.BigDecimal;  
import java.time.LocalDate;  
import java.time.LocalTime;  
import java.util.List;  
  
@Setter  
@Getter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "ordenes", schema = "public", catalog = "orders")  
public class Orden implements Serializable {  
 @Serial  
 private static final long *serialVersionUID* = 1L;  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "correlativo", nullable = false, length = 10)  
 private String correlativo;  
 @Column(name = "fecha", nullable = false)  
 private LocalDate fecha;  
 @Column(name = "hora", nullable = false)  
 private LocalTime hora;  
 @Column(name = "estado", nullable = false, length = 1)  
 private String estado;  
 @Column(name = "total", nullable = false, precision = 10, scale = 2)  
 private BigDecimal total;  
  
 *//relaciones con Cliente, Mesa y Usuario* @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "cliente\_id", referencedColumnName = "id", nullable = false)  
 private Cliente cliente;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "mesa\_id", referencedColumnName = "id", nullable = false)  
 private Mesa mesa;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "usuario\_id", referencedColumnName = "id", nullable = false)  
 private Usuario usuario;  
  
 *//relacion OneToMany con DetalleOrden* @OneToMany(mappedBy = "orden", cascade = CascadeType.*ALL*, orphanRemoval = true, fetch = FetchType.*LAZY*)  
 @JsonManagedReference *//para evitar la recursividad infinita al serializar* private List<DetalleOrden> detalleOrden;  
}

1. **Modificaciones en DetalleOrden.java**

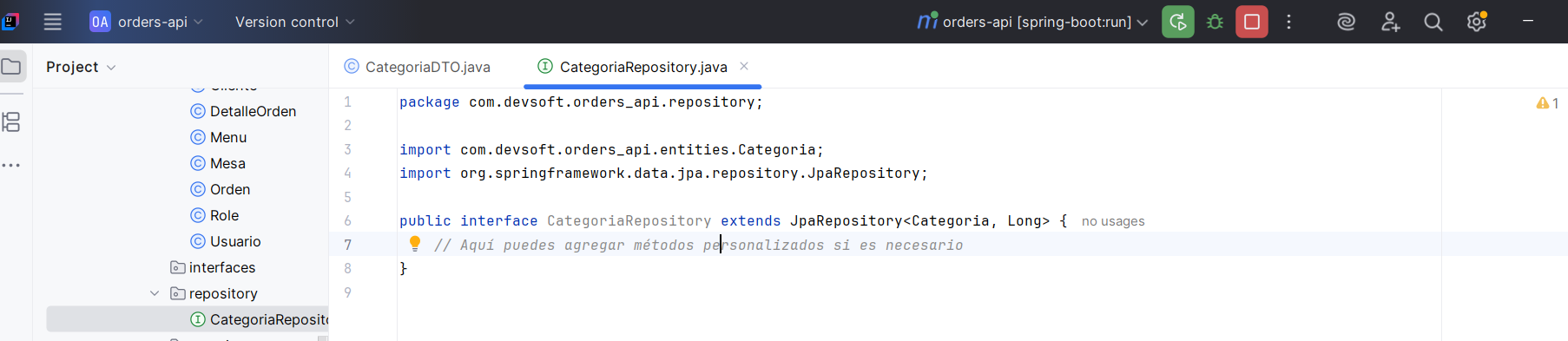
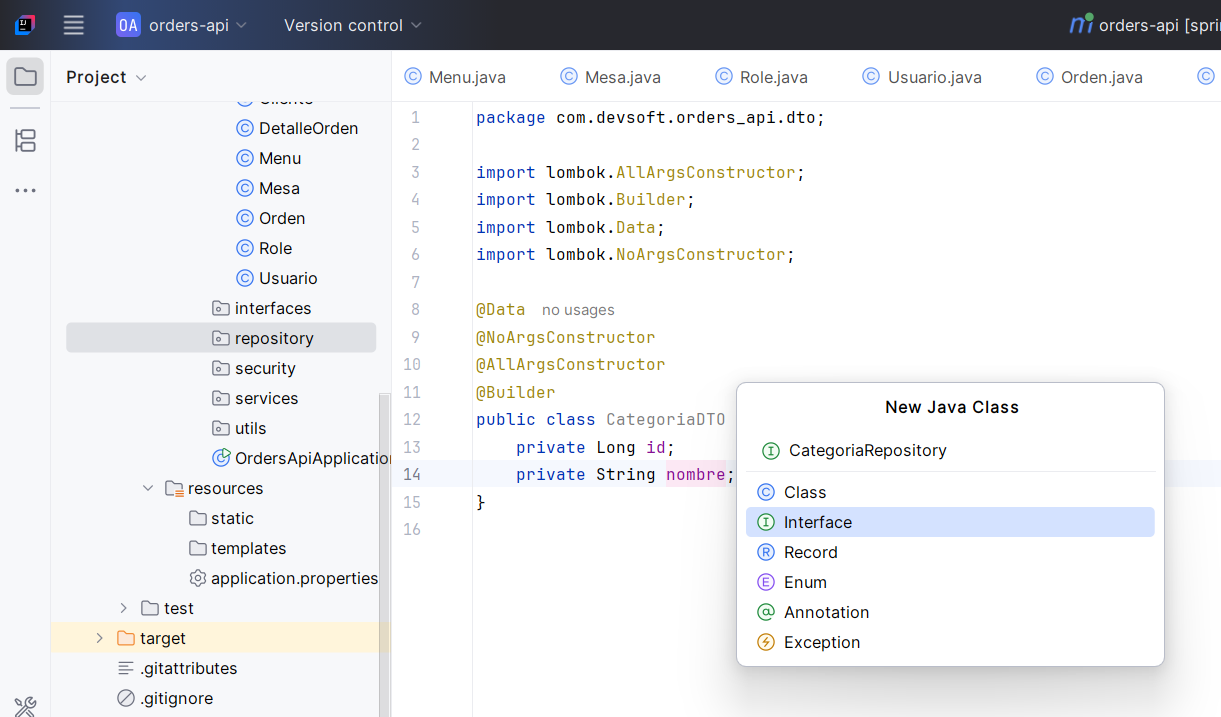
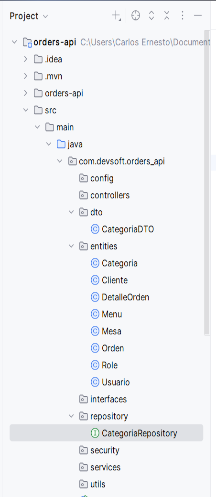
package com.devsoft.orders\_api.entities;  
  
import com.fasterxml.jackson.annotation.JsonBackReference;  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.io.Serial;  
import java.io.Serializable;  
import java.math.BigDecimal;  
  
@Setter  
@Getter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "detalle\_ordenes", schema = "public", catalog = "orders")  
public class DetalleOrden implements Serializable {  
 @Serial  
 private static final long *serialVersionUID* = 1L;  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "cantidad", nullable = false)  
 private int cantidad;  
 @Column(name = "precio", nullable = false, precision = 8, scale = 2)  
 private BigDecimal precio;  
 @Column(name = "subtotal", nullable = false, precision = 10, scale = 2)  
 private BigDecimal subtotal;  
  
 *//relaciones con Orden y Menu* @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "menu\_id", referencedColumnName = "id", nullable = false)  
 private Menu menu;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "orden\_id", referencedColumnName = "id", nullable = false)  
 @JsonBackReference *//para evitar cargar la consulta de la orden* private Orden orden;  
}

1. **En carpeta “DTO” se creó el archivo “CategoriaDTO”**

package com.devsoft.orders\_api.dto;  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class CategoriaDTO {  
 private Long id;  
 private String nombre;  
}

**Se crea Una Interface en package “Repository”**

package com.devsoft.orders\_api.repository;  
  
import com.devsoft.orders\_api.entities.Categoria;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
import java.util.Optional;  
  
@Repository  
public interface CategoriaRepository extends JpaRepository<Categoria, Long> {  
  
 *//Este método servirá para no insertar registros duplicados* Optional<Categoria> findByNombreIgnoreCase(String nombre);  
}



1. **Se crea una interface en package Interfaces.**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

package com.devsoft.orders\_api.interfaces;  
  
import com.devsoft.orders\_api.dto.CategoriaDTO;  
  
import java.util.List;  
  
public interface ICategoriaService {  
 List<CategoriaDTO> findAll();  
 CategoriaDTO findById(Long id);  
 CategoriaDTO save(CategoriaDTO categoriaDTO);  
 void delete(Long id);  
}

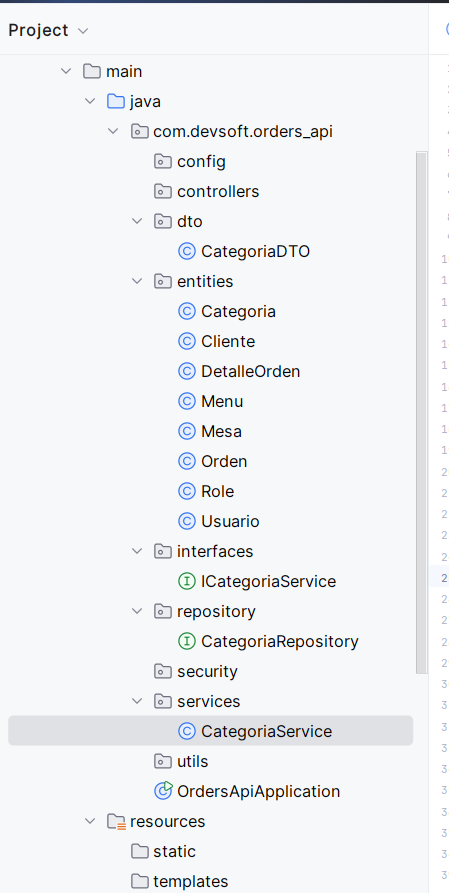
1. **Se crea una clase CategoriaService en package Se rvices.**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

package com.devsoft.orders\_api.services;  
  
import com.devsoft.orders\_api.dto.CategoriaDTO;  
import com.devsoft.orders\_api.entities.Categoria;  
import com.devsoft.orders\_api.interfaces.ICategoriaService;  
import com.devsoft.orders\_api.repository.CategoriaRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import java.util.Collections;  
import java.util.List;  
import java.util.stream.Collectors;  
  
@Service  
public class CategoriaService implements ICategoriaService { *//importar los métodos de la interfaz* @Autowired *//Inyectar por dependencia el repositorio* private CategoriaRepository categoriaRepository;  
  
 @Override  
 public List<CategoriaDTO> findAll() {  
 return categoriaRepository.findAll().stream()  
 .map(this::convertToDTO)  
 .collect(Collectors.*toList*());  
 }  
  
 @Override  
 public CategoriaDTO findById(Long id) {  
 return null;  
 }  
  
 @Override  
 public CategoriaDTO save(CategoriaDTO categoriaDTO) {  
 return null;  
 }  
  
 @Override  
 public void delete(Long id) {  
  
 }

*// Método privado para convertir una entidad Categoria a CategoriaDTO* private CategoriaDTO convertToDTO(Categoria categoria) {  
 return new CategoriaDTO(categoria.getId(),categoria.getNombre());  
 }  
}

****

1. **Se creó la clase CategoriaDTO.java en el package dto.**

package com.devsoft.orders\_api.dto;  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
  
*// Esta clase es un DTO (Data Transfer Object) que representa una categoría  
// y permite transferir datos entra el controlador y el servicio*public class CategoriaDTO {  
 private Long id;  
 private String nombre;  
}

1. **Modificaciones en CategoryRepository.java**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**

1. **Modificaciones en ICategoryService.java**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**

1. **Modificaciones en CategoriaService.java**

package com.devsoft.orders\_api.services;  
  
import com.devsoft.orders\_api.dto.CategoriaDTO;  
import com.devsoft.orders\_api.entities.Categoria;  
import com.devsoft.orders\_api.interfaces.ICategoriaService;  
import com.devsoft.orders\_api.repository.CategoriaRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.NoSuchElementException;  
import java.util.stream.Collectors;  
  
@Service  
public class CategoriaService implements ICategoriaService { *//importar los métodos de la interfaz* @Autowired *//Inyectar por dependencia el repositorio* private CategoriaRepository categoriaRepository;  
  
 @Override  
 public List<CategoriaDTO> findAll() {  
 return categoriaRepository.findAll().stream()  
 .map(this::convertToDTO)  
 .collect(Collectors.*toList*());  
 }  
  
 *// Método para obtener todas las categorías sin convertir a DTO* public List<Categoria> getAll(){  
 return categoriaRepository.findAll();  
 }  
  
 @Override  
 public CategoriaDTO findById(Long id) {  
 Categoria categoria = categoriaRepository.findById(id)  
 .orElseThrow(() -> new NoSuchElementException("No se encuentra una categoria con el ID: " + id));  
 return convertToDTO(categoria);  
 }  
  
 @Override  
 public CategoriaDTO save(CategoriaDTO dto) {  
 Categoria catNueva = new Categoria();  
 if(dto.getId() == null) {  
 catNueva.setNombre(dto.getNombre());  
 }else{  
 catNueva.setId(dto.getId());  
 catNueva.setNombre(dto.getNombre());  
 }  
 return convertToDTO(categoriaRepository.save(catNueva));  
 }  
  
 *// Este método busca una categoría por su nombre y devuelve un CategoriaDTO* @Override  
 public CategoriaDTO findByNombre(String nombre) {  
 return categoriaRepository.findByNombre(nombre);  
 }  
  
 @Override  
 public void delete(Long id) {  
 categoriaRepository.deleteById(id);  
 }  
  
 *// Método privado para convertir una entidad Categoria a CategoriaDTO* private CategoriaDTO convertToDTO(Categoria categoria) {  
 return new CategoriaDTO(categoria.getId(),categoria.getNombre());  
 }  
}

1. **Se creó CategoriaController.java en package controllers.**

**Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.**

package com.devsoft.orders\_api.controllers;  
  
import com.devsoft.orders\_api.dto.CategoriaDTO;  
import com.devsoft.orders\_api.interfaces.ICategoriaService;  
import org.apache.coyote.Response;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.dao.DataAccessException;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
@RestController  
@CrossOrigin  
@RequestMapping("/api")  
public class CategoriaController {  
 @Autowired  
 private ICategoriaService categoriaService;  
  
 @GetMapping("/categorias")  
 public ResponseEntity<?> getAll() {  
 List<CategoriaDTO> categoriaDTOList = categoriaService.findAll();  
 return ResponseEntity.*ok*(categoriaDTOList);  
 }  
 @PostMapping("/categorias")  
 public ResponseEntity<?> save(@RequestBody CategoriaDTO dto){  
 CategoriaDTO catPersisted = new CategoriaDTO();  
 Map<String, Object> response = new HashMap<>();  
 try{  
 CategoriaDTO catExiste = categoriaService.findByNombre(dto.getNombre());  
 if (catExiste != null && dto.getId() == null) {  
 response.put("message", "Ya existe una categoria con este nombre, digite otro");  
 return new ResponseEntity<Map<String,Object>>(response, HttpStatus.*CONFLICT*);  
 }  
 catPersisted = categoriaService.save(dto);  
 response.put("message", "Categoria guardada correctamente");  
 response.put("categoria", catPersisted);  
 return new ResponseEntity<Map<String,Object>>(response, HttpStatus.*CREATED*);  
 }catch (DataAccessException e) {  
 response.put("message", "Error al insertar el registro, intente nuevamente");  
 response.put("error", e.getMessage());  
 return new ResponseEntity<Map<String,Object>>(response, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
}

1. **Para pruebas en Postman se comentó la validación en try{}**

package com.devsoft.orders\_api.controllers;  
  
import com.devsoft.orders\_api.dto.CategoriaDTO;  
import com.devsoft.orders\_api.interfaces.ICategoriaService;  
import org.apache.coyote.Response;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.dao.DataAccessException;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
@RestController  
@CrossOrigin  
@RequestMapping("/api")  
public class CategoriaController {  
 @Autowired  
 private ICategoriaService categoriaService;  
  
 @GetMapping("/categorias")  
 public ResponseEntity<?> getAll() {  
 List<CategoriaDTO> categoriaDTOList = categoriaService.findAll();  
 return ResponseEntity.*ok*(categoriaDTOList);  
 }  
 @PostMapping("/categorias")  
 public ResponseEntity<?> save(@RequestBody CategoriaDTO dto){  
 CategoriaDTO catPersisted = new CategoriaDTO();  
 Map<String, Object> response = new HashMap<>();  
 try{  
 */\*CategoriaDTO catExiste = categoriaService.findByNombre(dto.getNombre());  
 if (catExiste != null && dto.getId() == null) {  
 response.put("message", "Ya existe una categoria con este nombre, digite otro");  
 return new ResponseEntity<Map<String,Object>>(response, HttpStatus.CONFLICT);  
 }\*/* catPersisted = categoriaService.save(dto);  
 response.put("message", "Categoria guardada correctamente");  
 response.put("categoria", catPersisted);  
 return new ResponseEntity<Map<String,Object>>(response, HttpStatus.*CREATED*);  
 }catch (DataAccessException e) {  
 response.put("message", "Error al insertar el registro, intente nuevamente");  
 response.put("error", e.getMessage());  
 return new ResponseEntity<Map<String,Object>>(response, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
}

1. **En pgAdmin, en la base de datos orders, se debe modificar el NOT NULL (desactivarlo) en la tabla Categorias, y de igual forma en la clase Categoria.java.**

**NOT NULL desactivado:**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**

**En el código de la clase Categoria.java, en el @Column de “descripción” cambiar el “nullable = false” a “nullable = true”:**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**

**Al realizar lo anterior (solo si es absolutamente necesario), se debe realizar una petición GET y POST en Postman.**

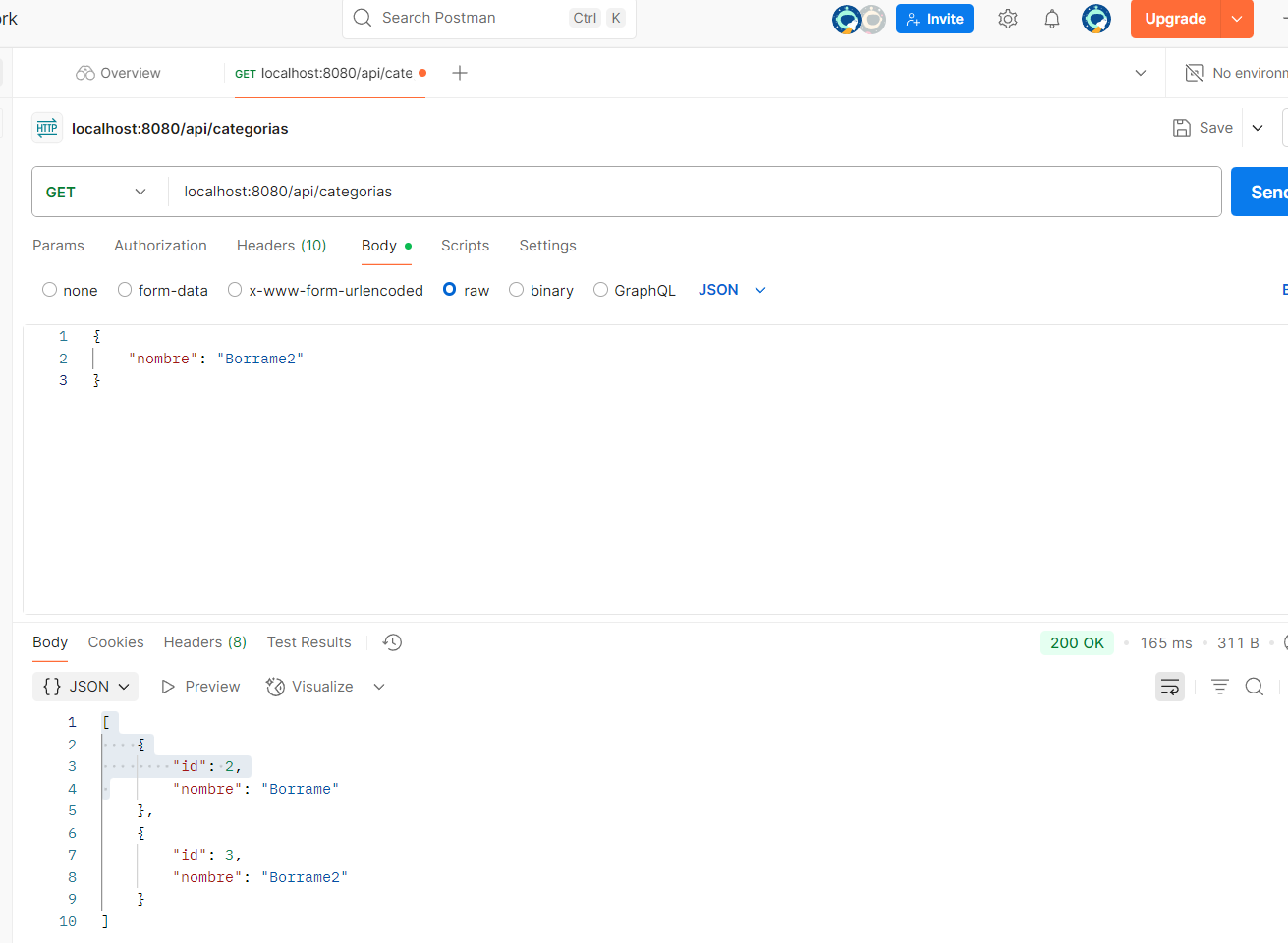
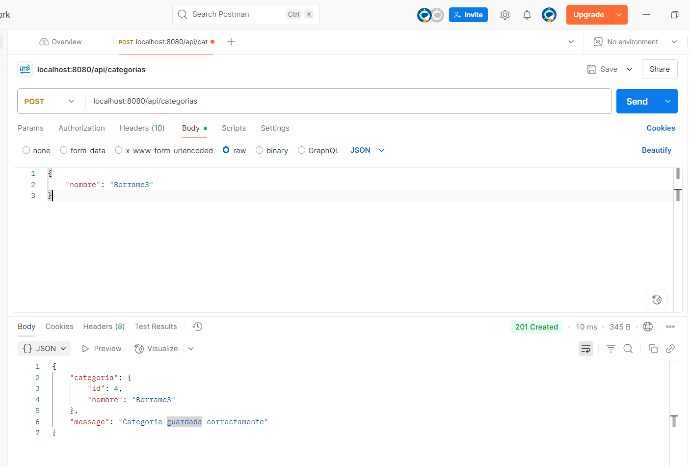
**localhost:8080/api/categorias**

**Configurar los Headers en Postman.**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**

**Comprobado:**



1. **En el package Controller se encuentra la API, es decir, se definen los endpoints. Donde cualquier usuario cliente podrá acceder, media vez todo esté implementado.**
2. **Se hizo cambios en CategoriaRepository:**

*// Método para buscar una categoría por su nombre almacenado en la base de datos a traves de un DTO* Categoria findByNombre(String nombre);  
}

1. **Se hizo cambios en CategoriaService**

*// Este método busca una categoría por su nombre y devuelve un CategoriaDTO*@Override  
public CategoriaDTO findByNombre(String nombre) {  
 Categoria categoria = categoriaRepository.findByNombre(nombre);  
 if (categoria == null) return null;  
 return convertToDTO(categoria);  
}

1. **En CategoriaController se debe agregar este método:**

*// Método para obtener una categoría por su ID*@GetMapping("/categorias/{id}") *// Ruta para obtener una categoría por su ID*public ResponseEntity<?> getById(@PathVariable Long id) {  
 return null;  
}

1. **Modificaciones en CategoriaService (general)**

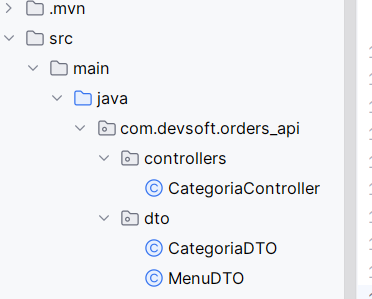
package com.devsoft.orders\_api.services;  
  
import com.devsoft.orders\_api.dto.CategoriaDTO;  
import com.devsoft.orders\_api.entities.Categoria;  
import com.devsoft.orders\_api.interfaces.ICategoriaService;  
import com.devsoft.orders\_api.repository.CategoriaRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.List;  
import java.util.stream.Collectors;  
  
@Service  
public class CategoriaService implements ICategoriaService {  
  
 @Autowired *//Inyectar por dependencia el repositorio* private CategoriaRepository categoriaRepository; *//Esto es un repositorio que maneja las operaciones CRUD de la entidad Categoria* @Override  
 @Transactional(readOnly = true)  
 public List<CategoriaDTO> findAll() { *// Este método obtiene todas las categorías y las convierte a CategoriaDTO* return categoriaRepository.findAll().stream()  
 .map(this::convertToDTO)  
 .collect(Collectors.*toList*());  
 }  
  
 *// Método para obtener todas una lista de entidades Categoria  
 /\*public List<Categoria> getAll(){  
 return categoriaRepository.findAll();  
 }\*/* @Override  
 public CategoriaDTO findById(Long id) {  
 Categoria categoria = categoriaRepository.findById(id).orElse(null);  
 if (categoria == null) return null;  
 return convertToDTO(categoria);  
 }  
  
 @Override  
 public CategoriaDTO save(CategoriaDTO dto) { *// Este método guarda una categoría y devuelve un CategoriaDTO* Categoria catNueva = new Categoria();  
 if(dto.getId() == null) {  
 *//nuevo registro* catNueva.setNombre(dto.getNombre());  
 }else{  
 *//actualizar registro* catNueva.setId(dto.getId());  
 catNueva.setNombre(dto.getNombre());  
 }  
 return convertToDTO(categoriaRepository.save(catNueva));  
 }  
  
 *// Este método busca una categoría por su nombre y devuelve un CategoriaDTO* @Override  
 public CategoriaDTO findByNombre(String nombre) {  
 Categoria categoria = categoriaRepository.findByNombre(nombre);  
 if (categoria == null) return null;  
 return convertToDTO(categoria);  
 }  
  
 @Override  
 public void delete(Long id) { *// Este método elimina una categoría por su ID* categoriaRepository.deleteById(id);  
 }  
  
 *// Método privado para convertir una entidad Categoria a CategoriaDTO* private CategoriaDTO convertToDTO(Categoria categoria) {  
 return new CategoriaDTO(categoria.getId(),categoria.getNombre());  
 }  
}

1. **Modificaciones en CategoriaController (general)**

package com.devsoft.orders\_api.controllers;  
  
import com.devsoft.orders\_api.dto.CategoriaDTO;  
import com.devsoft.orders\_api.entities.Categoria;  
import com.devsoft.orders\_api.interfaces.ICategoriaService;  
import org.apache.coyote.Response;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.dao.DataAccessException;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
import java.util.Objects;  
  
@RestController *// Anotación que indica que esta clase es un controlador REST*@CrossOrigin *// Permite solicitudes desde otros dominios (CORS)/cruzadas*@RequestMapping("/api") *// Define la ruta base para las solicitudes a este controlador*public class CategoriaController { *// Controlador para manejar las operaciones relacionadas con las categorías  
  
 //Anotación de inyección de dependencias* @Autowired  
 private ICategoriaService categoriaService; *// Inyecta el servicio de categorías  
  
 //Endpoint para obtener todas las categorías  
 //Ruta para obtener todas las categorías* @GetMapping("/categorias")  
 public ResponseEntity<?> getAll() {  
 List<CategoriaDTO> categoriaDTOList = categoriaService.findAll();  
 return ResponseEntity.*ok*(categoriaDTOList);  
 }  
  
 *// Endpoint para obtener una categoría por su ID* @GetMapping("/categorias/{id}") *// Ruta para obtener una categoría por su ID* public ResponseEntity<?> getById(@PathVariable Long id) {  
 CategoriaDTO categoriaDTO = null;  
 Map<String, Object> response = new HashMap<>();  
 try {  
 categoriaDTO = categoriaService.findById(id);  
 } catch (DataAccessException e) {  
 response.put("message", "Error al realizar la consulta a la base de datos");  
 response.put("error", e.getMessage());  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 return new ResponseEntity<CategoriaDTO>(categoriaDTO, HttpStatus.*OK*);  
 }  
  
 *// Método guardar una categoría por su ID* @PostMapping("/categorias")  
 public ResponseEntity<?> save(@RequestBody CategoriaDTO dto) {  
 CategoriaDTO catPersisted = new CategoriaDTO();  
 Map<String, Object> response = new HashMap<>();  
 try {  
 *// Validar que el nombre de la categoría no sea nulo o vacío* CategoriaDTO catExiste = categoriaService.findByNombre(dto.getNombre());  
 if (catExiste != null && dto.getId() == null) {  
 response.put("message", "Ya existe una categoria con este nombre, digite otro");  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*CONFLICT*); *// Retorna una respuesta con el estado 409 (CONFLICT) si ya existe una categoría con el mismo nombre* }  
 catPersisted = categoriaService.save(dto);  
 response.put("message", "Categoria guardada correctamente");  
 response.put("categoria", catPersisted);  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*CREATED*); *// Retorna una respuesta con el estado 201 (CREATED) y el objeto guardado* } catch (DataAccessException e) {  
 response.put("message", "Error al insertar el registro, intente nuevamente");  
 response.put("error", e.getMessage());  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*INTERNAL\_SERVER\_ERROR*); *// Retorna una respuesta con el estado 500 (INTERNAL\_SERVER\_ERROR) y el mensaje de error* }  
 }  
  
 *//endpoint para actualizar una categoría por su ID* @PutMapping("/categorias/{id}") *// Ruta para actualizar una categoría por su ID* public ResponseEntity<?> update(@PathVariable Long id, @RequestBody CategoriaDTO dto) {  
 CategoriaDTO catActual = categoriaService.findById(id);  
 CategoriaDTO catUpdated = null;  
 Map<String, Object> response = new HashMap<>();  
 if (catActual == null) {  
 response.put("message", "No existe una categoria con el ID: "  
 .concat(id.toString().concat("no existe en la base de datos")));  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*NOT\_FOUND*); *// Retorna una respuesta con el estado 404 (NOT FOUND) si no se encuentra la categoría* }  
 try {  
 CategoriaDTO catExiste = categoriaService.findByNombre(dto.getNombre());  
 if (catExiste != null && !Objects.*requireNonNull*(catExiste).getId().equals(id)) {  
 response.put("message", "Ya existe una categoria con este nombre, digite otro");  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*CONFLICT*); *// Retorna una respuesta con el estado 409 (CONFLICT) si ya existe una categoría con el mismo nombre* }  
 catActual.setNombre(dto.getNombre());  
 catUpdated = categoriaService.save(catActual);  
 response.put("message", "Categoria actualizada correctamente");  
 response.put("categoria", catUpdated);  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*OK*); *// Retorna* } catch (DataAccessException e) {  
 response.put("message", "Error al actualizar el registro, intente nuevamente");  
 response.put("error", e.getMessage());  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*INTERNAL\_SERVER\_ERROR*); *// Retorna una respuesta con el estado 500 (INTERNAL\_SERVER\_ERROR) y el mensaje de error* }  
 }  
  
 *//endpoint para eliminar una categoría por su ID* @DeleteMapping("/categorias/{id}") *// Ruta para eliminar una categoría por su ID* public ResponseEntity<?> delete(@PathVariable Long id) {  
 Map<String, Object> response = new HashMap<>();  
 CategoriaDTO catActual = categoriaService.findById(id);  
 if (catActual == null) {  
 response.put("message", "No existe una categoria con el ID: "  
 .concat(id.toString().concat("no existe en la base de datos")));  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*NOT\_FOUND*); *// Retorna una respuesta con el estado 404 (NOT FOUND) si no se encuentra la categoría* }  
 try {  
 categoriaService.delete(id);  
 }catch (DataAccessException e){  
 response.put("message", "Error al eliminar el registro, intente nuevamente");  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*INTERNAL\_SERVER\_ERROR*); *// Retorna una respuesta con el estado 500 (INTERNAL\_SERVER\_ERROR) y el mensaje de error* }  
 response.put("message", "Categoria eliminada correctamente");  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*OK*); *// Retorna una respuesta con el estado 200 (OK) y el mensaje de éxito* }  
}

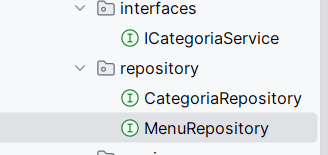
1. **Se creó MenuDTO en package dto.**

package com.devsoft.orders\_api.dto;  
  
  
import com.devsoft.orders\_api.entities.Categoria;  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.math.BigDecimal;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class MenuDTO {  
 private Long id;  
 private String nombre;  
 private String descripcion;  
 private String tipo;  
 private BigDecimal precioUnitario;  
 private String urlImagen;  
 private boolean disponible;  
 private Categoria categoria;  
}

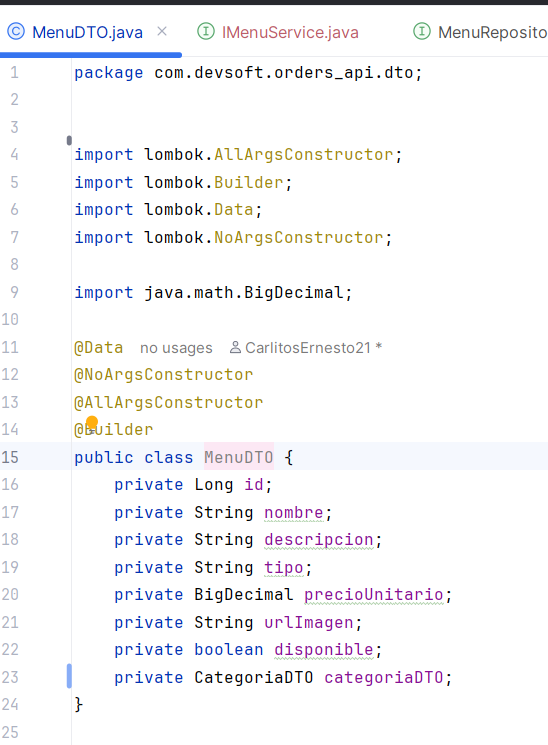


1. **Se creó el Repositorio MenuRepository en package Repository.**

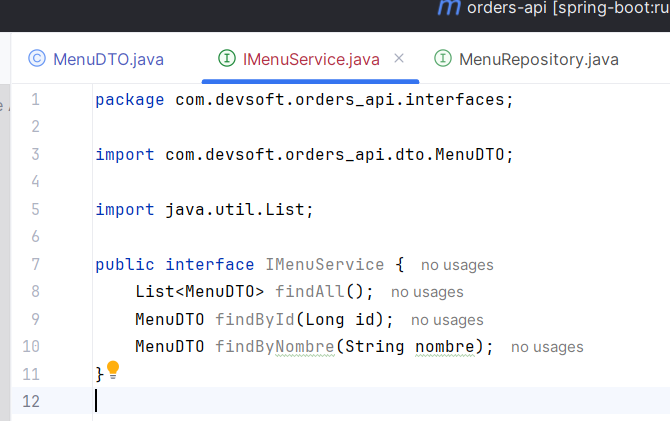
package com.devsoft.orders\_api.repository;  
  
import com.devsoft.orders\_api.entities.Menu;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface MenuRepository extends JpaRepository<Menu, Long> {  
 Menu findByNombre(String nombre);  
}



1. **Se hizo cambios en :**

****

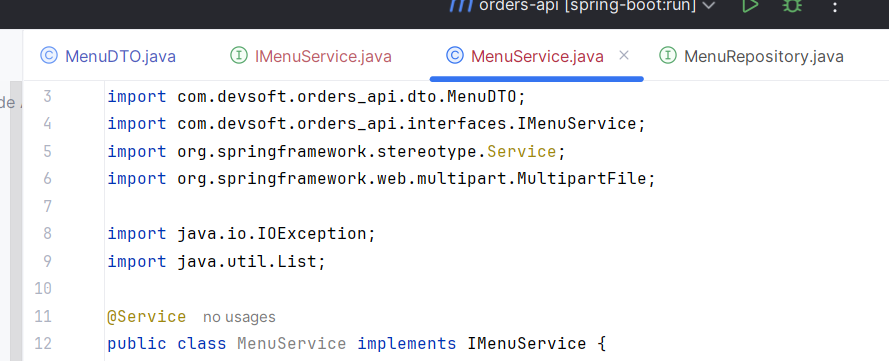
1. **Se creó la interfaz IMenuService**

****

1. **Modificaciones en IMenuService:**

package com.devsoft.orders\_api.interfaces;  
  
import com.devsoft.orders\_api.dto.MenuDTO;  
import org.springframework.web.multipart.MultipartFile;  
  
import java.io.IOException;  
import java.util.List;  
  
public interface IMenuService {  
 List<MenuDTO> findAll();  
 MenuDTO findById(Long id);  
 MenuDTO findByNombre(String nombre);  
 MenuDTO save(MenuDTO dto, MultipartFile imageFile) throws IOException;  
 void delete(Long id);  
}

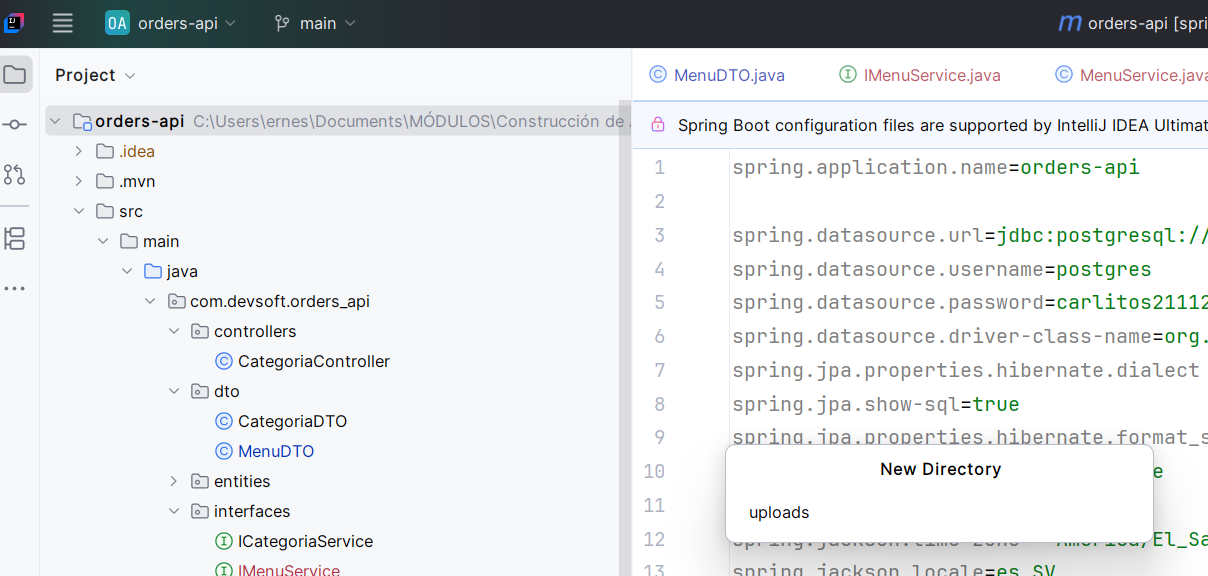
1. **Se creó MenuService en package services.**



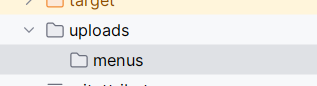
1. **Se creó una propiedad en application.properties:** se agregó hasta el final del archivo.

//Propiedad para almacenar los archivos  
app.upload.dir = "uploads"

1. **Se creó un directorio a nivel de proyecto**



1. **Se creó un directorio dentro de uploads llamado menus**



1. **Cambios en MenuService**
2. **En package útil se creó una clase llamada MenuMapper**

package com.devsoft.orders\_api.utils;  
  
import com.devsoft.orders\_api.dto.CategoriaDTO;  
import com.devsoft.orders\_api.dto.MenuDTO;  
import com.devsoft.orders\_api.entities.Menu;  
  
public class MenuMapper {  
  
 public static MenuDTO toDTO(Menu menu){  
  
 *//Metodo para conveertir una entidad Menu a un DTO MenuDTO* MenuDTO menuDTO = new MenuDTO();  
 menuDTO.setId(menu.getId());  
 menuDTO.setNombre(menu.getNombre());  
 menuDTO.setDescripcion(menu.getDescripcion());  
 menuDTO.setTipo(menu.getTipo());  
 menuDTO.setPrecioUnitario(menu.getPrecioUnitario());  
 menuDTO.setUrlImagen(menu.getUrlImagen());  
 menuDTO.setDisponible(menu.isDisponible());  
 menuDTO.setCategoriaDTO(new CategoriaDTO(menu.getCategoria().getId(),  
 menu.getCategoria().getNombre()));  
 return menuDTO;  
 }  
}



1. **Moficicaciones en MenuRepository se agrego a Optional**

@Repository  
public interface MenuRepository extends JpaRepository<Menu, Long> {  
 Optional<Menu> findByNombre(String nombre);  
}

1. **MenuDTO**

package com.devsoft.orders\_api.dto;  
  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.math.BigDecimal;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class MenuDTO {  
 private Long id;  
 private String nombre;  
 private String descripcion;  
 private String tipo;  
 private BigDecimal precioUnitario;  
 private String urlImagen;  
 private boolean disponible;  
 private CategoriaDTO categoriaDTO;  
}

1. **Menu.java**

package com.devsoft.orders\_api.entities;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.io.Serial;  
import java.io.Serializable;  
import java.math.BigDecimal;  
  
@Setter  
@Getter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "menus", schema = "public", catalog = "orders")  
public class Menu implements Serializable {  
 @Serial  
 private static final long *serialVersionUID* = 1L;  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name = "nombre", nullable = false, length = 80)  
 private String nombre;  
 @Column(name = "descripcion", nullable = false, length = 100)  
 private String descripcion;  
 @Column(name = "tipo", nullable = false, length = 1)  
 private String tipo; *//Producto o Platillo* @Column(name = "precio\_unitario", nullable = false, precision = 8, scale = 2)  
 private BigDecimal precioUnitario;  
 @Column(name = "url\_imagen", nullable = true, length = 100)  
 private String urlImagen;  
 @Column(name = "disponible", columnDefinition = "boolean default true")  
 private boolean disponible;  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "categoria\_id", referencedColumnName = "id", nullable = false)  
 private Categoria categoria;  
}

1. **Cambios en MenuMapper:**

package com.devsoft.orders\_api.utils;  
  
import com.devsoft.orders\_api.dto.CategoriaDTO;  
import com.devsoft.orders\_api.dto.MenuDTO;  
import com.devsoft.orders\_api.entities.Categoria;  
import com.devsoft.orders\_api.entities.Menu;  
  
public class MenuMapper {  
  
 public static MenuDTO toDTO(Menu menu){  
  
 *//Metodo para conveertir una entidad Menu a un DTO MenuDTO* MenuDTO menuDTO = new MenuDTO();  
 menuDTO.setId(menu.getId());  
 menuDTO.setNombre(menu.getNombre());  
 menuDTO.setDescripcion(menu.getDescripcion());  
 menuDTO.setTipo(menu.getTipo());  
 menuDTO.setPrecioUnitario(menu.getPrecioUnitario());  
 menuDTO.setUrlImagen(menu.getUrlImagen());  
 menuDTO.setDisponible(menu.isDisponible());  
 menuDTO.setCategoriaDTO(new CategoriaDTO(menu.getCategoria().getId(),  
 menu.getCategoria().getNombre()));  
 return menuDTO;  
 }  
  
 *// Método para convertir MenuDTO a una entidad Menu* public static Menu toEntity(MenuDTO dto){  
 Menu menu = new Menu();  
 menu.setId(dto.getId());  
 menu.setNombre(dto.getNombre());  
 menu.setDescripcion(dto.getDescripcion());  
 menu.setTipo(dto.getTipo());  
 menu.setPrecioUnitario(dto.getPrecioUnitario());  
 menu.setUrlImagen(dto.getUrlImagen());  
 menu.setDisponible(dto.isDisponible());  
 menu.setCategoria(new Categoria(dto.getCategoriaDTO().getId(),  
 dto.getCategoriaDTO().getNombre()));  
 return menu;  
 }  
}

**Competencia:** Construir la aplicación web respondiendo a los elementos específicos en el diseño.

**Actividad:**

* Creación de el método DELETE (eliminar; parte de CRUD).
* Transacción – parte 1.

1. **Creación de método delete en MenuService**

@Override  
public void delete(Long id) {  
 Menu menuActual = menuRepository.findById(id).orElse(null);  
 try {  
 menuRepository.deleteById(id);  
 if (menuActual != null && menuActual.getUrlImagen() != null) {  
 *// Obtenemos la ruta completa de la imagen anterior para eliminarla* Path rutaAnterior = Paths.*get*(uploadDir, "menus", menuActual.getUrlImagen());  
 Files.*deleteIfExists*(rutaAnterior);  
 }  
 } catch (DataAccessException e) {  
 System.*out*.println("Error: " + e.getMessage());  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
}

1. **Endpoint delete en MenuController:**

@DeleteMapping("/menus/{id}")  
public ResponseEntity<?> delete(@PathVariable Long id) {  
 Map<String, Object> response = new HashMap<>();  
 MenuDTO menuActual = menuService.findById(id);  
 if (menuActual == null) {  
 response.put("message", "No se puede eliminar el menú o producto con ID: "  
 .concat(id.toString().concat(" ya que no existe en la base de datos")));  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*NOT\_FOUND*);  
 }  
 try {  
 menuService.delete(id);  
 response.put("message", "Menú eliminado correctamente");  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*OK*);  
 } catch (DataIntegrityViolationException e) {  
 response.put("message", "No se puede eliminar el menú " + menuActual.getNombre() + ", porque está en uso");  
 response.put("error", e.getMessage());  
 return new ResponseEntity<>(response, HttpStatus.*CONFLICT*);  
 } catch(DataAccessException e) {  
 response.put("message", "No se puede eliminar el menú, ya que tiene ordenes asociadas");  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
}

1. **Crear archivo MesaDTO en package dto:**

package com.devsoft.orders\_api.dto;  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class MesaDTO {  
 private Long id;  
 private int numero;  
 private String ubicacion;  
}

1. **Crear archivo ClienteDTO en package dto:**

package com.devsoft.orders\_api.dto;  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class ClienteDTO {  
 private Long id;  
 private String nombre;  
 private String direccion;  
 private String telefono;  
 private String email;  
 private String tipoCliente;  
}

1. **Crear archive RoleDTO en package dto:**

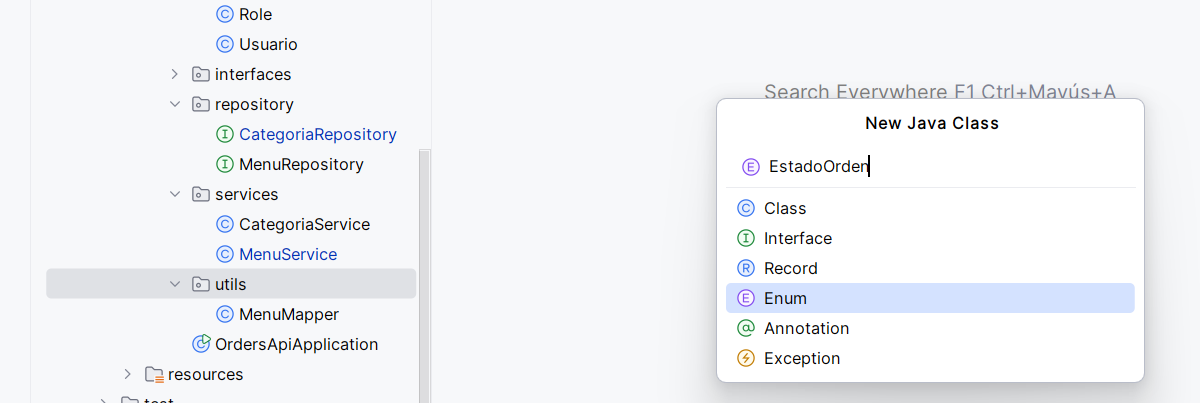
package com.devsoft.orders\_api.dto;  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class RoleDTO {  
 private Long id;  
 private String nombre;  
}

1. **Crear archivo UsuarioDTO en package dto:**

package com.devsoft.orders\_api.dto;  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class UsuarioDTO {  
 private Long id;  
 private String nombre;  
 private String usuario;  
 private boolean activo;  
 private RoleDTO roleDTO;  
}

1. **En el package utils, se crea un ENUM llamado EstadoOrden**

package com.devsoft.orders\_api.utils;  
  
public enum EstadoOrden {  
 *CREADA*,  
 *CONFIRMADA*,  
 *PREPARANDO*,  
 *LISTA*,  
 *ENTREGADA*,  
 *ANULADA*}

****

1. **Cambios en el campo estado de la entidad Orden.java**

Se cambio el tipo de dato para la columna “estado”:

@Column(name = "estado", nullable = false, length = 1)  
@Enumerated(EnumType.*STRING*)  
private EstadoOrden estado;

1. **Crear archivo DetalleOrdenDTO en package dto:**

package com.devsoft.orders\_api.dto;  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.math.BigDecimal;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class DetalleOrdenDTO {  
 private Long id;  
 private MenuDTO menuDTO;  
 private Integer cantidad;  
 private BigDecimal precio;  
 private BigDecimal subtotal;  
}

1. **Crear archivo OrdenDTO en package dto:**

package com.devsoft.orders\_api.dto;  
  
import com.devsoft.orders\_api.utils.EstadoOrden;  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.time.LocalDate;  
import java.time.LocalTime;  
import java.util.List;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class OrdenDTO {  
 private Long id;  
 private String correlativo;  
 private LocalDate fecha;  
 private LocalTime hora;  
 private EstadoOrden estado;

private BigDecimal total;  
 private ClienteDTO clienteDTO;  
 private MesaDTO mesaDTO;  
 private UsuarioDTO usuarioDTO;  
 *//colección para el detalle de la orden* private List<DetalleOrdenDTO> detalle;  
}

1. **Se crea la interface OrdenRepository en package repository:**

package com.devsoft.orders\_api.repository;  
  
import com.devsoft.orders\_api.entities.Orden;  
import com.devsoft.orders\_api.utils.EstadoOrden;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import java.util.List;  
  
public interface OrdenRepository extends JpaRepository<Orden,Long> {  
 List<Orden> findByEstado(EstadoOrden estado);  
}

1. **Se crea la interface DetalleOrdenRepository en package repository:**

package com.devsoft.orders\_api.repository;  
  
import com.devsoft.orders\_api.entities.DetalleOrden;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface DetalleOrdenRepository extends JpaRepository<DetalleOrden, Long> {  
}

1. **Se crea una clase en utils, OrdenMapper:**

package com.devsoft.orders\_api.utils;  
  
  
import com.devsoft.orders\_api.dto.\*;  
import com.devsoft.orders\_api.entities.Orden;  
  
import java.math.BigDecimal;  
import java.math.RoundingMode;  
import java.util.stream.Collectors;  
  
public class OrdenMapper {  
  
 public static OrdenDTO toDTO(Orden orden){  
 if(orden == null) return null;  
 OrdenDTO dto = new OrdenDTO();  
 dto.setId(orden.getId());  
 dto.setCorrelativo(orden.getCorrelativo());  
 dto.setFecha(orden.getFecha());  
 dto.setHora(orden.getHora());  
 dto.setEstado(orden.getEstado());  
 dto.setTotal(orden.getTotal());  
 dto.setClienteDTO(new ClienteDTO(  
 orden.getCliente().getId(),  
 orden.getCliente().getNombre(),  
 orden.getCliente().getDireccion(),  
 orden.getCliente().getTelefono(),  
 orden.getCliente().getEmail(),  
 orden.getCliente().getTipoCliente()  
 ));  
 dto.setMesaDTO(new MesaDTO(  
 orden.getMesa().getId(),  
 orden.getMesa().getNumero(),  
 orden.getMesa().getUbicacion()  
 ));  
 dto.setUsuarioDTO(new UsuarioDTO(  
 orden.getUsuario().getId(),  
 orden.getUsuario().getNombre(),  
 orden.getUsuario().getUsuario(),  
 orden.getUsuario().isActivo(),  
 new RoleDTO(  
 orden.getUsuario().getRole().getId(),  
 orden.getUsuario().getRole().getNombre()  
 )  
 ));  
 *//mapeamos el detalle de la orden* if(orden.getDetalleOrden() != null){  
 dto.setDetalle(orden.getDetalleOrden().stream().map(d ->{  
 DetalleOrdenDTO doDTO = new DetalleOrdenDTO();  
 doDTO.setId(d.getId());  
 doDTO.setCantidad(d.getCantidad());  
 doDTO.setPrecio(d.getPrecio());  
 doDTO.setSubtotal(  
 BigDecimal.*valueOf*(d.getCantidad())  
 .multiply(d.getPrecio())  
 .setScale(2, RoundingMode.*HALF\_UP*)  
 );  
 doDTO.setMenuDTO(new MenuDTO(  
 d.getMenu().getId(),  
 d.getMenu().getNombre(),  
 d.getMenu().getDescripcion(),  
 d.getMenu().getTipo(),  
 d.getMenu().getPrecioUnitario(),  
 d.getMenu().getUrlImagen(),  
 d.getMenu().isDisponible(),  
 new CategoriaDTO(  
 d.getMenu().getCategoria().getId(),  
 d.getMenu().getCategoria().getNombre()  
 )  
 ));  
 return doDTO;  
 }).collect(Collectors.*toList*()));  
 }  
 return dto;  
 }  
}

1. **Se crea una interface IOrdenService.java en package interfaces.**

package com.devsoft.orders\_api.interfaces;  
  
import com.devsoft.orders\_api.dto.OrdenDTO;  
import com.devsoft.orders\_api.utils.EstadoOrden;  
  
import java.util.List;  
  
public interface IOrdenService {  
 List<OrdenDTO> findAll();  
 List<OrdenDTO> findByEstado(EstadoOrden estado);  
 OrdenDTO findById(Long id);  
 OrdenDTO registerOrUpdate(OrdenDTO ordenDTO);  
 void anular(Long id);  
 OrdenDTO changeState(OrdenDTO dto);  
}

1. **Se crea la interface ClienteRepository.java en package repository:**

package com.devsoft.orders\_api.repository;  
  
import com.devsoft.orders\_api.entities.Cliente;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface ClienteRepository extends JpaRepository<Cliente, Long> {  
}

1. **Se crea la interface MesaRepository.java en package repository:**

package com.devsoft.orders\_api.repository;  
  
import com.devsoft.orders\_api.entities.Mesa;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface MesaRepository extends JpaRepository<Mesa, Long> {  
}

1. **Se crea la interface UsuarioRepository.java en package repository:**

package com.devsoft.orders\_api.repository;  
  
import com.devsoft.orders\_api.entities.Usuario;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface UsuarioRepository extends JpaRepository<Usuario, Long> {  
}

1. **Se crea la clase OrdenService.java en package services:**

package com.devsoft.orders\_api.services;  
  
import com.devsoft.orders\_api.dto.DetalleOrdenDTO;  
import com.devsoft.orders\_api.dto.OrdenDTO;  
import com.devsoft.orders\_api.entities.\*;  
import com.devsoft.orders\_api.interfaces.IOrdenService;  
import com.devsoft.orders\_api.repository.\*;  
import com.devsoft.orders\_api.utils.EstadoOrden;  
import com.devsoft.orders\_api.utils.OrdenMapper;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import javax.swing.text.html.Option;  
import java.time.LocalDate;  
import java.time.LocalTime;  
import java.time.format.DateTimeFormatter;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class OrdenService implements IOrdenService {  
 @Autowired  
 private OrdenRepository ordenRepository;  
 @Autowired  
 private ClienteRepository clienteRepository;  
 @Autowired  
 private MesaRepository mesaRepository;  
 @Autowired  
 private UsuarioRepository usuarioRepository;  
 @Autowired  
 private MenuRepository menuRepository;  
  
 @Override  
 @Transactional(readOnly = true)  
 public List<OrdenDTO> findAll() {  
 return ordenRepository.findAll()  
 .stream().map(OrdenMapper::*toDTO*).toList();  
 }  
  
 @Override  
 @Transactional(readOnly = true)  
 public List<OrdenDTO> findByEstado(EstadoOrden estado) {  
 return ordenRepository.findByEstado(estado)  
 .stream().map(OrdenMapper::*toDTO*).toList();  
 }  
  
 @Override  
 @Transactional(readOnly = true)  
 public OrdenDTO findById(Long id) {  
 return ordenRepository.findById(id)  
 .map(OrdenMapper::*toDTO*)  
 .orElse(null);  
 }  
  
 @Override  
 @Transactional  
 public OrdenDTO registerOrUpdate(OrdenDTO ordenDTO) {  
 *//validamos referencias de objetos* Optional<Cliente> clienteOpt = clienteRepository.findById(ordenDTO.getClienteDTO().getId());  
 Optional<Mesa> mesaOpt = mesaRepository.findById(ordenDTO.getMesaDTO().getId());  
 Optional<Usuario> userOpt = usuarioRepository.findById(ordenDTO.getUsuarioDTO().getId());  
 if (clienteOpt.isEmpty() || mesaOpt.isEmpty() || userOpt.isEmpty()  
 || ordenDTO.getDetalle() == null || ordenDTO.getDetalle().isEmpty()){  
 return null; *//gestionar en el controlador* }  
 Orden orden = null; *//objeto a persistir* if (ordenDTO.getId() == null){  
 *//Lógica para registrar una nueva orden* orden = new Orden();  
 *//seteando los atributos* orden.setFecha(LocalDate.*now*());  
 orden.setHora(LocalTime.*now*());  
 orden.setEstado(EstadoOrden.*CREADA*);  
 orden.setCorrelativo(generarCorrelativo());  
 orden.setTotal(ordenDTO.getTotal());  
 orden.setDetalleOrden(new ArrayList<>());  
 }else{  
 *//Lógica para actualizar una orden existente* Optional<Orden> ordenOpt = ordenRepository.findById(ordenDTO.getId());  
 if (ordenOpt.isEmpty()){  
 return null;  
 }  
 orden = ordenOpt.get();  
 *//limpiamos el detalle previo* orden.getDetalleOrden().clear();  
 }  
 *//seteamos nuevas referencias* orden.setCliente(clienteOpt.get());  
 orden.setMesa(mesaOpt.get());  
 orden.setUsuario(userOpt.get());  
  
 *//agregando el detalle de la orden* for (DetalleOrdenDTO detalleDTO : ordenDTO.getDetalle()) {  
 Optional<Menu> menuOpt = menuRepository.findById(detalleDTO.getMenuDTO().getId());  
 if (menuOpt.isEmpty()) return null;  
 Menu menu = menuOpt.get();  
 DetalleOrden detalleOrden = new DetalleOrden();  
 detalleOrden.setCantidad(detalleDTO.getCantidad());  
 detalleOrden.setPrecio(detalleDTO.getPrecio());  
 detalleOrden.setSubtotal(detalleDTO.getSubtotal());  
 detalleOrden.setMenu(menu);  
 detalleOrden.setOrden(orden);  
 *//agregamos el detalleOrden a la Orden* orden.getDetalleOrden().add(detalleOrden);  
 }  
 Orden ordenPersisted = ordenRepository.save(orden);  
 return OrdenMapper.*toDTO*(ordenPersisted);  
 }  
  
 @Override  
 @Transactional  
 public void anular(Long id) {  
 Optional ordenOpt = ordenRepository.findById(id);  
 if (ordenOpt.isEmpty()) return;  
 Orden orden = (Orden) ordenOpt.get();  
 orden.setEstado(EstadoOrden.*ANULADA*);  
 ordenRepository.save(orden);  
 }  
  
 @Override  
 @Transactional  
 public OrdenDTO changeState(OrdenDTO dto) {  
 Optional ordenOpt = ordenRepository.findById(dto.getId());  
 if (ordenOpt.isEmpty()) return null;  
 Orden orden = (Orden) ordenOpt.get();  
 *//cambiamos el estado de la orden* orden.setEstado(dto.getEstado());  
 return OrdenMapper.*toDTO*(ordenRepository.save(orden));  
 }  
  
 *//métodos auxiliares* private String generarCorrelativo() {  
 LocalDate currentDate = LocalDate.*now*();  
 String yearMonth = currentDate.format(DateTimeFormatter.*ofPattern*("yyyyMM"));  
 Long maxCorrelativo = ordenRepository.findMaxCorrelativoByYearAndMonth(yearMonth);  
 Long nextCorrelativo = (maxCorrelativo != null) ? maxCorrelativo + 1 : 1;  
 return yearMonth + String.*format*("%04d", nextCorrelativo);  
 }  
}

1. **Se agregó un método nuevo a la interface OrdenRepository:**

package com.devsoft.orders\_api.repository;  
  
import com.devsoft.orders\_api.entities.Orden;  
import com.devsoft.orders\_api.utils.EstadoOrden;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
  
import java.util.List;  
  
public interface OrdenRepository extends JpaRepository<Orden,Long> {  
 List<Orden> findByEstado(EstadoOrden estado);  
  
 *//Método para obtener el correlativo maximo por mes y año* @Query("SELECT MAX(CAST(SUBSTRING(o.correlativo,7) AS Long)) FROM Orden o WHERE SUBSTRING(o.correlativo,1,6) = :yearMonth")  
 Long findMaxCorrelativoByYearAndMonth(@Param("yearMonth") String yearMonth);  
}

1. **Creando controlador en package Controller “OrdenController”**

package com.devsoft.orders\_api.controllers;  
  
import com.devsoft.orders\_api.dto.CategoriaDTO;  
import com.devsoft.orders\_api.dto.OrdenDTO;  
import com.devsoft.orders\_api.entities.Orden;  
import com.devsoft.orders\_api.interfaces.IOrdenService;  
import com.devsoft.orders\_api.utils.EstadoOrden;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.dao.DataAccessException;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
import java.util.Objects;  
  
@RestController  
@CrossOrigin  
@RequestMapping("/api")  
public class OrdenController {  
 @Autowired  
 private IOrdenService ordenService;  
  
 *//Endpoint para listar todas las órdenes* @GetMapping("/ordenes")  
 public ResponseEntity<?> getAll() {  
 List<OrdenDTO> ordenes = ordenService.findAll();  
 return ResponseEntity.*ok*(ordenes);  
 }  
  
 *//Endpoint para listar órdenes por estado* @GetMapping("/ordenes/estado/{estado}")  
 public ResponseEntity<?> getAllEstado(@PathVariable EstadoOrden estado) {  
 List<OrdenDTO> lista = ordenService.findByEstado(estado);  
 return ResponseEntity.*ok*(lista);  
 }  
  
 *//Endpoint para obtener una orden por su ID* @GetMapping("/ordenes/{id}") *// Ruta para obtener una categoría por su ID* public ResponseEntity<?> getById(@PathVariable Long id) {  
 Map<String, Object> response = new HashMap<>();  
 try {  
 OrdenDTO dto = ordenService.findById(id);  
 if (dto == null) {  
 response.put("message", "La orden con ID: "  
 .concat(id.toString()  
 .concat(" no existe en la base de datos")));  
 return new ResponseEntity<>(response, HttpStatus.*NOT\_FOUND*);  
 }  
 return new ResponseEntity<>(dto, HttpStatus.*OK*);  
 } catch (DataAccessException e) {  
 response.put("message", "Error al realizar la consulta a la base de datos");  
 response.put("error", e.getMessage());  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
  
 *//Endpoint para crear una nueva orden* @PostMapping("/ordenes")  
 public ResponseEntity<?> create(@RequestBody OrdenDTO dto) {  
 Map<String, Object> response = new HashMap<>();  
 try {  
 OrdenDTO dtoCreated = ordenService.registerOrUpdate(dto);  
 if (dtoCreated == null) {  
 response.put("message", "No se pudo crear la orden, verifique los datos ingresados");  
 return new ResponseEntity<>(response, HttpStatus.*BAD\_REQUEST*);  
 }  
 response.put("message", "La orden ha sido creada con éxito");  
 response.put("orden", dtoCreated);  
 return new ResponseEntity<>(response, HttpStatus.*CREATED*);  
 } catch (DataAccessException e) {  
 response.put("message", "Error al realizar la órden a la base de datos");  
 response.put("error", e.getMessage());  
 return new ResponseEntity<>(response, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
  
 *//Endpoint para actualizar una orden* @PutMapping("/ordenes/{id}") *// Ruta para actualizar una categoría por su ID* public ResponseEntity<?> update(@PathVariable Long id,  
 @RequestBody OrdenDTO dto) {  
 OrdenDTO ordenActual = ordenService.findById(id);  
 Map<String, Object> response = new HashMap<>();  
 if (ordenActual == null) {  
 response.put("message", "No se puede editar la órden con el ID: "  
 .concat(id.toString().concat("no existe en la base de datos")));  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*NOT\_FOUND*);  
 }  
 try {  
 ordenActual.setClienteDTO(dto.getClienteDTO());  
 ordenActual.setMesaDTO(dto.getMesaDTO());  
 ordenActual.setUsuarioDTO(dto.getUsuarioDTO());  
 ordenActual.setTotal(dto.getTotal());  
 ordenActual.setDetalle(dto.getDetalle());  
 OrdenDTO dtoUptated = ordenService.registerOrUpdate(ordenActual);  
 if (dtoUptated == null) {  
 response.put("message", "No se pudo actualizar la orden, verifique los datos ingresados");  
 return new ResponseEntity<>(response, HttpStatus.*BAD\_REQUEST*);  
 }  
 response.put("message", "La orden ha sido actualizada con éxito");  
 response.put("orden", dtoUptated);  
 return new ResponseEntity<>(response, HttpStatus.*ACCEPTED*);  
 } catch (DataAccessException e) {  
 response.put("message", "Error al actualizar el registro, intente nuevamente");  
 response.put("error", e.getMessage());  
 return new ResponseEntity<Map<String, Object>>(response, HttpStatus.*INTERNAL\_SERVER\_ERROR*); *// Retorna una respuesta con el estado 500 (INTERNAL\_SERVER\_ERROR) y el mensaje de error* }  
 }  
  
 *//Endpoint para cambiar el estado de una orden* @PutMapping("/ordenes/cambiar-estado")  
 public ResponseEntity<?> changeState(@RequestBody OrdenDTO dto) {  
 Map<String, Object> response = new HashMap<>();  
 try {  
 OrdenDTO ordenUpdated = ordenService.changeState(dto);  
 if (ordenUpdated == null) {  
 response.put("message", "No se pudo cambiar el estado de la orden, verifique los datos ingresados");  
 return new ResponseEntity<>(response, HttpStatus.*BAD\_REQUEST*);  
 }  
 response.put("message", "El estado de la orden ha cambiado a: " + ordenUpdated.getEstado());  
 response.put("orden", ordenUpdated);  
 return new ResponseEntity<>(response, HttpStatus.*ACCEPTED*);  
 } catch (DataAccessException e) {  
 response.put("message", "Error al cambiar el estado de la orden");  
 response.put("error", e.getMessage());  
 return new ResponseEntity<>(response, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
  
 *//Endpoint para anular una orden* @PutMapping("/ordenes/{id}")  
 public ResponseEntity<?> anularOrden (@PathVariable Long id) {  
 Map<String, Object> response = new HashMap<>();  
 OrdenDTO ordenActual = ordenService.findById(id);  
 if (ordenActual == null) {  
 response.put("message", "No se puede anular la orden con el ID: "  
 .concat(id.toString().concat(" no existe en la base de datos")));  
 return new ResponseEntity<>(response, HttpStatus.*NOT\_FOUND*);  
 }  
 try {  
 ordenService.anular(id);  
 response.put("message", "La orden ha sido anulada con éxito");  
 return new ResponseEntity<>(response, HttpStatus.*OK*);  
 } catch (DataAccessException e) {  
 response.put("message", "Error al anular la orden");  
 response.put("error", e.getMessage());  
 return new ResponseEntity<>(response, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
 }  
}

1. **Endpoints.**

Colección de órdenes:

localhost:8080/api/ordenes/

Órdenes por estado

localhost:8080/api/ordenes/estado/ENTREGADA

Órdenes por ID:

localhost:8080/api/ordenes/1

1. **Insertando datos a la base.**

--INSERTANDO ROLES

--INSERT INTO roles (nombre) VALUES ('ADMIN');

--INSERT INTO roles (nombre) VALUES ('CLIENTE');

--UPDATE roles SET nombre = 'MESERO/A' WHERE id=2;

--SELECT \* FROM roles;

--INSERTANDO UN USUARIO

--INSERT INTO usuarios (nombre, usuario, password,activo,role\_id) VALUES ('ernesto','ernesto','carlitos',true,2);

--SELECT \* FROM usuarios;

--INSERTANDO MESAS

--INSERT INTO mesas(numero,ubicacion) VALUES(1,'Interior');

--INSERT INTO mesas(numero,ubicacion) VALUES(2,'Interior');

--INSERT INTO mesas(numero,ubicacion) VALUES(3,'Exterior');

--INSERT INTO mesas(numero,ubicacion) VALUES(4,'Exterior');

--INSERT INTO mesas(numero,ubicacion) VALUES(5,'Terraza');

--SELECT \* FROM mesas;

--INSERTANDO CLIENTES

--INSERT INTO clientes(nombre, direccion,email,telefono,tipo\_cliente)

--VALUES ('Marta Beatriz','Cantón Santa Bárbara, El Paraíso', 'martabeatriz@gmail.com', '6300-6520','P');

--SELECT \* FROM clientes;

1. **JSON:**

{

"total": 0.0,

"clienteDTO": {"id":1},

"mesaDTO": {"id":3},

"usuarioDTO": {"id":1},

"detalle": [

{

"menuDTO": {"id":4},

"cantidad": 2,

"precio": 12,

"subtotal": 24

},

{

"menuDTO": {"id":1},

"cantidad": 3,

"precio": 4,

"subtotal": 12

}

]

}

1. **U**
2. **U**
3. **U**
4. **U**
5. **U**
6. **U**
7. **U**
8. **U**
9. **U**