Archivos Base:

Base:

```
package com.example.InvOpBack.Entities;

import jakarta.persistence.*;
import lombok.*;

import java.io.Serializable;
import java.time.LocalDateTime;

@MappedSuperclass
@Getter
@Setter
@AllArgsConstructor
public class Base implements Serializable {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   @Column(columnDefinition = "boolean default true") // Para que la base de datos lo
inicialice a true
   private boolean estado = true; // Por defecto activo

   private LocalDateTime fechaAlta;
   private LocalDateTime fechaModificacion; // Fecha y hora de la última modificación.
   private LocalDateTime fechaBaja;

   public Base() {
       this.estado = true; // Asegurarse de que esté en true por defecto
       this.fechaAlta = LocalDateTime.now(); // Se establece la fecha de alta al crear la
entidad
   }
}
```

BaseRepository:

```
package com.example.InvOpBack.Repository;

import com.example.InvOpBack.Entities.Base;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.NoRepositoryBean;

import java.io.Serializable;

@NoRepositoryBean
public interface BaseRepository <E extends Base, ID extends Serializable> extends
JpaRepository<E, ID> {

}
```

BaseService:

```
package com.example.InvOpBack.Service;

import com.example.InvOpBack.Entities.Base;

import java.io.Serializable;
import java.util.List;
```

```java
public interface BaseService <E extends Base, ID extends Serializable>{
   public E findById(ID id) throws Exception;
   public List<E> findAll() throws Exception;
   public E save(E entity) throws Exception;
   public E update (ID id, E entity) throws Exception;
   public boolean delete(ID id) throws Exception;

}
```

BaseServiceImpl:

```java
package com.example.InvOpBack.Service;

import com.example.InvOpBack.Entities.Base;
import com.example.InvOpBack.Repository.BaseRepository;
import jakarta.transaction.Transactional;
import org.springframework.data.repository.NoRepositoryBean;

import java.io.Serializable;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors; // Para filtrar por estado


public abstract class BaseServiceImpl <E extends Base, ID extends Serializable>
implements BaseService<E,ID>{

   protected BaseRepository<E, ID> baseRepository;

   public BaseServiceImpl(BaseRepository<E, ID> baseRepository) {
       this.baseRepository = baseRepository;
   }

   @Override
   @Transactional
   public E findById(ID id) throws Exception {
       try {
           // Asegúrate de que solo se busquen entidades con estado = true si es el
comportamiento deseado
           // Por ahora, busca por ID y luego verifica el estado
           Optional<E> entityOptional = baseRepository.findById(id);
           if (entityOptional.isPresent() && entityOptional.get().isEstado()) { // Solo
devuelve si está activo
               return entityOptional.get();
           } else if (entityOptional.isPresent() && !entityOptional.get().isEstado()) {
               throw new Exception("La entidad con ID: " + id + " se encuentra dada de
baja.");
           } else {
               throw new Exception("No se encontró la entidad con ID: " + id);
           }
       } catch (Exception e) {
           throw new Exception("Error al buscar por ID: " + id + ": " + e.getMessage(),
e);
       }
   }

   @Override
   @Transactional // Para que la lista también considere el estado
   public List<E> findAll() throws Exception {
       try {
           // Puedes agregar una condición aquí si solo quieres listar los activos
           // Por ejemplo, si BaseRepository tiene un metodo List<E> findByEstado(boolean
estado);
```

```
            // return baseRepository.findByEstado(true);
            return baseRepository.findAll(); // Devuelve todos, el filtrado por estado
podría ser en el controller o servicio específico
        } catch (Exception e) {
            throw new Exception("Error al obtener todos los registros", e);
        }
    }

    @Override
    @Transactional
    public E save(E entity) throws Exception {
        try {
            // La fecha de alta se setea en el constructor de Base, pero puedes asegurarte
aquí
            if (entity.getFechaAlta() == null) {
                entity.setFechaAlta(LocalDateTime.now());
            }
            // Asegurarse de que el estado sea true al guardar uno nuevo
            entity.setEstado(true);
            return baseRepository.save(entity);
        } catch (Exception e) {
            throw new Exception("Error al guardar la entidad: " + e.getMessage(), e);
        }
    }

    @Override
    @Transactional
    public E update(ID id, E entity) throws Exception {
        try {
            Optional<E> entityOptional = baseRepository.findById(id);
            if (entityOptional.isPresent()) {
                E entityExistente = entityOptional.get();
                // Asegúrate de que el ID de la entidad a actualizar sea el mismo que el
del path
                entity.setId((Long) id);
                // Mantener la fecha de alta original y el estado si no se especifica lo
contrario en la entidad
                entity.setFechaAlta(entityExistente.getFechaAlta());
                if (!entityExistente.isEstado()) { // Si ya estaba dada de baja, no
permitir modificar si la lógica no lo permite
                    throw new Exception("No se puede modificar una entidad dada de baja
con ID: " + id);
                }
                entity.setEstado(entityExistente.isEstado()); // Mantener el estado
existente
                entity.setFechaModificacion(LocalDateTime.now()); // Actualizar la fecha
de modificación

                return baseRepository.save(entity);
            } else {
                throw new Exception("No se encontró la entidad con ID: " + id);
            }
        } catch (Exception e) {
            throw new Exception("Error al actualizar la entidad con ID: " + id + ": " +
e.getMessage(), e);
        }
    }

    @Override
    @Transactional
    public boolean delete(ID id) throws Exception {
        try {
            E entity = baseRepository.findById(id)
                    .orElseThrow(() -> new Exception("No se encontró la entidad con ID: "
+ id));
```

```
            if (entity.isEstado()) { // Solo si la entidad está activa
                entity.setEstado(false);
                entity.setFechaBaja(LocalDateTime.now());
                baseRepository.save(entity); // Guardar la entidad con la baja lógica
                return true;
            } else {
                throw new Exception("La entidad con ID: " + id + " ya se encuentra dada de
baja.");
            }
        } catch (Exception e) {
            throw new Exception("Error al dar de baja la entidad con ID: " + id + ": " +
e.getMessage(), e);
        }
    }
}
```

BaseController:

```
package com.example.InvOpBack.Controller;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import com.example.InvOpBack.Entities.Base;
import java.io.Serializable;


public interface BaseController <E extends Base, ID extends Serializable> {

    public ResponseEntity<?> getAll();

    public ResponseEntity<?> getOne(@PathVariable ID id);

    public ResponseEntity<?> save (@RequestBody E entity);

    public ResponseEntity<?> update(@PathVariable ID id, @RequestBody E entity);

    public ResponseEntity<?> delete(@PathVariable ID id);

}
```

BaseControllerImpl:

```
package com.example.InvOpBack.Controller;

import com.example.InvOpBack.Entities.Base;
import com.example.InvOpBack.Service.BaseService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Map;

public abstract class BaseControllerImpl <E extends Base, S extends BaseService<E, Long>>
implements BaseController<E, Long> {
    @Autowired
    protected S servicio;

    @GetMapping("")
    public ResponseEntity<?> getAll() {
        try {
```

```
            return ResponseEntity.status(HttpStatus.OK).body(servicio.findAll());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(Map.of("error",
"Error, por favor intente más tarde"));
        }
    }

    @GetMapping("/{id}")
    public ResponseEntity<?> getOne(@PathVariable Long id){
        try {
            return ResponseEntity.status(HttpStatus.OK).body(servicio.findById(id));
        } catch (Exception e){
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(Map.of("error",
"Error, por favor intente más tarde"));
        }
    }

    @PostMapping("")
    public ResponseEntity<?> save(@RequestBody E entity){
        try {
            return ResponseEntity.status(HttpStatus.OK).body(servicio.save(entity));
        } catch (Exception e){
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(Map.of("error",
"Error, por favor intente más tarde"));
        }
    }

    @PutMapping("/{id}")
    public ResponseEntity<?> update(@PathVariable Long id, @RequestBody E entity){
        try {
            return ResponseEntity.status(HttpStatus.OK).body(servicio.update(id,
entity));
        } catch (Exception e){
            e.printStackTrace();
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(Map.of("error",
"Error, por favor intente más tarde"));
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<?> delete(@PathVariable Long id){
        try {
            return
ResponseEntity.status(HttpStatus.NO_CONTENT).body(servicio.delete(id));
        } catch (Exception e){
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(Map.of("error",
"Error, por favor intente más tarde"));
        }
    }
}
```

Entities:

Articulo:

```
package com.example.InvOpBack.Entities;


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

```java
@Entity
@AllArgsConstructor
@NoArgsConstructor
@Setter
@Getter
@ToString
@Builder
@Table(name = "Articulo")
@JsonIgnoreProperties({"hibernateLazyInitializer", "proveedorPredeterminado"}) // Ignora el proxy
public class Articulo extends Base {
   private String nombreArticulo;
   private String descripcionArticulo;
   private float precioVentaArt;
   private float costoAlmacenamientoUnidad;
   private int stockActual;
   private int demandaArticulo;

   //Se cambió de LAZY por EAGER. Esto cargará el proveedor predeterminado junto con el artículo, evitando el problema de serialización.
   @ManyToOne(fetch = FetchType.EAGER)
   //@JsonIgnore
   @JoinColumn(name = "id_proveedor_predeterminado")
   private Proveedor proveedorPredeterminado;

}
```

ArticuloProveedor:

```java
package com.example.InvOpBack.Entities;

import jakarta.persistence.*;
import lombok.*;

import java.util.Date;

@Entity
@AllArgsConstructor
@NoArgsConstructor
@Setter
@Getter
@ToString
@Builder
@Table(name = "ArticuloProveedor")
public class ArticuloProveedor extends Base{
   private double costoCompra;
   private double costoPorPedido;
   private double costoPedido;
   private int demoraEntrega;
   private int desviacionEstandar;
   private double cantidadPedido;
   private float precioUnitario;
   private double puntoPedido;
   private double stockSeguridad;
   private double CGI;
   private int intervaloRevision;
   private double costoAlmacenamiento;
   private int loteOptimo;
   private int valorCGI;
   private int inventarioMaximo;


   @Enumerated(EnumType.STRING)
   private ModeloInventario modeloInventario;
```

```
    @ManyToOne (fetch = FetchType.LAZY)
    private Proveedor proveedor;

    @ManyToOne (fetch = FetchType.LAZY)
    private Articulo articulo;

    /*
    @OneToMany (fetch = FetchType.LAZY)
    private OrdenCompra ordenCompra;
    */
}
```

ArticuloVenta:

```
package com.example.InvOpBack.Entities;

import com.fasterxml.jackson.annotation.JsonBackReference;
import jakarta.persistence.*;
import lombok.*;

@Entity
@AllArgsConstructor
@NoArgsConstructor
@Setter
@Getter
@ToString
@Builder
@Table(name = "ArticuloVenta")
public class ArticuloVenta extends Base {
    private int cantArticuloVenta; // Cantidad de artículos vendidos.
    private float precioSubTotal; // Precio subtotal de la venta.
    private float precioUnitario; // Precio unitario del artículo.

    // Relación Many-to-One con Articulo. Un artículo puede estar en muchas líneas de
venta.
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "id_articulo") // Clave foránea para Articulo
    private Articulo articulo;

    // Relación Many-to-One con Venta. Una línea de artículo pertenece a una única venta.
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "id_venta")
    @JsonBackReference // Indica que esta es la parte "hija" de la relación
    private Venta venta;
}
```

EstadoOrdenCompra:

```
package com.example.InvOpBack.Entities;

public enum EstadoOrdenCompra {
    Cancelada,
    Pendiente,
    Enviada,
    Finalizada;

}
```

ModeloInventario:

```
package com.example.InvOpBack.Entities;

public enum ModeloInventario {
   loteFijo,
   intervaloFijo;

}
```

OrdenCompra:

```
package com.example.InvOpBack.Entities;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonManagedReference;
import jakarta.persistence.*;
import lombok.*;

import java.util.ArrayList;
import java.util.List;

@Entity
@AllArgsConstructor
@NoArgsConstructor
@Setter
@Getter
@ToString
@Builder
@Table(name = "OrdenCompra")
@JsonIgnoreProperties({"hibernateLazyInitializer", "proveedor"}) // Ignora el proxy
public class OrdenCompra extends Base {
   private float totalOrdenCompra;
   private int cantidadOrdenCompra;
   @Enumerated(EnumType.STRING)
   private EstadoOrdenCompra estadoOrdenCompra;
   @ManyToOne(fetch = FetchType.LAZY)
   private Proveedor proveedor;
   @OneToMany(mappedBy = "ordenCompra", cascade = CascadeType.ALL, orphanRemoval = true)
   @Builder.Default
   private List<OrdenCompraArticulo> ordenCompraArticulo = new ArrayList<>();
}
```

OrdenCompraArticulo:

```
package com.example.InvOpBack.Entities;
import com.fasterxml.jackson.annotation.JsonBackReference;
import jakarta.persistence.*;
import lombok.*;

@Entity
@AllArgsConstructor
@NoArgsConstructor
@Setter
@Getter
@ToString
@Builder
@Table(name = "OrdenCompraArticulo")
public class OrdenCompraArticulo extends Base{

   private int cantOCA;
   private float precioSubTotalOCA;
```

```
    private float precioUnitarioOCA;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "id_orden_compra")
    @JsonBackReference // Indica que esta es la parte "hija" de la relación
    private OrdenCompra ordenCompra;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "id_articulo")
    private Articulo articulo;

}
```

Proveedor:

```
package com.example.InvOpBack.Entities;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import jakarta.persistence.CascadeType;
import jakarta.persistence.Entity;
import jakarta.persistence.OneToMany;
import jakarta.persistence.Table;
import lombok.*;

import java.util.ArrayList;
import java.util.List;

@Entity
@AllArgsConstructor
@NoArgsConstructor
@Setter
@Getter
@ToString
@Builder
@Table(name = "Proveedor")
@JsonIgnoreProperties({"hibernateLazyInitializer", "ordenesCompra"}) // Ignora el proxy y
la lista de órdenes
public class Proveedor extends Base {
    private String nombreProveedor;
    private String cuit;
    @OneToMany(mappedBy = "proveedor", cascade = CascadeType.ALL, orphanRemoval = true)
    @Builder.Default
    private List<OrdenCompra> ordenesCompra = new ArrayList<>();
}
```

Venta:

```
package com.example.InvOpBack.Entities;


import com.fasterxml.jackson.annotation.JsonManagedReference;
import jakarta.persistence.*;
import jakarta.persistence.Table;
import lombok.*;

import java.util.ArrayList;
import java.time.LocalDateTime; // Usar LocalDateTime para fechas modernas
import java.util.List;

@Entity
@AllArgsConstructor
@NoArgsConstructor
@Setter
```

```
@Getter
@ToString
@Builder
@Table(name = "Venta")
public class Venta extends Base {

   private LocalDateTime fechaVenta; // Fecha y hora de la venta.
   private float costoTotal; // Costo total de la venta.

   // Relación One-to-Many con ArticuloVenta. Una venta puede tener muchos artículos.
   @OneToMany(mappedBy = "venta", cascade = CascadeType.ALL, orphanRemoval = true, fetch
= FetchType.LAZY)
   @Builder.Default
   @JsonManagedReference // Indica que esta es la parte "padre" de la relación
   private List<ArticuloVenta> articuloVenta = new ArrayList<>();

}
```

Repository:

ArticuloProveedorRepository:

```
package com.example.InvOpBack.Repository;

import com.example.InvOpBack.Entities.Articulo;
import com.example.InvOpBack.Entities.ArticuloProveedor;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;


import java.util.List;

@Repository
public interface ArticuloProveedorRepository extends BaseRepository<ArticuloProveedor,
Long> {

   // Encuentra todas las relaciones ArticuloProveedor por el ID del artículo.
   List<ArticuloProveedor> findByArticuloId(Long articuloId);

   // Encuentra todas las relaciones ArticuloProveedor por el ID del proveedor.
   List<ArticuloProveedor> findByProveedorId(Long proveedorId);

   // Encuentra una relación ArticuloProveedor específica por el ID del proveedor y el ID
del artículo.
   // Usamos una query JPQL para un JOIN explícito, aunque Spring Data JPA podría
inferirlo.
   @Query("SELECT ap FROM ArticuloProveedor ap WHERE ap.proveedor.id = :proveedorId AND
ap.articulo.id = :articuloId")
   ArticuloProveedor findByProveedorIdAndArticuloId(@Param("proveedorId") Long
proveedorId, @Param("articuloId") Long articuloId);

   @Query("SELECT CASE WHEN COUNT(ap) = 0 THEN true ELSE false END " +
           "FROM ArticuloProveedor ap " +
           "WHERE ap.articulo.id = :articuloId AND ap.proveedor.id = :proveedorId")
   boolean noExisteArticuloProveedor(@Param("articuloId") Long articuloId,
@Param("proveedorId") Long proveedorId);

}
```

ArticuloRepository:

```java
package com.example.InvOpBack.Repository;

import com.example.InvOpBack.Entities.Articulo;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.Optional;

@Repository
public interface ArticuloRepository extends BaseRepository<Articulo,Long> {
    // Metodo para verificar si un artículo ya existe por nombre (para unicidad).
    Optional<Articulo> findByNombreArticulo(String nombreArticulo);

    // Metodo para verificar si un proveedor es predeterminado para algún artículo.
    boolean existsByProveedorPredeterminadoId(Long proveedorId);

    List<Articulo> findByEstadoTrue();

    Optional<List<Articulo>> findByProveedorPredeterminadoId(Long proveedorId);
}
```

ArticuloVentaReposirtoy:

```java
package com.example.InvOpBack.Repository;



import com.example.InvOpBack.Entities.ArticuloVenta;

import org.springframework.stereotype.Repository;



// Repositorio para la entidad ArticuloVenta.

@Repository

public interface ArticuloVentaRepository extends BaseRepository<ArticuloVenta, Long> {

    // Puedes añadir métodos de consulta personalizados si los necesitas,

    // como encontrar todas las líneas de venta para una venta específica.

    // List<ArticuloVenta> findByVentaId(Long ventaId);

}
```

OrdenCompraRepository:

```java
package com.example.InvOpBack.Repository;

import com.example.InvOpBack.Entities.EstadoOrdenCompra;
import com.example.InvOpBack.Entities.OrdenCompra;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;


@Repository
```

```java
public interface OrdenCompraRepository extends BaseRepository <OrdenCompra,Long> {

   // Metodo para buscar órdenes de compra por el ID del proveedor.
   List<OrdenCompra> findByProveedorId(Long proveedorId);

   // Metodo para buscar órdenes de compra por estado.
   List<OrdenCompra> findByEstadoOrdenCompra(EstadoOrdenCompra estado);

   // Metodo para buscar si existe alguna Orden de Compra en estado PENDIENTE o ENVIADA
para un artículo específico.
   // Esta consulta es crucial para la lógica de "no permitir venta si hay OC activa" o
"no generar OC si ya existe".
   @Query("SELECT oc FROM OrdenCompra oc JOIN oc.ordenCompraArticulo oca WHERE
oca.articulo.id = :articuloId AND oc.estadoOrdenCompra IN :estados")
   List<OrdenCompra> findActiveOrdersForArticulo(@Param("articuloId") Long articuloId,
@Param("estados") List<EstadoOrdenCompra> estados);

   @Query("SELECT CASE WHEN COUNT(oc) > 0 THEN true ELSE false END " +
           "FROM OrdenCompra oc " +
           "WHERE oc.proveedor.id = :proveedorId " +
           "AND oc.estadoOrdenCompra IN :estados")
   boolean existeOrdenCompraEnEstados(@Param("proveedorId") Long proveedorId,
                                      @Param("estados") List<EstadoOrdenCompra> estados);

}
```

ProveedorRepository:

```java
package com.example.InvOpBack.Repository;

import com.example.InvOpBack.Entities.Proveedor;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.Optional;

@Repository
public interface ProveedorRepository extends BaseRepository<Proveedor,Long> {
   Optional<Proveedor> findByCuit(String Cuit);
   @Query ("SELECT CASE WHEN COUNT(oc) > 0 THEN true ELSE false END FROM OrdenCompra oc
WHERE oc.proveedor.id = :proveedorId AND (oc.estadoOrdenCompra = 'Pendiente' OR
oc.estadoOrdenCompra = 'Enviada')")
   boolean hasActiveOrders(@Param("proveedorId") Long proveedorId);
   // Nuevo metodo para listar sólo proveedores activos
   List<Proveedor> findByEstadoTrue();
}
```

VentaRepository:

```java
package com.example.InvOpBack.Repository;



import com.example.InvOpBack.Entities.Venta;



public interface VentaRepository extends BaseRepository<Venta, Long> {

   // Puedes añadir métodos de consulta personalizados si los necesitas,

   // como encontrar todas las líneas de venta para una venta específica.
```

```
    // List<Venta> findByVentaId(Long ventaId);



}
```

OrdenCompraArticuloRepository:

```
package com.example.InvOpBack.Repository;

import com.example.InvOpBack.Entities.OrdenCompraArticulo;
import org.springframework.stereotype.Repository;

import java.util.List;

// Repositorio para la entidad OrdenCompraArticulo.
@Repository
public interface OrdenCompraArticuloRepository extends BaseRepository<OrdenCompraArticulo, Long> {
    // Puedes añadir métodos de consulta personalizados si los necesitas,
    // como encontrar todas las líneas de artículos para una orden de compra específica.
    List<OrdenCompraArticulo> findByOrdenCompraId(Long ordenCompraId);

}
```

Service:

ArticuloProveedorService:

```
package com.example.InvOpBack.Service;


import com.example.InvOpBack.DTOs.ArticuloProveedorDTO;
import com.example.InvOpBack.DTOs.ListadoDTO;
import com.example.InvOpBack.Entities.ArticuloProveedor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public interface ArticuloProveedorService extends BaseService<ArticuloProveedor,Long>{
    ArticuloProveedor altaAP(ArticuloProveedorDTO articuloProveedorDTO) throws Exception;
    ArticuloProveedor modificarAP(Long id, ArticuloProveedorDTO articuloProveedorDTO)
throws Exception;
    List<ListadoDTO> listadoArticulosPorProveedor(Long proveedorId) throws Exception;
    List<ArticuloProveedor> findByArticuloId(Long articuloId) throws Exception;
    List<ArticuloProveedor> findByProveedorId(Long proveedorId); // Cambiado a List
    ArticuloProveedor findByProveedorIdAndArticuloId(Long proveedorId, Long articuloId);
}
```

ArticuloProveedorServiceImpl:

```
package com.example.InvOpBack.Service;

import com.example.InvOpBack.DTOs.ArticuloProveedorDTO;
import com.example.InvOpBack.DTOs.ListadoDTO;
import com.example.InvOpBack.Entities.Articulo;
import com.example.InvOpBack.Entities.ArticuloProveedor;
import com.example.InvOpBack.Entities.ModeloInventario;
import com.example.InvOpBack.Entities.Proveedor;
import com.example.InvOpBack.Repository.ArticuloProveedorRepository;
import com.example.InvOpBack.Repository.ArticuloRepository;
import com.example.InvOpBack.Repository.ProveedorRepository;
```

```java
import jakarta.transaction.Transactional;

import lombok.extern.java.Log;
import org.springframework.stereotype.Service;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Service
public class ArticuloProveedorServiceImpl extends BaseServiceImpl<ArticuloProveedor,Long>
implements ArticuloProveedorService, Serializable {


   private ArticuloProveedorRepository articuloProveedorRepository;

   private ArticuloRepository articuloRepository;

   private ProveedorRepository proveedorRepository;


   public ArticuloProveedorServiceImpl(ArticuloProveedorRepository
articuloProveedorRepository,
                                       ArticuloRepository articuloRepository,
                                       ProveedorRepository proveedorRepository) {
       super(articuloProveedorRepository);
       this.articuloProveedorRepository = articuloProveedorRepository;
       this.articuloRepository = articuloRepository;
       this.proveedorRepository = proveedorRepository;
   }

   @Override
   @Transactional
   public ArticuloProveedor altaAP(ArticuloProveedorDTO articuloProveedorDTO) throws
Exception {
       ArticuloProveedor articuloProveedor = new ArticuloProveedor();
       Articulo articulo =
articuloRepository.findById(articuloProveedorDTO.getId_articulo()).orElseThrow(() -> new
Exception("Artículo no encontrado"));
       if (articulo == null) throw new Exception("Artículo no encontrado");
       Proveedor proveedor =
proveedorRepository.findById(articuloProveedorDTO.getId_proveedor()).orElseThrow(() ->
new Exception("Proveedor no encontrado"));
       if (proveedor == null) throw new Exception("Proveedor no encontrado");
       articuloProveedor.setArticulo(articulo);
       articuloProveedor.setProveedor(proveedor);
       articuloProveedor.setPrecioUnitario(articuloProveedorDTO.getPrecioUnitario());
       articuloProveedor.setCostoPedido(articuloProveedorDTO.getCostoPedido());
       articuloProveedor.setDemoraEntrega(articuloProveedorDTO.getDemoraEntrega());
       articuloProveedor.setCostoCompra(articuloProveedorDTO.getCostoCompra());
       articuloProveedor.setModeloInventario(articuloProveedorDTO.getModeloInventario());
       articuloProveedorRepository.save(articuloProveedor);
       return articuloProveedor;
   }

   @Override
   @Transactional
   public List<ArticuloProveedor> findByArticuloId(Long articuloId) throws Exception {
       try {
           return articuloProveedorRepository.findByArticuloId(articuloId);
       } catch (Exception e){
           throw new Exception(e.getMessage());
       }
   }
```

```
   public List<ArticuloProveedor> findByProveedorId(Long proveedorId) {
       List<ArticuloProveedor> buscarArticulos =
articuloProveedorRepository.findByProveedorId(proveedorId);
       return buscarArticulos;
   }


   public ArticuloProveedor findByProveedorIdAndArticuloId(Long proveedorId, Long
articuloId){
       return
articuloProveedorRepository.findByProveedorIdAndArticuloId(proveedorId,articuloId);
   }

   //Cálculo StockSeguridad
   private double calculoStockSeguridad(Long articuloProveedorId) throws Exception {
           // Validar que el artículo exista y esté activo antes de buscar sus
proveedores.
       Optional<ArticuloProveedor> articuloProveedorOpt =
articuloProveedorRepository.findById(articuloProveedorId);
       if (!articuloProveedorOpt.isPresent() || !articuloProveedorOpt.get().isEstado()) {
           throw new Exception("ArtículoProveedor no encontrado o inactivo con ID: " +
articuloProveedorId);
       }
       ArticuloProveedor articuloProveedor = articuloProveedorOpt.get();

       double stockSeguridad=0;
       if (articuloProveedor.getModeloInventario() == ModeloInventario.loteFijo) {
           stockSeguridad = 1.64 * articuloProveedor.getDesviacionEstandar() *
Math.sqrt(articuloProveedor.getDemoraEntrega());
       } else {
           stockSeguridad = 1.64 * articuloProveedor.getDesviacionEstandar() *
Math.sqrt(articuloProveedor.getDemoraEntrega() +
articuloProveedor.getIntervaloRevision());
       }
       return stockSeguridad;
   }

   // Cálculo de cantidadPedido
   private double calcularCantidadPedido(Long articuloProveedorId) throws Exception {
       // Validar que el artículo exista y esté activo antes de buscar sus proveedores.
       Optional<ArticuloProveedor> articuloProveedorOpt =
articuloProveedorRepository.findById(articuloProveedorId);
       if (!articuloProveedorOpt.isPresent() || !articuloProveedorOpt.get().isEstado()) {
           throw new Exception("ArtículoProveedor no encontrado o inactivo con ID: " +
articuloProveedorId);
       }
       ArticuloProveedor articuloProveedor = articuloProveedorOpt.get();

       double cantidadPedido=0;
       if (articuloProveedor.getModeloInventario() == ModeloInventario.loteFijo) {
           cantidadPedido =
Math.sqrt((2*articuloProveedor.getArticulo().getDemandaArticulo()*articuloProveedor.getCo
stoPorPedido())/(articuloProveedor.getArticulo().getCostoAlmacenamientoUnidad()));
       } else {
           cantidadPedido =
(((articuloProveedor.getArticulo().getDemandaArticulo()/365)*(articuloProveedor.getInterv
aloRevision()+articuloProveedor.getDemoraEntrega()))+articuloProveedor.getStockSeguridad(
)-articuloProveedor.getArticulo().getStockActual());
       }
       return cantidadPedido;
   }

   // Cálculo del Punto de Pedido
   private double calcularPuntoPedido(Long articuloProveedorId) throws Exception {
```

```
        Optional<ArticuloProveedor> articuloProveedorOpt =
articuloProveedorRepository.findById(articuloProveedorId);
        if (!articuloProveedorOpt.isPresent() || !articuloProveedorOpt.get().isEstado()) {
            throw new Exception("ArtículoProveedor no encontrado o inactivo con ID: " +
articuloProveedorId);
        }
        ArticuloProveedor articuloProveedor = articuloProveedorOpt.get();
        double
puntoPedido=((articuloProveedor.getArticulo().getDemandaArticulo()/365)*articuloProveedor
.getDemoraEntrega())+articuloProveedor.getStockSeguridad();
        return puntoPedido;
    }

    //Calculo costoCompra
    private double calcularCostoCompra(Long articuloProveedorId) throws Exception {
        Optional<ArticuloProveedor> articuloProveedorOpt =
articuloProveedorRepository.findById(articuloProveedorId);
        if (!articuloProveedorOpt.isPresent() || !articuloProveedorOpt.get().isEstado()) {
            throw new Exception("ArtículoProveedor no encontrado o inactivo con ID: " +
articuloProveedorId);
        }
        ArticuloProveedor articuloProveedor = articuloProveedorOpt.get();
        double
costoCompra=articuloProveedor.getArticulo().getDemandaArticulo()*articuloProveedor.getPre
cioUnitario();
        return costoCompra;
    }

    //Calculo costoPedido
    private double calcularCostoPedido(Long articuloProveedorId) throws Exception {
        Optional<ArticuloProveedor> articuloProveedorOpt =
articuloProveedorRepository.findById(articuloProveedorId);
        if (!articuloProveedorOpt.isPresent() || !articuloProveedorOpt.get().isEstado()) {
            throw new Exception("ArtículoProveedor no encontrado o inactivo con ID: " +
articuloProveedorId);
        }
        ArticuloProveedor articuloProveedor = articuloProveedorOpt.get();
        double
costoPedido=(articuloProveedor.getArticulo().getDemandaArticulo()/articuloProveedor.getCa
ntidadPedido())*articuloProveedor.getCostoPorPedido();
        return costoPedido;
    }

    //Calculo costoAlmacenamiento
    private double calcularCostoAlmacenamiento(Long articuloProveedorId) throws Exception
{
        Optional<ArticuloProveedor> articuloProveedorOpt =
articuloProveedorRepository.findById(articuloProveedorId);
        if (!articuloProveedorOpt.isPresent() || !articuloProveedorOpt.get().isEstado()) {
            throw new Exception("ArtículoProveedor no encontrado o inactivo con ID: " +
articuloProveedorId);
        }
        ArticuloProveedor articuloProveedor = articuloProveedorOpt.get();
        double
costoAlmacenamiento=(articuloProveedor.getCantidadPedido()/2)*articuloProveedor.getArticu
lo().getCostoAlmacenamientoUnidad();
        return costoAlmacenamiento;
    }

    //Calculo CGI
    private double calcularCGI(Long articuloProveedorId) throws Exception {
        Optional<ArticuloProveedor> articuloProveedorOpt =
articuloProveedorRepository.findById(articuloProveedorId);
        if (!articuloProveedorOpt.isPresent() || !articuloProveedorOpt.get().isEstado()) {
```

```
            throw new Exception("ArtículoProveedor no encontrado o inactivo con ID: " +
articuloProveedorId);
        }
        ArticuloProveedor articuloProveedor = articuloProveedorOpt.get();
        double
CGI=articuloProveedor.getCostoPedido()+articuloProveedor.getCostoCompra()+articuloProveed
or.getCostoAlmacenamiento();
        return CGI;
    }

    public void calcularTodo(Long articuloProveedorId) throws Exception {
        Optional<ArticuloProveedor> articuloProveedorOpt =
articuloProveedorRepository.findById(articuloProveedorId);
        if (!articuloProveedorOpt.isPresent() || !articuloProveedorOpt.get().isEstado()) {
            throw new Exception("ArtículoProveedor no encontrado o inactivo con ID: " +
articuloProveedorId);
        }
        ArticuloProveedor articuloProveedor = articuloProveedorOpt.get();
        //Cálculo Stock seguridad
        articuloProveedor.setStockSeguridad(calculoStockSeguridad(articuloProveedorId));
        //Cálculo cantidad a pedir
        articuloProveedor.setCantidadPedido(calcularCantidadPedido(articuloProveedorId));
        if (articuloProveedor.getModeloInventario() == ModeloInventario.loteFijo) {
            articuloProveedor.setPuntoPedido(calcularPuntoPedido(articuloProveedorId));
        }
        //Cálculo costoCompra
        articuloProveedor.setCostoCompra(calcularCostoCompra(articuloProveedorId));
        //Cálculo costoPedido
        articuloProveedor.setCostoPedido(calcularCantidadPedido(articuloProveedorId));
        //Cálculo costoAlmacenamiento

articuloProveedor.setCostoAlmacenamiento(calcularCostoAlmacenamiento(articuloProveedorId)
);
        //Cálculo CGI
        articuloProveedor.setCGI(calcularCGI(articuloProveedorId));
    }

    @Override
    public ArticuloProveedor modificarAP(Long id, ArticuloProveedorDTO
articuloProveedorDTO) throws Exception {
        return null;
    }

    @Override
    public List<ListadoDTO> listadoArticulosPorProveedor(Long proveedorId) throws
Exception {

        return List.of();
    }
}
```

ArticuloService:

```
package com.example.InvOpBack.Service;

import com.example.InvOpBack.DTOs.ArticuloDTO;
import com.example.InvOpBack.DTOs.ProveedorPredeterminadoDTO;
import com.example.InvOpBack.Entities.Articulo;
import jakarta.transaction.Transactional;
import org.springframework.stereotype.Service;

@Service
public interface ArticuloService extends BaseService<Articulo,Long>{
    Articulo altaArticulo(ArticuloDTO dto) throws Exception;
```

```
    Articulo modificarArticulo(Long id, ArticuloDTO dto) throws Exception;
    Articulo establecerProveedor(Long id, ProveedorPredeterminadoDTO dto) throws
Exception;
}
```

ArticuloServiceImpl:

```
package com.example.InvOpBack.Service;

import com.example.InvOpBack.DTOs.ArticuloDTO;
import com.example.InvOpBack.DTOs.ProveedorPredeterminadoDTO;
import com.example.InvOpBack.Entities.*;
import com.example.InvOpBack.Repository.ArticuloRepository;
import com.example.InvOpBack.Repository.OrdenCompraRepository;
import com.example.InvOpBack.Repository.ProveedorRepository;
import jakarta.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.io.Serializable;
import java.time.LocalDateTime;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
public class ArticuloServiceImpl extends BaseServiceImpl<Articulo,Long> implements
ArticuloService, Serializable {

    private ArticuloRepository articuloRepository;
    private ProveedorRepository proveedorRepository;
    private OrdenCompraRepository ordenCompraRepository; // Inyectar para validaciones de
baja.

    @Autowired
    public ArticuloServiceImpl(ArticuloRepository articuloRepository,
                               ProveedorRepository proveedorRepository,
                               OrdenCompraRepository ordenCompraRepository) {
        super(articuloRepository);
        this.articuloRepository = articuloRepository;
        this.proveedorRepository = proveedorRepository;
        this.ordenCompraRepository = ordenCompraRepository;
    }

    @Override
    @Transactional
    public Articulo altaArticulo(ArticuloDTO articuloDTO) throws Exception {
        Optional<Articulo> existingArticulo =
articuloRepository.findByNombreArticulo(articuloDTO.getNombreArticulo());
        if (existingArticulo.isPresent()) {
            throw new Exception("Ya existe un artículo con el mismo nombre.");
        }
        Articulo articulo = new Articulo();
        articulo.setNombreArticulo(articuloDTO.getNombreArticulo());
        articulo.setDescripcionArticulo(articuloDTO.getDescripcionArticulo());
        articulo.setPrecioVentaArt(articuloDTO.getPrecioVentaArt() != null ?
articuloDTO.getPrecioVentaArt() : 0.0f);
        articulo.setCostoAlmacenamientoUnidad(articuloDTO.getCostoAlmacenamiento() != null
? articuloDTO.getCostoAlmacenamiento() : 0.0f);
        articulo.setDemandaArticulo(articuloDTO.getDemandaArticulo() != null ?
articuloDTO.getDemandaArticulo() : 0);
        articulo.setStockActual(articuloDTO.getStockActual() != null ?
articuloDTO.getStockActual() : 0);
```

```
        // Manejar proveedor predeterminado
        if (articuloDTO.getIdProveedorPredeterminado() != null) {
            Proveedor proveedor =
proveedorRepository.findById(articuloDTO.getIdProveedorPredeterminado())
                    .orElseThrow(() -> new Exception("Proveedor predeterminado no
encontrado con ID: " + articuloDTO.getIdProveedorPredeterminado()));
            if (!proveedor.isEstado()) {
                throw new Exception("El proveedor predeterminado está inactivo y no puede
ser asignado.");
            }
            articulo.setProveedorPredeterminado(proveedor);
        } else {
            articulo.setProveedorPredeterminado(null);
        }
        return super.save(articulo);
    }

    @Override
    @Transactional
    public Articulo modificarArticulo(Long id, ArticuloDTO articuloDTO) throws Exception {
        Articulo articuloExistente = articuloRepository.findById(id)
                .orElseThrow(() -> new Exception("Artículo no encontrado con ID: " + id));
        Optional<Articulo> existingArticulo =
articuloRepository.findByNombreArticulo(articuloDTO.getNombreArticulo());
        if (existingArticulo.isPresent() && !existingArticulo.get().getId().equals(id)) {
            throw new Exception("Ya existe otro artículo con el mismo nombre y
descripción.");
        }

        // Actualizar solo los campos que no son nulos en el DTO.
        if (articuloDTO.getNombreArticulo() != null) {
            articuloExistente.setNombreArticulo(articuloDTO.getNombreArticulo());
        }
        if (articuloDTO.getDescripcionArticulo() != null) {

articuloExistente.setDescripcionArticulo(articuloDTO.getDescripcionArticulo());
        }
        if (articuloDTO.getPrecioVentaArt() != null) {
            articuloExistente.setPrecioVentaArt(articuloDTO.getPrecioVentaArt());
        }
        if (articuloDTO.getCostoAlmacenamiento() != null) {

articuloExistente.setCostoAlmacenamientoUnidad(articuloDTO.getCostoAlmacenamiento());
        }
        if (articuloDTO.getDemandaArticulo() != null) {
            articuloExistente.setDemandaArticulo(articuloDTO.getDemandaArticulo());
        }
        if (articuloDTO.getStockActual() != null) {
            articuloExistente.setStockActual(articuloDTO.getStockActual());
        }

        // Habilitar proveedor predeterminado en modificación
        if (articuloDTO.getIdProveedorPredeterminado() != null) {
            Proveedor proveedor =
proveedorRepository.findById(articuloDTO.getIdProveedorPredeterminado())
                    .orElseThrow(() -> new Exception("Proveedor predeterminado no
encontrado con ID: " + articuloDTO.getIdProveedorPredeterminado()));
            if (!proveedor.isEstado()) {
                throw new Exception("El proveedor predeterminado está inactivo y no puede
ser asignado.");
            }
            articuloExistente.setProveedorPredeterminado(proveedor);
        } else {
            articuloExistente.setProveedorPredeterminado(null);
```

```
        }

        return super.update(id, articuloExistente); // Usar el update de BaseServiceImpl.
    }

    @Transactional
    @Override
    public Articulo establecerProveedor(Long id, ProveedorPredeterminadoDTO dto) throws
Exception {
        Articulo articulo = articuloRepository.findById(id)
                .orElseThrow(() -> new Exception("Artículo no encontrado con ID: " + id));
        if (!articulo.isEstado()) {
            throw new Exception("El artículo con ID: " + id + " está inactivo y no se le
puede establecer un proveedor predeterminado.");
        }
        if (dto.getIdProveedor() != null) {
            Proveedor proveedor = proveedorRepository.findById(dto.getIdProveedor())
                    .orElseThrow(() -> new Exception("Proveedor no encontrado con ID: " +
dto.getIdProveedor()));
            if (!proveedor.isEstado()) {
                throw new Exception("El proveedor con ID: " + dto.getIdProveedor() + "
está inactivo y no puede ser establecido como predeterminado.");
            }
            articulo.setProveedorPredeterminado(proveedor);
        } else {
            articulo.setProveedorPredeterminado(null);
        }
        articulo.setFechaModificacion(LocalDateTime.now());
        return articuloRepository.save(articulo);
    }

    @Override
    @Transactional
    public boolean delete(Long id) throws Exception {
        Articulo articulo = articuloRepository.findById(id)
                .orElseThrow(() -> new Exception("Artículo no encontrado con ID: " + id));
        if (!articulo.isEstado()) {
            throw new Exception("El artículo con ID: " + id + " ya está inactivo.");
        }

        // *** VALIDACIONES PARA BAJA LÓGICA DE ARTÍCULO ***
        // 1. No permitir la baja si el artículo tiene órdenes de compra pendientes o
enviadas.
        List<EstadoOrdenCompra> estadosActivosOC =
Arrays.asList(EstadoOrdenCompra.Pendiente, EstadoOrdenCompra.Enviada);
        List<OrdenCompra> activeOrders =
ordenCompraRepository.findActiveOrdersForArticulo(articulo.getId(), estadosActivosOC);
        if (!activeOrders.isEmpty()) {
            throw new Exception("No se puede dar de baja el artículo porque tiene órdenes
de compra pendientes o enviadas.");
        }

        // 2. No permitir la baja si el artículo tiene unidades en stock.
        if (articulo.getStockActual() > 0) {
            throw new Exception("No se puede dar de baja el artículo porque tiene unidades
en stock (" + articulo.getStockActual() + ").");
        }

        // Si pasa las validaciones, procede con la baja lógica a través de
BaseServiceImpl.
        return super.delete(id);
    }

    @Override
    @Transactional
```

```
    public List<Articulo> findAll() throws Exception {
        // Listar solo artículos activos
        return articuloRepository.findByEstadoTrue();
    }

}
```

ProveedorService:

```
package com.example.InvOpBack.Service;

import com.example.InvOpBack.DTOs.ProveedorDTO;
import com.example.InvOpBack.Entities.Proveedor;
import org.springframework.stereotype.Service;

@Service
public interface ProveedorService extends BaseService<Proveedor, Long> {
    Proveedor altaProveedor(ProveedorDTO dto) throws Exception;
    //Proveedor modificarProveedor(Long id, ProveedorDTO dto) throws Exception; //
Modificada la firma para incluir ID
    // Metodo para la baja lógica con validaciones específicas.
    boolean bajaProveedor(Long id) throws Exception;

}
```

ProveedorServiceImpl:

```
package com.example.InvOpBack.Service;


import com.example.InvOpBack.DTOs.ArticuloProveedorDTO;
import com.example.InvOpBack.DTOs.ProveedorDTO;
import com.example.InvOpBack.Entities.Articulo;
import com.example.InvOpBack.Entities.ArticuloProveedor;
import com.example.InvOpBack.Entities.EstadoOrdenCompra;
import com.example.InvOpBack.Entities.OrdenCompra;
import com.example.InvOpBack.Entities.Proveedor;
import com.example.InvOpBack.Repository.ArticuloProveedorRepository;
import com.example.InvOpBack.Repository.ArticuloRepository;
import com.example.InvOpBack.Repository.OrdenCompraRepository;
import com.example.InvOpBack.Repository.ProveedorRepository;
import jakarta.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;

@Service
public class ProveedorServiceImpl extends BaseServiceImpl<Proveedor, Long> implements
ProveedorService {

    private ProveedorRepository proveedorRepository;
    private ArticuloRepository articuloRepository;
    private ArticuloProveedorRepository articuloProveedorRepository;
    private OrdenCompraRepository ordenCompraRepository;

    @Autowired
    public ProveedorServiceImpl(ProveedorRepository proveedorRepository,
                                ArticuloRepository articuloRepository,
                                ArticuloProveedorRepository articuloProveedorRepository,
                                OrdenCompraRepository ordenCompraRepository) {
        super(proveedorRepository);
```

```
        this.proveedorRepository = proveedorRepository;
        this.articuloRepository = articuloRepository;
        this.articuloProveedorRepository = articuloProveedorRepository;
        this.ordenCompraRepository = ordenCompraRepository;
    }

    @Override
    @Transactional
    public Proveedor altaProveedor(ProveedorDTO proveedorDTO) throws Exception {
        if (proveedorDTO.getNombreProveedor() == null ||
proveedorDTO.getNombreProveedor().isEmpty()) {
            throw new Exception("El nombre del proveedor es obligatorio.");
        }
        if (proveedorDTO.getCuit() == null || proveedorDTO.getCuit().isEmpty()) {
            throw new Exception("El CUIT del proveedor es obligatorio.");
        }

        // Validación 1: Verificar que el CUIT no esté duplicado
        if (proveedorRepository.findByCuit(proveedorDTO.getCuit()).isPresent()) {
            throw new Exception("Ya existe un proveedor con el CUIT: " +
proveedorDTO.getCuit());
        }

        // Validación 2: Obligar a que se asocien artículos
        if (proveedorDTO.getArticulosAsociados() == null ||
proveedorDTO.getArticulosAsociados().isEmpty()) {
            throw new Exception("Es obligatorio asociar al menos un artículo al crear un
proveedor.");
        }

        Proveedor proveedor = new Proveedor();
        proveedor.setNombreProveedor(proveedorDTO.getNombreProveedor());
        proveedor.setCuit(proveedorDTO.getCuit());

        Proveedor proveedorGuardado = super.save(proveedor);

        // Asociar artículos si se proporcionan en el DTO.
        for (var rel : proveedorDTO.getArticulosAsociados()) {
            Articulo articulo = articuloRepository.findById(rel.getId_articulo())
                    .orElseThrow(() -> new Exception("El artículo con ID " +
rel.getId_articulo() + " no existe. El proveedor no es dado de alta."));
            if
(articuloProveedorRepository.findByProveedorIdAndArticuloId(proveedorGuardado.getId(),
articulo.getId()) != null) {
                continue;
            }
            // Crear la relación ArticuloProveedor
            ArticuloProveedor ap = new ArticuloProveedor();
            ap.setArticulo(articulo);
            ap.setProveedor(proveedorGuardado);
            ap.setCostoCompra(rel.getCostoCompra());
            ap.setCostoPedido(rel.getCostoPedido());
            ap.setDemoraEntrega(rel.getDemoraEntrega());
            ap.setPrecioUnitario(rel.getPrecioUnitario());
            ap.setModeloInventario(rel.getModeloInventario());
            ap.setPuntoPedido(rel.getPuntoPedido());
            ap.setStockSeguridad(rel.getStockSeguridad());
            ap.setInventarioMaximo(rel.getInventarioMaximo());
            ap.setValorCGI(rel.getValorCGI());
            articuloProveedorRepository.save(ap);
        }
        return proveedorGuardado;
    }
```

```
    // PARA GIACO Y ROBER HABRÍA QUE SACARLO PORQUE EL MVP NO LO PIDE Y COMPLICA UN
TOQUESINHO LA LÓGICA

    // Modificar Proveedor simplificado para MVP
    @Transactional
    public Proveedor modificarProveedor(Long id, ProveedorDTO proveedorDTO) throws
Exception {
        Proveedor proveedorExistente = proveedorRepository.findById(id)
                .orElseThrow(() -> new Exception("Proveedor no encontrado con ID: " +
id));
        if (proveedorDTO.getNombreProveedor() != null) {
            proveedorExistente.setNombreProveedor(proveedorDTO.getNombreProveedor());
        }
        if (proveedorDTO.getCuit() != null) {
            proveedorExistente.setCuit(proveedorDTO.getCuit());
        }
        proveedorExistente.setFechaModificacion(LocalDateTime.now());
        return super.update(id, proveedorExistente);
    }

    @Override
    @Transactional
    public boolean delete(Long id) throws Exception {
        return bajaProveedor(id);
    }

    @Override
    @Transactional
    public boolean bajaProveedor(Long id) throws Exception {
        Proveedor proveedor = proveedorRepository.findById(id)
                .orElseThrow(() -> new Exception("Proveedor no encontrado con ID: " +
id));
        if (articuloRepository.existsByProveedorPredeterminadoId(id)) {
            throw new Exception("No se puede dar de baja el proveedor porque es el
proveedor predeterminado de uno o más artículos activos.");
        }
        List<OrdenCompra> activeOrders = ordenCompraRepository.findByProveedorId(id);
        boolean hasActiveOrders = activeOrders.stream()
                .anyMatch(oc -> oc.getEstadoOrdenCompra() == EstadoOrdenCompra.Pendiente
||
                        oc.getEstadoOrdenCompra() == EstadoOrdenCompra.Enviada);
        if (hasActiveOrders) {
            throw new Exception("No se puede dar de baja el proveedor porque tiene órdenes
de compra pendientes o enviadas.");
        }
        return super.delete(id);
    }

    @Override
    @Transactional
    public List<Proveedor> findAll() throws Exception {
        // Retornar sólo proveedores activos
        return proveedorRepository.findByEstadoTrue();
    }
}
```

OrdenCompraService:

```
package com.example.InvOpBack.Service;



import com.example.InvOpBack.DTOs.OrdenCompraDTO;

import com.example.InvOpBack.Entities.EstadoOrdenCompra;
```

```java
import com.example.InvOpBack.Entities.OrdenCompra;
import org.springframework.stereotype.Service;

import java.util.List;

// Interfaz para el servicio de Orden de Compra.
@Service
public interface OrdenCompraService extends BaseService<OrdenCompra, Long> {
    // Métodos específicos para la lógica de negocio de Orden de Compra.
    OrdenCompra altaOrdenCompra(OrdenCompraDTO dto) throws Exception;
    OrdenCompra modificarOrdenCompra(Long id, OrdenCompraDTO dto) throws Exception;
    boolean cancelarOrdenCompra(Long id) throws Exception;
    OrdenCompra finalizarOrdenCompra(Long id) throws Exception; // Marcar como finalizada y actualizar stock.

    // Metodo para encontrar órdenes de compra activas para un artículo específico.
    List<OrdenCompra> findActiveOrdersForArticulo(Long articuloId, List<EstadoOrdenCompra> estados) throws Exception;
}
```

OrdenCompraServiceImpl:

```java
package com.example.InvOpBack.Service;

import com.example.InvOpBack.DTOs.OrdenCompraDTO;
import com.example.InvOpBack.Entities.*;
import com.example.InvOpBack.Repository.ArticuloProveedorRepository;
import com.example.InvOpBack.Repository.ArticuloRepository;
import com.example.InvOpBack.Repository.OrdenCompraArticuloRepository;
import com.example.InvOpBack.Repository.OrdenCompraRepository;
import com.example.InvOpBack.Repository.ProveedorRepository;
import jakarta.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.io.Serializable;
import java.time.LocalDateTime;
```

```java
import java.util.Arrays;
import java.util.List;
import java.util.Optional;


// Implementación del servicio para la entidad OrdenCompra.
@Service
public class OrdenCompraServiceImpl extends BaseServiceImpl<OrdenCompra, Long> implements
OrdenCompraService, Serializable {


   private OrdenCompraRepository ordenCompraRepository;
   private ProveedorRepository proveedorRepository;
   private ArticuloRepository articuloRepository;
   private ArticuloProveedorRepository articuloProveedorRepository;
   private OrdenCompraArticuloRepository ordenCompraArticuloRepository;



   @Autowired
   public OrdenCompraServiceImpl(OrdenCompraRepository ordenCompraRepository,
                                 ProveedorRepository proveedorRepository,
                                 ArticuloRepository articuloRepository,
                                 ArticuloProveedorRepository articuloProveedorRepository,
                                 OrdenCompraArticuloRepository
ordenCompraArticuloRepository) {
       super(ordenCompraRepository);
       this.ordenCompraRepository = ordenCompraRepository;
       this.proveedorRepository = proveedorRepository;
       this.articuloRepository = articuloRepository;
       this.articuloProveedorRepository = articuloProveedorRepository;
       this.ordenCompraArticuloRepository = ordenCompraArticuloRepository;
   }


   @Override
   @Transactional
   public OrdenCompra altaOrdenCompra(OrdenCompraDTO ordenCompraDTO) throws Exception {
       if (ordenCompraDTO.getId_proveedor() == null) {
           throw new Exception("El ID del proveedor es obligatorio.");
```

```
        }

        if (ordenCompraDTO.getArticulosOrdenCompra() == null ||
ordenCompraDTO.getArticulosOrdenCompra().isEmpty()) {

            throw new Exception("La orden de compra debe contener al menos un artículo.");

        }


        Optional<Proveedor> proveedorOpt =
proveedorRepository.findById(ordenCompraDTO.getId_proveedor());

        if (!proveedorOpt.isPresent() || !proveedorOpt.get().isEstado()) { // Validar que
el proveedor exista y esté activo.

            throw new Exception("Proveedor no encontrado o inactivo con ID: " +
ordenCompraDTO.getId_proveedor());

        }

        Proveedor proveedor = proveedorOpt.get();


        OrdenCompra ordenCompra = new OrdenCompra();

        ordenCompra.setProveedor(proveedor);

        ordenCompra.setEstadoOrdenCompra(EstadoOrdenCompra.Pendiente); // Nueva OC
siempre en Pendiente.

        ordenCompra.setFechaAlta(LocalDateTime.now());

        ordenCompra.setCantidadOrdenCompra(0); // Se calculará después de agregar
artículos.

        ordenCompra.setTotalOrdenCompra(0.0f); // Se calculará después de agregar
artículos.

        ordenCompra.setEstado(true); // Activa por defecto.


        OrdenCompra savedOrdenCompra = ordenCompraRepository.save(ordenCompra); // Guardar
para obtener ID.


        int totalCantidad = 0;

        float totalCosto = 0.0f;


        for (OrdenCompraDTO.ArticuloOrdenCompraDetalle detalle :
ordenCompraDTO.getArticulosOrdenCompra()) {

            if (detalle.getId_articulo() == null) {

                throw new Exception("El ID del artículo es obligatorio en el detalle.");

            }

            if (detalle.getCantidad() <= 0) {

                throw new Exception("La cantidad del artículo debe ser mayor a 0.");
```

```
            }


            Optional<Articulo> articuloOpt =
articuloRepository.findById(detalle.getId_articulo());
            if (!articuloOpt.isPresent() || !articuloOpt.get().isEstado()) { // Validar
que el artículo exista y esté activo.
                throw new Exception("Artículo no encontrado o inactivo con ID: " +
detalle.getId_articulo());
            }
            Articulo articulo = articuloOpt.get();


            // *** Consigna: "Verificar la existencia de otra OC activa (pendiente -
enviada) para ese artículo e informar al usuario en caso afirmativo." ***
            List<EstadoOrdenCompra> estadosActivosOC =
Arrays.asList(EstadoOrdenCompra.Pendiente, EstadoOrdenCompra.Enviada);
            List<OrdenCompra> activeOrders =
ordenCompraRepository.findActiveOrdersForArticulo(articulo.getId(), estadosActivosOC);
            if (!activeOrders.isEmpty()) {
                throw new Exception("Ya existe una orden de compra activa (Pendiente o
Enviada) para el artículo: " + articulo.getNombreArticulo());
            }


            // Buscar la relación ArticuloProveedor para obtener el precio unitario del
proveedor.
            ArticuloProveedor articuloProveedor =
articuloProveedorRepository.findByProveedorIdAndArticuloId(proveedor.getId(),
articulo.getId());
            if (articuloProveedor == null) {
                throw new Exception("El artículo con ID: " + articulo.getId() + " no está
asociado al proveedor con ID: " + proveedor.getId() + " o la relación no existe.");
            }


            OrdenCompraArticulo oca = new OrdenCompraArticulo();
            oca.setOrdenCompra(savedOrdenCompra);
            oca.setArticulo(articulo); // Asignar el artículo a la línea de OC.
            oca.setCantOCA(detalle.getCantidad());
            oca.setPrecioUnitarioOCA(articuloProveedor.getPrecioUnitario()); // Usar
precio del ArticuloProveedor.
            oca.setPrecioSubTotalOCA(detalle.getCantidad() *
articuloProveedor.getPrecioUnitario());
```

```
            oca.setFechaAlta(LocalDateTime.now());
            oca.setEstado(true);

            ordenCompraArticuloRepository.save(oca);

            // Añadir al control de la entidad OrdenCompra (para manejar las relaciones
JPA)
            savedOrdenCompra.getOrdenCompraArticulo().add(oca);

            totalCantidad += detalle.getCantidad();
            totalCosto += oca.getPrecioSubTotalOCA();
        }

        savedOrdenCompra.setCantidadOrdenCompra(totalCantidad);
        savedOrdenCompra.setTotalOrdenCompra(totalCosto);
        return ordenCompraRepository.save(savedOrdenCompra); // Guardar para actualizar
totales.
    }

    @Override
    @Transactional
    public OrdenCompra modificarOrdenCompra(Long id, OrdenCompraDTO ordenCompraDTO) throws
Exception {
        OrdenCompra ordenCompraExistente = ordenCompraRepository.findById(id)
                .orElseThrow(() -> new Exception("Orden de compra no encontrada con ID: "
+ id));

        // *** Consigna: "Puede ser modificada mientras permanezca en ese estado
[Pendiente]." ***
        if (ordenCompraExistente.getEstadoOrdenCompra() != EstadoOrdenCompra.Pendiente) {
            throw new Exception("Solo se pueden modificar órdenes de compra en estado
Pendiente.");
        }

        // Si se permite cambiar el proveedor, se debería validar y actualizar aquí.
        // Por simplicidad, asumimos que el proveedor no cambia.
        Proveedor proveedor = ordenCompraExistente.getProveedor(); // Mantenemos el
proveedor original.
```

```
        if (ordenCompraDTO.getArticulosOrdenCompra() == null ||
ordenCompraDTO.getArticulosOrdenCompra().isEmpty()) {
            throw new Exception("La orden de compra debe contener al menos un artículo
para modificar.");
        }

        // Eliminar las líneas de artículos existentes y agregar las nuevas.
        // Esto es una estrategia común, pero puede ser ineficiente para muchas líneas.
        // Una alternativa es hacer un "diff" para solo actualizar/eliminar/añadir lo
necesario.
ordenCompraArticuloRepository.deleteAll(ordenCompraExistente.getOrdenCompraArticulo());
        ordenCompraExistente.getOrdenCompraArticulo().clear(); // Limpiar la lista para
JPA.

        int totalCantidad = 0;
        float totalCosto = 0.0f;

        for (OrdenCompraDTO.ArticuloOrdenCompraDetalle detalle :
ordenCompraDTO.getArticulosOrdenCompra()) {
            if (detalle.getId_articulo() == null) {
                throw new Exception("El ID del artículo es obligatorio en el detalle.");
            }
            if (detalle.getCantidad() <= 0) {
                throw new Exception("La cantidad del artículo debe ser mayor a 0.");
            }

            Optional<Articulo> articuloOpt =
articuloRepository.findById(detalle.getId_articulo());
            if (!articuloOpt.isPresent() || !articuloOpt.get().isEstado()) {
                throw new Exception("Artículo no encontrado o inactivo con ID: " +
detalle.getId_articulo());
            }
            Articulo articulo = articuloOpt.get();

            ArticuloProveedor articuloProveedor =
articuloProveedorRepository.findByProveedorIdAndArticuloId(proveedor.getId(),
articulo.getId());
```

```
            if (articuloProveedor == null) {
                throw new Exception("El artículo con ID: " + articulo.getId() + " no está asociado al proveedor con ID: " + proveedor.getId() + " o la relación no existe.");
            }

            OrdenCompraArticulo oca = new OrdenCompraArticulo();
            oca.setOrdenCompra(ordenCompraExistente);
            oca.setArticulo(articulo);
            oca.setCantOCA(detalle.getCantidad());
            oca.setPrecioUnitarioOCA(articuloProveedor.getPrecioUnitario());
            oca.setPrecioSubTotalOCA(detalle.getCantidad() * articuloProveedor.getPrecioUnitario());
            oca.setFechaAlta(LocalDateTime.now());
            oca.setEstado(true);

            ordenCompraArticuloRepository.save(oca);
            ordenCompraExistente.getOrdenCompraArticulo().add(oca); // Añadir a la lista de la entidad.

            totalCantidad += detalle.getCantidad();
            totalCosto += oca.getPrecioSubTotalOCA();
        }

        ordenCompraExistente.setCantidadOrdenCompra(totalCantidad);
        ordenCompraExistente.setTotalOrdenCompra(totalCosto);
        ordenCompraExistente.setFechaModificacion(LocalDateTime.now());
        return ordenCompraRepository.save(ordenCompraExistente);
    }

    @Override
    @Transactional
    public boolean cancelarOrdenCompra(Long id) throws Exception {
        OrdenCompra ordenCompra = ordenCompraRepository.findById(id)
                .orElseThrow(() -> new Exception("Orden de compra no encontrada con ID: " + id));
```

```
        // *** Consigna: "Una OC solo puede ser cancelada cuando está en estado Pendiente." ***
        if (ordenCompra.getEstadoOrdenCompra() != EstadoOrdenCompra.Pendiente) {
            throw new Exception("Solo se pueden cancelar órdenes de compra en estado Pendiente.");
        }

        ordenCompra.setEstadoOrdenCompra(EstadoOrdenCompra.Cancelada);
        ordenCompra.setFechaBaja(LocalDateTime.now()); // Baja lógica.
        ordenCompra.setEstado(false); // Marcar como inactiva.
        ordenCompraRepository.save(ordenCompra);
        return true;
    }

    @Override
    @Transactional
    public OrdenCompra finalizarOrdenCompra(Long id) throws Exception {
        OrdenCompra ordenCompra = ordenCompraRepository.findById(id)
                .orElseThrow(() -> new Exception("Orden de compra no encontrada con ID: " + id));

        // *** Consigna: "El último paso es colocar la OC en estado Finalizada, ahí el sistema debe actualizar el inventario del producto que se ordenó." ***
        // Y "Cuando una OC pasa al estado Enviada no puede ser modificada ni cancelada. Este paso se realiza en forma manual."
        // Esto implica que una OC debe estar en estado 'Enviada' para ser 'Finalizada'.
        if (ordenCompra.getEstadoOrdenCompra() != EstadoOrdenCompra.Enviada) {
            throw new Exception("Solo se pueden finalizar órdenes de compra en estado Enviada.");
        }

        // Actualizar inventario de los artículos en la orden de compra.
        for (OrdenCompraArticulo oca : ordenCompra.getOrdenCompraArticulo()) {
            Articulo articulo = oca.getArticulo(); // Ya tenemos el artículo en OCA.
            if (articulo != null) {
                articulo.setStockActual(articulo.getStockActual() + oca.getCantOCA());
                articulo.setFechaModificacion(LocalDateTime.now()); // Actualizar la fecha de modificación del artículo.
```

```
                articuloRepository.save(articulo);


                ArticuloProveedor articuloProveedor =
articuloProveedorRepository.findByProveedorIdAndArticuloId(articulo.getProveedorPredeterm
inado().getId(),articulo.getId());


                // *** Consigna: "Si con la OC la cantidad del artículo no supera el Punto
de Pedido con modelo Lote Fijo informar al usuario." ***
                // Esto es una advertencia, no un error que impida la finalización.
                if (articuloProveedor.getModeloInventario() == ModeloInventario.loteFijo
&& articulo.getStockActual() < articuloProveedor.getPuntoPedido()) {
                    System.out.println("ADVERTENCIA: El stock actual del artículo '" +
articulo.getNombreArticulo() +
                            "' (" + articulo.getStockActual() + ") aún no supera su Punto
de Pedido (" +
                            articuloProveedor.getPuntoPedido() + ") después de finalizar
la OC. ID Articulo: " + articulo.getId());
                    // Aquí podrías implementar un sistema de notificaciones o logs más
robusto.
                }


            }
        }
        ordenCompra.setEstadoOrdenCompra(EstadoOrdenCompra.Finalizada);
        ordenCompra.setFechaModificacion(LocalDateTime.now()); // O fecha de finalización.
        return ordenCompraRepository.save(ordenCompra);
    }


    @Override
    @Transactional
    public List<OrdenCompra> findActiveOrdersForArticulo(Long articuloId,
List<EstadoOrdenCompra> estados) throws Exception {
        // Método para encontrar órdenes de compra activas para un artículo específico.
        // Utiliza el método de repositorio `findActiveOrdersForArticulo` que ya
definimos.
        return ordenCompraRepository.findActiveOrdersForArticulo(articuloId, estados);
    }
}
```

VentaService:

```java
package com.example.InvOpBack.Service;


import com.example.InvOpBack.DTOs.VentaDTO;
import com.example.InvOpBack.Entities.Venta;
import org.springframework.stereotype.Service;


// Interfaz para el servicio de Venta.
@Service
public interface VentaService extends BaseService<Venta, Long> {
    // Métodos específicos para la lógica de negocio de Venta.
    Venta altaVenta(VentaDTO dto) throws Exception;
    // La modificación de ventas suele ser restringida o no permitida.
    // Si se permite, la lógica es compleja (revertir stock, etc.).
    Venta modificarVenta(Long id, VentaDTO dto) throws Exception;
    // Baja lógica para ventas.
    boolean bajaVenta(Long id) throws Exception;
}
```

VentaServicerImpl:

```java
package com.example.InvOpBack.Service;


import com.example.InvOpBack.DTOs.OrdenCompraDTO;
import com.example.InvOpBack.DTOs.VentaDTO;
import com.example.InvOpBack.Entities.Articulo;
import com.example.InvOpBack.Entities.ArticuloProveedor;
import com.example.InvOpBack.Entities.ArticuloVenta;
import com.example.InvOpBack.Entities.EstadoOrdenCompra;
import com.example.InvOpBack.Entities.ModeloInventario;
import com.example.InvOpBack.Entities.OrdenCompra;
import com.example.InvOpBack.Entities.Venta;
import com.example.InvOpBack.Repository.ArticuloRepository;
import com.example.InvOpBack.Repository.ArticuloProveedorRepository;
import com.example.InvOpBack.Repository.ArticuloVentaRepository;
import com.example.InvOpBack.Repository.OrdenCompraRepository;
```

```java
import com.example.InvOpBack.Repository.VentaRepository;
import jakarta.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.io.Serializable;
import java.time.LocalDateTime;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;

// Implementación del servicio para la entidad Venta.
@Service
public class VentaServiceImpl extends BaseServiceImpl<Venta, Long> implements
VentaService, Serializable {

   private VentaRepository ventaRepository;
   private ArticuloRepository articuloRepository;
   private ArticuloVentaRepository articuloVentaRepository;
   private OrdenCompraService ordenCompraService; // Inyectado para la lógica de
generación de OC.
   private ArticuloProveedorRepository articuloProveedorRepository; // Inyectado para
obtener ArticuloProveedor.

   @Autowired
   public VentaServiceImpl(VentaRepository ventaRepository,
                           ArticuloRepository articuloRepository,
                           ArticuloVentaRepository articuloVentaRepository,
                           OrdenCompraService ordenCompraService,
                           ArticuloProveedorRepository articuloProveedorRepository) {
       super(ventaRepository);
       this.ventaRepository = ventaRepository;
       this.articuloRepository = articuloRepository;
       this.articuloVentaRepository = articuloVentaRepository;
       this.ordenCompraService = ordenCompraService;
       this.articuloProveedorRepository = articuloProveedorRepository;
```

```
    }

    @Override
    @Transactional
    public Venta altaVenta(VentaDTO ventaDTO) throws Exception {
        if (ventaDTO.getArticulosVenta() == null ||
ventaDTO.getArticulosVenta().isEmpty()) {
            throw new Exception("La venta debe contener al menos un artículo.");
        }

        Venta venta = new Venta();
        venta.setFechaVenta(LocalDateTime.now()); //DETERMINAR SI LO DEJAMOS ASI O LO
DETERMINA EL FRONT
        venta.setCostoTotal(0.0f); // Se calculará.
        venta.setFechaAlta(LocalDateTime.now());
        venta.setEstado(true);

        Venta savedVenta = ventaRepository.save(venta); // Guardar para obtener el ID.

        float totalCosto = 0.0f;

        for (VentaDTO.ArticuloVentaDetalle detalle : ventaDTO.getArticulosVenta()) {
            if (detalle.getId_articulo() == null) {
                throw new Exception("El ID del artículo es obligatorio en el detalle de
venta.");
            }
            if (detalle.getCantidad() <= 0) {
                throw new Exception("La cantidad del artículo en la venta debe ser mayor a
0.");
            }

            Optional<Articulo> articuloOpt =
articuloRepository.findById(detalle.getId_articulo());
            if (!articuloOpt.isPresent() || !articuloOpt.get().isEstado()) { // Validar
que el artículo exista y esté activo.
                throw new Exception("Artículo no encontrado o inactivo con ID: " +
detalle.getId_articulo());
            }
```

```
            Articulo articulo = articuloOpt.get();


            // *** Consigna: "No se debe permitir ventas superiores al stock actual." ***
            if (articulo.getStockActual() < detalle.getCantidad()) {
                throw new Exception("Stock insuficiente para el artículo: " +
articulo.getNombreArticulo() + ". Stock actual: " + articulo.getStockActual() + ",
Cantidad solicitada: " + detalle.getCantidad());
            }


            ArticuloVenta av = new ArticuloVenta();
            av.setVenta(savedVenta);
            av.setArticulo(articulo);
            av.setCantArticuloVenta(detalle.getCantidad());
            av.setPrecioUnitario(articulo.getPrecioVentaArt()); // Usar el precio de venta
del artículo.
            av.setPrecioSubTotal(detalle.getCantidad() * articulo.getPrecioVentaArt());
            av.setFechaAlta(LocalDateTime.now());
            av.setEstado(true);


            articuloVentaRepository.save(av);
            savedVenta.getArticuloVenta().add(av); // Añadir a la lista de la entidad
Venta.


            totalCosto += av.getPrecioSubTotal();


            // Actualizar stock del artículo
            articulo.setStockActual(articulo.getStockActual() - detalle.getCantidad());
            articulo.setFechaModificacion(LocalDateTime.now());
            articuloRepository.save(articulo); // Guardar el artículo con el stock
actualizado.


            //DEBEN USAR ESTA JPQL PARA INGRESAR A LOS CAMPOS NECESARIOS
            ArticuloProveedor articuloProveedor =
articuloProveedorRepository.findByProveedorIdAndArticuloId(articulo.getProveedorPredeterm
inado().getId(),articulo.getId());
```

```
            // *** Lógica para generar Orden de Compra si es Lote Fijo y se cumple la
condición ***

            // "Si el modelo de Inventario es LF, la cantidad actual es >= al PP y no hay
ordenes pendientes/enviada para el articulo, generar una orden..."

            // Asumo que la lógica correcta es: "Si el stock actual es MENOR O IGUAL al
Punto de Pedido".

            // Y que el modelo de inventario se obtiene del Articulo.




            if (articuloProveedor.getModeloInventario() == ModeloInventario.loteFijo &&
articulo.getStockActual() < articuloProveedor.getPuntoPedido()) {

                if (articulo.getStockActual() <= articuloProveedor.getPuntoPedido()) {

                    List<EstadoOrdenCompra> estadosActivosOC =
Arrays.asList(EstadoOrdenCompra.Pendiente, EstadoOrdenCompra.Enviada);

                    List<OrdenCompra> activeOrders =
ordenCompraService.findActiveOrdersForArticulo(articulo.getId(), estadosActivosOC);

                    if (activeOrders.isEmpty()) {

                        if (articulo.getProveedorPredeterminado() == null) {

                            System.out.println("ADVERTENCIA: Artículo '" +
articulo.getNombreArticulo() + "' alcanzó Punto de Pedido, pero no tiene proveedor
predeterminado para generar OC automática.");

                        } else {

                            OrdenCompraDTO ocAutomaticaDTO = new OrdenCompraDTO();


ocAutomaticaDTO.setId_proveedor(articulo.getProveedorPredeterminado().getId());

                            OrdenCompraDTO.ArticuloOrdenCompraDetalle ocDetalle = new
OrdenCompraDTO.ArticuloOrdenCompraDetalle();

                            ocDetalle.setId_articulo(articulo.getId());

                            ocDetalle.setCantidad((int)
articuloProveedor.getCantidadPedido());


ocAutomaticaDTO.setArticulosOrdenCompra(Arrays.asList(ocDetalle));

                            try {

                                ordenCompraService.altaOrdenCompra(ocAutomaticaDTO);

                                System.out.println("INFO: Orden de compra automática
generada para artículo " + articulo.getNombreArticulo()

                                        + " con cantidad " + ocDetalle.getCantidad() +
".");

                            } catch (Exception e) {

                                System.err.println("ERROR al generar OC automática para "
+ articulo.getNombreArticulo() + ": " + e.getMessage());
```

```
                            }
                        }
                    } else {
                        System.out.println("INFO: Artículo '" +
articulo.getNombreArticulo() + "' alcanzó Punto de Pedido, pero ya tiene una OC activa.
No se generó nueva OC.");
                    }
                }
            }
        }
        savedVenta.setCostoTotal(totalCosto);
        return ventaRepository.save(savedVenta);
    }

    @Override
    @Transactional
    public Venta modificarVenta(Long id, VentaDTO ventaDTO) throws Exception {
        // La modificación de ventas suele ser restringida o no permitida en sistemas de
inventario
        // para mantener la integridad histórica. Si se implementa, requiere una lógica
compleja
        // para revertir y re-aplicar cambios de stock.
        throw new Exception("La modificación de ventas no está permitida o
implementada.");
    }

    @Override
    @Transactional
    public boolean bajaVenta(Long id) throws Exception {
        Venta venta = ventaRepository.findById(id)
                .orElseThrow(() -> new Exception("Venta no encontrada con ID: " + id));

        // *** Lógica de baja lógica de venta: Reversar el stock de los artículos. ***
        // Esto depende de la política de negocio (ej. ¿Se puede "deshacer" una venta?).
        // Si se revierte el stock, asegúrate de no generar nuevas OCs en este proceso.
        for (ArticuloVenta av : venta.getArticuloVenta()) {
            Articulo articulo = av.getArticulo();
```

```
            if (articulo != null) {

                articulo.setStockActual(articulo.getStockActual() +
av.getCantArticuloVenta()); // Revertir stock.

                articulo.setFechaModificacion(LocalDateTime.now());

                articuloRepository.save(articulo);

            }

        }



        venta.setEstado(false); // Marcar como inactiva.

        venta.setFechaBaja(LocalDateTime.now());

        ventaRepository.save(venta);

        return true;

    }

}
```

DTOs:

ArticuloDTO:

```
package com.example.InvOpBack.DTOs;

import com.example.InvOpBack.Entities.ModeloInventario;
import lombok.*;

@Data
public class ArticuloDTO {

    private String nombreArticulo;
    private String descripcionArticulo;
    private Float precioVentaArt;
    private Float costoAlmacenamiento;
    private Integer stockActual;
    private Integer demandaArticulo;

    private Long idProveedorPredeterminado;
}
```

ArticuloProveedorDTO:

```
package com.example.InvOpBack.DTOs;


import com.example.InvOpBack.Entities.ModeloInventario;
import lombok.Data;
import lombok.Getter;
import lombok.Setter;

import java.util.Date;

@Data
@Getter
```

```java
@Setter
public class ArticuloProveedorDTO {
   private float costoCompra;
   private float costoPedido;
   private int demoraEntrega;
   private float precioUnitario;
   private ModeloInventario modeloInventario;
   private int desviacionEstandar;
   private int intervaloRevision;
   private Integer stockSeguridad;
   private Integer loteOptimo;
   private Integer puntoPedido;
   private Long  id_proveedor;
   private Long id_articulo;
   private float costoPorPedido;
   private int valorCGI;
   private int inventarioMaximo;
}
```

ListadoDTO:

```java
package com.example.InvOpBack.DTOs;


import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.Getter;
import lombok.Setter;

import java.util.Date;

@Data
@Getter
@Setter
@AllArgsConstructor
public class ListadoDTO {
   private Long idArticulo;
   private String nombreArticulo;
   private boolean esProveedorPredeterminado;

}
```

ProveedorDTO:

```java
package com.example.InvOpBack.DTOs;

import lombok.Data;
import lombok.Getter;
import lombok.Setter;

import java.util.List;

@Data
@Getter
@Setter
public class ProveedorDTO {
   private String nombreProveedor;
   private String Cuit;
   private List<ArticuloProveedorDTO> articulosAsociados;

}
```

ProveedorDeterminadoDTO:

```
package com.example.InvOpBack.DTOs;

import lombok.Data;
import lombok.Getter;
import lombok.Setter;

@Data
@Getter
@Setter
public class ProveedorPredeterminadoDTO {
   private Long idProveedor;

}
```

OrdenCompraDTO:

```
package com.example.InvOpBack.DTOs;



import com.example.InvOpBack.Entities.EstadoOrdenCompra;

import lombok.Data;

import lombok.Getter;

import lombok.Setter;

import java.util.List;



@Data

@Getter

@Setter

public class OrdenCompraDTO {

   private Long id_proveedor; // ID del proveedor de la orden de compra.

   private List<ArticuloOrdenCompraDetalle> articulosOrdenCompra; // Lista de artículos
en la orden.



   // Clase interna para el detalle de cada artículo en la orden de compra.

   @Data

   @Getter

   @Setter

   public static class ArticuloOrdenCompraDetalle {

       private Long id_articulo; // ID del artículo.

       private int cantidad; // Cantidad del artículo.

   }
```

```
}
```

VentaDTO:

```
package com.example.InvOpBack.DTOs;


import lombok.Data;

import lombok.Getter;

import lombok.Setter;


import java.time.LocalDateTime;

import java.util.List;


@Data

@Getter

@Setter

public class VentaDTO {


   private List<ArticuloVentaDetalle> articulosVenta; // Lista de artículos en la venta.


   // Clase interna para el detalle de cada artículo en la venta.

   @Data

   @Getter

   @Setter

   public static class ArticuloVentaDetalle {

       private Long id_articulo; // ID del artículo.

       private int cantidad; // Cantidad del artículo a vender.

       // El precio unitario y subtotal se pueden calcular en el servicio.

   }

}
```

Controller:

ArticuloController:

```
package com.example.InvOpBack.Controller;

import com.example.InvOpBack.DTOs.ArticuloDTO;
import com.example.InvOpBack.DTOs.ProveedorPredeterminadoDTO;
```

```java
import com.example.InvOpBack.Entities.Articulo;
import com.example.InvOpBack.Service.ArticuloService;
import com.example.InvOpBack.Service.ArticuloServiceImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@CrossOrigin(origins = "*")
@RequestMapping(path = "/articulos")
public class ArticuloController extends BaseControllerImpl<Articulo, ArticuloServiceImpl>
{

   @Autowired
   private ArticuloService articuloService;


   @PostMapping("/crear")
   public ResponseEntity<?> altaArticulo(@RequestBody ArticuloDTO ArticuloDTO) throws
Exception{
       try{
           Articulo articulo = articuloService.altaArticulo(ArticuloDTO);
           return ResponseEntity.status(HttpStatus.OK).body(articulo);
       }catch(Exception e){
           return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error
inesperado: "+ e.getMessage());
       }
   }

   @PutMapping("/modificar/{id}")
   public ResponseEntity<?> modificarArticulo(@PathVariable Long id, @RequestBody
ArticuloDTO articuloDTO) {
       try {
           Articulo articuloModificado = articuloService.modificarArticulo(id,
articuloDTO);
           return ResponseEntity.status(HttpStatus.OK).body(articuloModificado);
       } catch (Exception e) {
           return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Error al modificar
artículo: " + e.getMessage());
       }
   }
   @PutMapping("/proveedor-predeterminado/{id}")
   public ResponseEntity<?> establecerProveedor(@PathVariable Long id, @RequestBody
ProveedorPredeterminadoDTO proveedorPredeterminadoDTO ) {
       try {
           Articulo articuloModificado = articuloService.establecerProveedor(id,
proveedorPredeterminadoDTO);
           return ResponseEntity.status(HttpStatus.OK).body(articuloModificado);
       } catch (Exception e) {
           return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Error al modificar
artículo: " + e.getMessage());
       }
   }
}
```

ArticuloProveedorController:

```java
package com.example.InvOpBack.Controller;

import com.example.InvOpBack.DTOs.ArticuloProveedorDTO;
import com.example.InvOpBack.Entities.ArticuloProveedor;
import com.example.InvOpBack.Service.ArticuloProveedorServiceImpl;
import org.springframework.beans.factory.annotation.Autowired;
```

```java
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@CrossOrigin(origins = "*")
@RequestMapping(path = "/articulo-proveedor")
public class ArticuloProveedorController extends BaseControllerImpl<ArticuloProveedor,
ArticuloProveedorServiceImpl> {

   @Autowired
   ArticuloProveedorServiceImpl articuloProveedorService;

   @PostMapping("/crear")
   public ResponseEntity<?> altaAP (@RequestBody ArticuloProveedorDTO
articuloProveedorDTO) throws Exception{
       try{
           ArticuloProveedor artprov =
articuloProveedorService.altaAP(articuloProveedorDTO);
           return ResponseEntity.ok(artprov);
       } catch (Exception e) {
           return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(e.getMessage());
       }
   }

   @GetMapping("/listado/{proveedorId}")
   public ResponseEntity<?> listadoArticulosPorProveedor (@PathVariable Long proveedorId)
{
       try {
           return
ResponseEntity.status(HttpStatus.OK).body(servicio.listadoArticulosPorProveedor(proveedo
rId));
       } catch (Exception e) {
           return ResponseEntity.status(HttpStatus.NOT_FOUND).body(("{\"error\": \"" +
e.getMessage() + "\"}"));
       }
   }
}
```

OrdenCompraController:

```java
package com.example.InvOpBack.Controller;



import com.example.InvOpBack.DTOs.OrdenCompraDTO;

import com.example.InvOpBack.Entities.EstadoOrdenCompra;

import com.example.InvOpBack.Entities.OrdenCompra;

import com.example.InvOpBack.Service.OrdenCompraService;

import com.example.InvOpBack.Service.OrdenCompraServiceImpl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.*;
```

```java
import java.util.Arrays;


// Controlador REST para la entidad OrdenCompra.
@RestController
@CrossOrigin(origins = "*")
@RequestMapping(path = "/ordenescompra")
public class OrdenCompraController extends BaseControllerImpl<OrdenCompra,
OrdenCompraServiceImpl> {


   @Autowired
   private OrdenCompraService ordenCompraService;


   // Endpoint para dar de alta una nueva orden de compra.
   @PostMapping("/crear")
   public ResponseEntity<?> altaOrdenCompra(@RequestBody OrdenCompraDTO ordenCompraDTO) {
       try {
           OrdenCompra ordenCompra = ordenCompraService.altaOrdenCompra(ordenCompraDTO);
           return ResponseEntity.status(HttpStatus.CREATED).body(ordenCompra);
       } catch (Exception e) {
           return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Error al crear la
orden de compra: " + e.getMessage());
       }
   }


   // Endpoint para modificar una orden de compra.
   @PutMapping("/modificar/{id}")
   public ResponseEntity<?> modificarOrdenCompra(@PathVariable Long id, @RequestBody
OrdenCompraDTO ordenCompraDTO) {
       try {
           OrdenCompra ordenCompraModificada =
ordenCompraService.modificarOrdenCompra(id, ordenCompraDTO);
           return ResponseEntity.status(HttpStatus.OK).body(ordenCompraModificada);
       } catch (Exception e) {
           return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Error al modificar
la orden de compra: " + e.getMessage());
       }
   }
```

```
    // Endpoint para cancelar una orden de compra.
    @PutMapping("/cancelar/{id}")
    public ResponseEntity<?> cancelarOrdenCompra(@PathVariable Long id) {
        try {
            boolean cancelado = ordenCompraService.cancelarOrdenCompra(id);
            if (cancelado) {
                return ResponseEntity.status(HttpStatus.OK).body("Orden de compra " + id
+ " cancelada exitosamente.");
            } else {
                return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("No se pudo
cancelar la orden de compra " + id + ".");
            }
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Error al cancelar
la orden de compra: " + e.getMessage());
        }
    }


    // Endpoint para finalizar una orden de compra (actualiza inventario).
    @PutMapping("/finalizar/{id}")
    public ResponseEntity<?> finalizarOrdenCompra(@PathVariable Long id) {
        try {
            OrdenCompra ordenCompraFinalizada =
ordenCompraService.finalizarOrdenCompra(id);
            return ResponseEntity.status(HttpStatus.OK).body(ordenCompraFinalizada);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Error al finalizar
la orden de compra: " + e.getMessage());
        }
    }


    // Endpoint para obtener órdenes de compra activas para un artículo (para la lógica de
venta/reposición).
    @GetMapping("/activasPorArticulo/{articuloId}")
    public ResponseEntity<?> getActiveOrdersForArticulo(@PathVariable Long articuloId) {
        try {
```

```
            // Se asume que el servicio ya tiene la lógica para determinar qué estados son
"activos".

            // Podrías pasar los estados explícitamente si el servicio lo requiere.

            return
ResponseEntity.status(HttpStatus.OK).body(ordenCompraService.findActiveOrdersForArticulo
(articuloId, Arrays.asList(EstadoOrdenCompra.Pendiente, EstadoOrdenCompra.Enviada)));

        } catch (Exception e) {

            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error al
buscar órdenes activas: " + e.getMessage());

        }

    }



    // Aquí se heredan los métodos getAll, getOne de BaseControllerImpl.

    // El metodo delete de BaseControllerImpl hará una baja lógica de la OC.

}
```

ProveedorController:

```
package com.example.InvOpBack.Controller;


import com.example.InvOpBack.DTOs.ProveedorDTO;
import com.example.InvOpBack.Entities.Proveedor;
import com.example.InvOpBack.Service.ProveedorServiceImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@CrossOrigin(origins = "*")
@RequestMapping(path = "/proveedores") // Cambiado a plural para consistencia con
/articulos
public class ProveedorController extends BaseControllerImpl<Proveedor,
ProveedorServiceImpl> {

    @Autowired
    ProveedorServiceImpl proveedorService;

    @PostMapping("/crear")
    public ResponseEntity<?> altaProveedor (@RequestBody ProveedorDTO proveedorDTO) throws
Exception{
        try{
            Proveedor proveedor = proveedorService.altaProveedor (proveedorDTO);
            return ResponseEntity.ok(proveedor);
        } catch (Exception e) {
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(e.getMessage());
        }
    }


    @PutMapping("/modificar/{id}") // Nuevo endpoint para modificar proveedor
```

```
   public ResponseEntity<?> modificarProveedor(@PathVariable Long id, @RequestBody
ProveedorDTO proveedorDTO) {
       try {
           Proveedor proveedorModificado = proveedorService.modificarProveedor(id,
proveedorDTO);
           return ResponseEntity.status(HttpStatus.OK).body(proveedorModificado);
       } catch (Exception e) {
           return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Error al modificar
proveedor: " + e.getMessage());
       }
   }


   // Se renombra para evitar conflicto con el delete genérico de BaseControllerImpl si
se usa el mismo path.
   @DeleteMapping("/baja/{id}")
   public ResponseEntity<?> bajaProveedor(@PathVariable Long id) {
       try {
           boolean bajaExitosa = proveedorService.bajaProveedor(id);
           if (bajaExitosa) {
               return ResponseEntity.status(HttpStatus.OK).body("Proveedor dado de baja
lógicamente.");
           } else {
               return ResponseEntity.status(HttpStatus.NOT_FOUND).body("No se pudo dar
de baja el proveedor.");
           }
       } catch (Exception e) {
           return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Error al dar de
baja proveedor: " + e.getMessage());
       }
   }
}
```

VentaController:

```
package com.example.InvOpBack.Controller;


import com.example.InvOpBack.DTOs.VentaDTO;

import com.example.InvOpBack.Entities.Venta;

import com.example.InvOpBack.Service.VentaService;

import com.example.InvOpBack.Service.VentaServiceImpl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.*;


@RestController

@CrossOrigin(origins = "*")

@RequestMapping(path = "/ventas")

public class VentaController extends BaseControllerImpl<Venta, VentaServiceImpl> {
```

```
    @Autowired
    private VentaService ventaService;


    // Endpoint para dar de alta una nueva venta.
    @PostMapping("/crear")
    public ResponseEntity<?> altaVenta(@RequestBody VentaDTO ventaDTO) {
        try {
            Venta venta = ventaService.altaVenta(ventaDTO);
            return ResponseEntity.status(HttpStatus.CREATED).body(venta);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Error al crear la
venta: " + e.getMessage());
        }
    }


    // Endpoint para modificar una venta (si estuviera permitido, el servicio lo
manejará).
    @PutMapping("/modificar/{id}")
    public ResponseEntity<?> modificarVenta(@PathVariable Long id, @RequestBody VentaDTO
ventaDTO) {
        try {
            Venta ventaModificada = ventaService.modificarVenta(id, ventaDTO);
            return ResponseEntity.status(HttpStatus.OK).body(ventaModificada);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Error al modificar
la venta: " + e.getMessage());
        }
    }


    // Endpoint para realizar la baja lógica de una venta.
    @DeleteMapping("/baja/{id}")
    public ResponseEntity<?> bajaVenta(@PathVariable Long id) {
        try {
            boolean bajaExitosa = ventaService.bajaVenta(id);
            if (bajaExitosa) {
                return ResponseEntity.status(HttpStatus.OK).body("Venta " + id + " dada
de baja lógicamente.");
```

```
            } else {

                return ResponseEntity.status(HttpStatus.NOT_FOUND).body("No se pudo dar
de baja la venta " + id + ".");

            }

        } catch (Exception e) {

            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Error al dar de
baja la venta: " + e.getMessage());

        }

    }


}
```

InvOpBackApplication

```
package com.example.InvOpBack;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class InvOpBackApplication {

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

}
```

Application.properties:

```
spring.application.name=InvOpBack

# Configuración de la base de datos
spring.datasource.url=jdbc:postgresql://localhost:5432/BD-InvOp
spring.datasource.username=postgres
spring.datasource.password=root
spring.datasource.driver-class-name=org.postgresql.Driver

# Configuración de Hibernate (JPA)
spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```