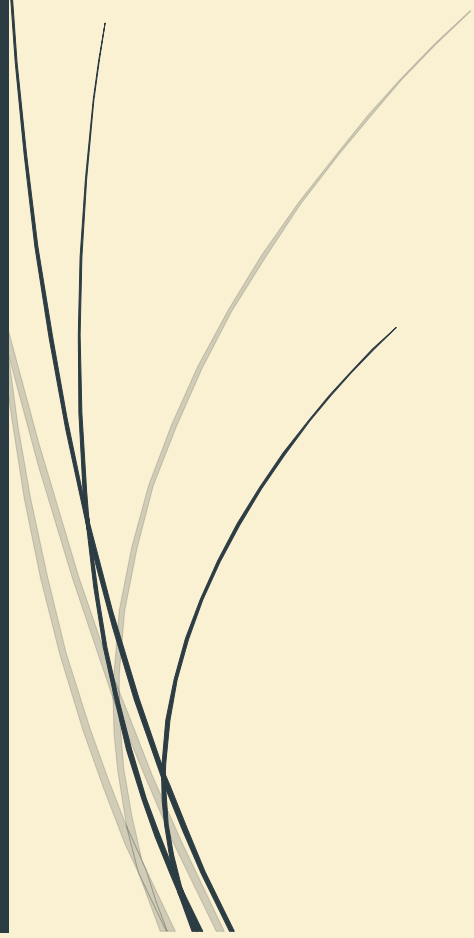
10-12-2023

Proyecto

Thymeleaf – Springboot – JPA - Oracle



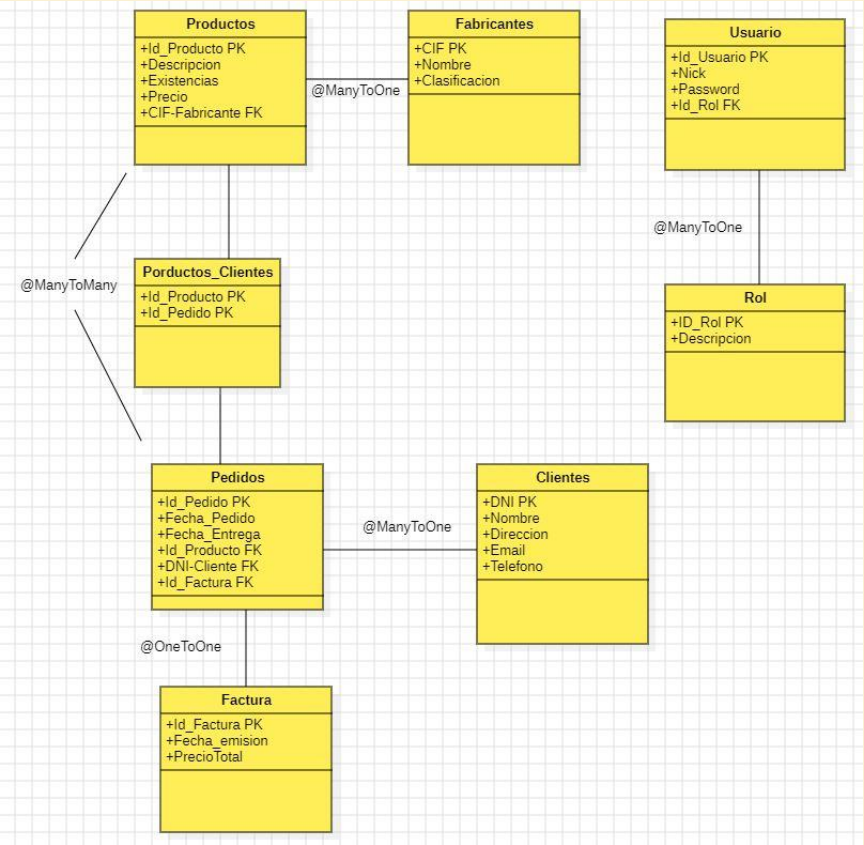
Pablo López Sánchez

2º GS DESARROLLO DE APLICACIONES WEB

INDICE

Creación de tablas

Para mostrar las relaciones que tendrán las tablas y las entidades entre si he realizado un **diagrama entidad-relación** para que se vea mucho mejor:



Como se puede observar, por un lado, tendré **usuario y rol**, y por otro lado estará **fabricantes, productos, pedidos, clientes y factura.**

Usuario - Rol: tendrán una relación **@ManyToOne**.

Fabricantes - Productos: tendrán una relación **@ManyToOne**.

Productos - Pedidos: tendrán una relación **@ManyToMany** por lo que he creado una tabla llamada **Productos_Clientes** para unir ambas.

Pedidos - Factura: tendrán una relación **@OneToOne**.

Pedidos - Clientes: tendrán una relación **@ManyToOne**.

Las tablas se deberán **crear en un orden adecuado** para que no haya conflicto:

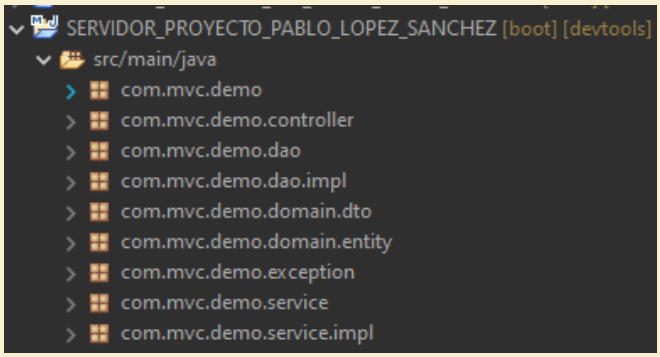
```
CREATE TABLE USUARIO (...
CREATE TABLE CLIENTE (...
CREATE TABLE FABRICANTE (...
CREATE TABLE FACTURA (...
CREATE TABLE PRODUCTO (...
CREATE TABLE PEDIDO (...
CREATE TABLE PRODUCTO_PEDIDO (...
CREATE SEQUENCE SEQ_USUARIO...
CREATE SEQUENCE SEQ_ROL...
CREATE SEQUENCE SEQ_FACTURA...
CREATE SEQUENCE SEQ_PRODUCTO...
CREATE SEQUENCE SEQ_PEDIDO...
```

Además, recalcar que los id de usuario, rol, factura, producto y pedido se generaran con una **secuencia**. Fabricante utilizara un **CIF** y Cliente un **DNI**.

Estructura del proyecto

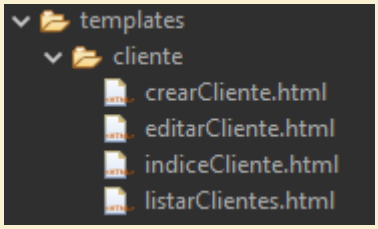
Paquetes:

Con respecto a los paquetes la estructura a seguir es la misma que llevamos usando durante todo el curso



Templates:

Y para las vistas de cada elemento del proyecto usare 4, **índice** (decidiremos que hacer), **crear** (vista para crear), **editar** (vista para editar un elemento) y **listar** (vista para mostrar todas las instancias del elemento):



Entidades y DTO’s

ClienteEntity:

```java
@Entity
@Table(name = "CLIENTE")
public class ClienteEntity implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Column(name = "DNI")
    private String dni;

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

    @Column(name = "DIRECCION")
    private String direccion;

    @Column(name = "EMAIL")
    private String email;

    @Column(name = "TELEFONO")
    private String telefono;
```

FabricanteEntity:

```java
@Entity
@Table(name = "FABRICANTE")
public class FabricanteEntity implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Column(name = "CIF")
    private String cif;

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

    @Column(name = "CLASIFICACION")
    private int clasificacion;
```

FacturaEntity:

```java
@Entity
@Table(name = "FACTURA")
public class FacturaEntity implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "seqFactura")
    @SequenceGenerator(name = "seqFactura", allocationSize = 1, sequenceName = "SEQ_FACTURA")
    @Column(name = "ID_FACTURA")
    private Integer idFactura;

    @Column(name = "FECHA_EMISION")
    private LocalDate fechaEmision;

    @Column(name = "PRECIO_TOTAL")
    private double precioTotal;
```

PedidoEntity:

```java
@Entity
@Table(name = "PEDIDO")
public class PedidoEntity implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "seqPedido")
    @SequenceGenerator(name = "seqPedido", allocationSize = 1, sequenceName = "SEQ_PEDIDO")
    @Column(name = "ID_PEDIDO")
    private Integer idPedido;

    @Column(name = "FECHA_PEDIDO")
    private LocalDate fechaPedido;

    @Column(name = "FECHA_ENTREGA")
    private LocalDate fechaEntrega;

    @ManyToOne
    @JoinColumn(name = "CLIENTE_DNI", referencedColumnName = "DNI")
    private ClienteEntity cliente;

    @OneToOne
    @JoinColumn(name = "FACTURA_ID")
    private FacturaEntity factura;

    @JoinTable(name = "PRODUCTO_PEDIDO",
            joinColumns = @JoinColumn(name = "ID_PEDIDO"),
            inverseJoinColumns = @JoinColumn(name = "ID_PRODUCTO")
    )
    @ManyToMany
    private List<ProductoEntity> lstProductos;
```

ProductoEntity:

```java
@Entity
@Table(name = "PRODUCTO")
public class ProductoEntity implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "seqProducto")
    @SequenceGenerator(name = "seqProducto", allocationSize = 1, sequenceName = "SEQ_PRODUCTO")
    @Column(name = "ID_PRODUCTO")
    private Integer idProducto;

    @Column(name = "DESCRIPCION")
    private String descripcion;

    @Column(name = "EXISTENCIAS")
    private int existencias;

    @Column(name = "PRECIO")
    private double precio;

    @ManyToOne
    @JoinColumn(name = "FABRICANTE_CIF", referencedColumnName = "CIF")
    private FabricanteEntity fabricante;

    @ManyToMany(mappedBy = "lstProductos")
    private List<PedidoEntity> lstPedidos;
```

RolEntity:

```java
@Entity
@Table(name = "ROL")
public class RolEntity implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "seqRol")
    @SequenceGenerator(name = "seqRol", allocationSize = 1, sequenceName = "SEQ_ROL")
    @Column(name = "ID_ROL")
    private Integer idRol;

    @Column(name = "DESCRIPCION")
    private String descripcion;
```

UsuarioEntity:

```java
@Entity
@Table(name = "USUARIO")
public class UsuarioEntity implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "seqUsuario")
    @SequenceGenerator(name = "seqUsuario", allocationSize = 1, sequenceName = "SEQ_USUARIO")
    @Column(name = "ID_USUARIO")
    private Integer idUsuario;

    @Column(name = "NICK")
    private String nick;

    @Column(name = "PASSWD")
    private String passwd;

    @ManyToOne
    @JoinColumn(name = "ROL_ID", referencedColumnName = "ID_ROL")
    private RolEntity rol;
```

Los **DTO's** son igual que los Entity pero estos no están asociados a columnas ni tienen relaciones ni nada.

Métodos de los DAO's

Los métodos que necesito para gestionar mis datos son guardar, editar, eliminar, encontrar por id y encontrar a todos:

```java
public interface IDAOCliente {
    public void save(ClienteEntity cliente);
    public void merge(ClienteEntity cliente);
    public void remove(ClienteEntity cliente);
    public ClienteEntity findById(String dni);
    public List<ClienteEntity> findAll();
}
```

Todas las implementaciones llevan la etiqueta **@Repository** y la inyección del **EntityManager** con la etiqueta **@PersistenceContext**:

```java
@Repository
public class PedidoDAOImpl implements IDAOPedido {

    @PersistenceContext
    private EntityManager em;
```

Métodos de los servicios

Los métodos de todos los servicios son guardar, actualizar, eliminar, obtener por id y obtener todos:

```java
public interface IServiceCliente {
    public void guardarCliente(ClienteDTO clienteDTO) throws MiExcepcion;
    public void actualizarCliente(ClienteDTO clienteDTO);
    public void eliminarCliente(String dni);
    public ClienteDTO obtenerClientePorId(String dni);
    public List<ClienteDTO> listarTodosLosClientes();
}
```

Todos los servicios tienen la etiqueta **@Service** y las inyecciones que necesiten para realizar todas las funciones etiquetadas con **@Autowired**:

```java
@Service
public class ProductoServiceImpl implements IServiceProducto{

    @Autowired
    private IDAOProducto productoDAO;

    @Autowired
    private IDAOFabricante fabricanteDAO;
```

Los métodos que realizan modificaciones en la base de datos como guardar, actualizar y eliminar están etiquetados con **@Transactional**:

```java
@Override
@Transactional
public void guardarPedido(
```

Y los métodos que solo recogen información como obtener por id y obtener todos, con **@Transactional** y la condición **(readOnly=true)**:

```java
@Override
@Transactional(readOnly = true)
public List<PedidoDTO> listarTodosLosPedidos()
```

Por último, he realizado una excepción a la hora de crear clientes y fabricantes para que en el caso de que el dni o cif no se haya introducido o este repetido, lleve a mi página de error:

```java
@Override
@Transactional
public void guardarCliente(ClienteDTO clienteDTO) throws MiExcepcion {
    ClienteEntity clienteEntity = null;
    try {
        clienteEntity = clienteDAO.findById(clienteDTO.getDni());

        if (null != clienteEntity || clienteDTO.getDni() == "") {
            throw new MiExcepcion(100, "Error en cliente duplicado");
        }
        clienteEntity = new ClienteEntity();
        clienteEntity.setDni(clienteDTO.getDni());
        clienteEntity.setNombre(clienteDTO.getNombre());
        clienteEntity.setDireccion(clienteDTO.getDireccion());
        clienteEntity.setEmail(clienteDTO.getEmail());
        clienteEntity.setTelefono(clienteDTO.getTelefono());

        clienteDAO.save(clienteEntity);
    } catch (MiExcepcion e) {
        throw e;
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
```

Controladores y vistas

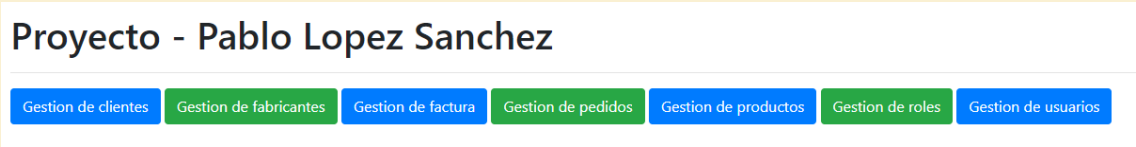
A parte de mi index controller que muestra el índice desde el que partiremos en la web, tengo todos los controladores con la etiqueta **@Controller** y **@RequestMapping** y las inyecciones de los servicios que necesiten etiquetados con **@Autowired:**

```java
@Controller
@RequestMapping("/pedido")
public class PedidoController {

    @Autowired
    private IServicePedido pedidoService;

    @Autowired
    private IServiceCliente clienteService;

    @Autowired
    private IServiceFactura facturaService;

    @Autowired
    private IServiceProducto productoService;
```

En los controladores tengo los métodos para moverse entre vistas a través de href's y los métodos que hacen que pueda crear, modificar, borrar y ver los datos en base de datos y en las vistas.

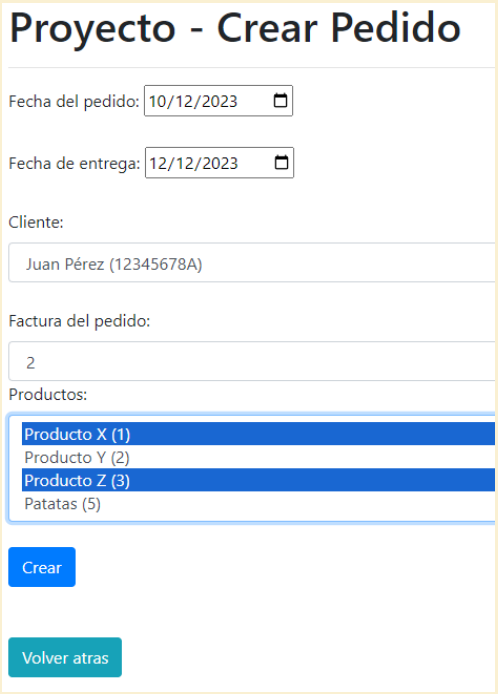
Movilidad por la web

Partiremos desde en index.html:



En todos los elementos tenemos crear, ver y volver atrás:

En la vista de crear, podremos crear entidades que se almacenaran en base de datos:



También podremos ver nuestras entidades que tendrán las acciones de borrar y editar:

Esto sería la vista de editar:

Y, por último, la vista de cuando salta **MiExcepcion**, por ejemplo, cuando le damos un dni a un cliente o cif a un fabricante y este ya existe o dejamos el campo vacío: