Universidad de los Andes
Departamento de Ingeniería de Sistemas y Computación

Entrega 1:
Documentación

Ana María Hernández - 202220870
Karol Vanesa Montaño - 202317397
Julian Mondragon - 202221122

Sistemas Transaccionales
Sección 06
Docente Linda Rodríguez Castro

Bogotá, Colombia
2024-20

Documentación de las clases de Java Spring y de la arquitectura de la aplicación

Arquitectura de la aplicación

La arquitectura de ProyectoSuperAndes está basada en el patrón MVC (Modelo-Vista-Controlador). Así, tenemos por el momento tres principales carpetas: modelo, controller y repositorio.

Los modelos representan los conceptos claves para este negocio, así como las tablas del modelo relacional que diseñamos. Por ejemplo, clases como Producto, OrdenDeCompra y Proveedor modelan las relaciones de la base de datos, es por eso que usamos anotaciones JPA para hacer claras tanto sus relaciones como para que se reflejen las llaves primarias compuestas de algunas clases.

Los repositorios son interfaces que extienden JpaRepository, como ProductoRepository, BodegaRepository, y OrdenDeCompraRepository. Las clases de esta carpeta hacen de intermediarios entre la base de datos y la lógica de negocio de la aplicación que hemos hecho; es decir, gracias a estas clases se permite ejecutar las operaciones CRUD desde nuestro código, y son una herramienta para dar respuesta a lo que solicitan los controllers.

Los controllers o controladores, como ProductoController y OrdenDeCompraController, son los responsables de recibir y manejar las peticiones HTTP que llegan. Para nuestro caso, estas pueden llegar del puerto local 8080 cuando ingresamos una URL en nuestro navegador o de Postman. A través de los endpoints (o parte de las URLs) que definimos, se reciben las solicitudes.

Clarificamos también que la clase ProyectoApplication es el main o el punto de partida que inicia nuestra aplicación.

En resumen sobre la arquitectura de nuestra aplicación, los modelos definen cómo se estructuran los datos según la base de datos relacional que hicimos en SQLdeveloper y el modelo relacional que seguimos, los repositorios gestionan el acceso a los datos y por ende tienen notación SQL en sus anotaciones de Query, y los controladores manejan la interacción con las peticiones del usuario.

Finalmente, al usar herramientas como Postman podemos probar dichas interacciones mediante solicitudes GET, POST, PUT o DELETE, que invocan los métodos que hicimos en los controladores.

Gracias a todos estos elementos en conjunto, logramos ver en tiempo real las respuestas de nuestra aplicación, así como el procesamiento de los datos que creamos.

Sobre las clases

Clase Bodega

La clase Bodega se encarga de mapear la tabla bodega en la base de datos. Representa una bodega asociada a una sucursal, con atributos como id, nombre, tamaño y capacidad para almacenar productos. Tiene una relación de muchos a uno con la entidad Sucursal, lo

que significa que cada bodega está vinculada a una sucursal. Además, utiliza JPA para indicar que se genera automáticamente el valor del id, que es la llave primaria. Los métodos getters y setters permiten acceder y modificar los valores de los atributos de la clase.

```java
//Clase que mapea la tabla Bodega en la base de datos
@Entity
@Table(name="bodega")
public class Bodega {
    @Id //Indica que es la llave primaria
    @GeneratedValue(strategy=GenerationType.AUTO) //Genera automaticamente el valor de la llave primaria

    //Atributos de la clase
    private Integer id;
    private String nombre;
    private Double tamanio;
    private Integer capacidad;

    //Relacion con la tabla Sucursal
    @ManyToOne
    @JoinColumn(name="id_sucursal", referencedColumnName="id")
    private Sucursal idSucursal;

    public Bodega(String nombre, Double tamanio, Integer capacidad, Sucursal idSucursal){
        this.nombre = nombre;
        this.tamanio = tamanio;
        this.capacidad = capacidad;
        this.idSucursal = idSucursal;
    }

    public Bodega(){
    ;}
```

Clase Categoria

La clase Categoria se encarga de mapear la tabla categoría en la base de datos. Representa una categoría de productos con atributos como código, nombre, descripción y características de almacenamiento. El atributo código es la llave primaria, y se genera automáticamente utilizando JPA. Los métodos getters y setters permiten acceder y modificar los valores de los atributos de la clase.

```java
//Clase que mapea la tabla Categoria en la base de datos
@Entity
@Table(name="categoria")
public class Categoria {

    @Id //Indica que es la llave primaria
    @GeneratedValue(strategy = GenerationType.AUTO) //Genera automaticamente el valor de la llave primaria

    //Atributos de la clase
    private Integer codigo;
    private String nombre;
    private String descripcion;
    private String caracteristicasAlmacenamiento;

    public Categoria(String nombre, String descripcion, String caracteristicasAlmacenamiento){
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.caracteristicasAlmacenamiento = caracteristicasAlmacenamiento;
    }

    public Categoria()
    {;}
```

Clase Ciudad

La clase Ciudad se encarga de mapear la tabla ciudad en la base de datos. Representa una ciudad con los atributos código y nombre. El atributo código es la llave primaria y se genera automáticamente utilizando JPA. Los métodos getters y setters permiten acceder y modificar los valores de los atributos.

```java
//Clase que mapea la tabla Ciudad en la base de datos
@Entity
@Table(name="ciudad")
public class Ciudad {

    @Id //Indica que es la llave primaria
    @GeneratedValue(strategy=GenerationType.AUTO) //Genera automaticamente el valor de la llave primaria
    private Integer codigo;

    //Atributos de la clase
    private String nombre;


    public Ciudad(String nombre)
    {
        this.nombre = nombre;
    }

    public Ciudad()
    {;}
```

Clase OrdenDeCompra

La clase OrdenDeCompra se encarga de mapear la tabla orden_de_compra en la base de datos. Representa una orden de compra con atributos como número, fecha de entrega, estado y fecha de creación. El número es la llave primaria y se genera automáticamente usando JPA. Además, la clase tiene relaciones de muchos a uno con las tablas Sucursal y Proveedor, lo que permite asociar cada orden de compra con una sucursal y un proveedor. Los métodos getters y setters facilitan el acceso y modificación de los atributos.

```java
//Clase que mapea la tabla OrdenDeCompra en la base de datos
@Entity
@Table(name="orden_de_compra")
public class OrdenDeCompra {

    @Id //Indica que es la llave primaria
    @GeneratedValue(strategy=GenerationType.AUTO) //Genera automaticamente el valor de la llave primaria
    private Integer numero;

    //Atributos de la clase
    private LocalDate fechaEntrega;
    private String estado;
    private LocalDate fechaCreacion;


    //Relacion con la tabla Sucursal
    @ManyToOne
    @JoinColumn(name="id_sucursal", referencedColumnName="id")
    private Sucursal idSucursal;

    //Relacion con la tabla Proveedor
    @ManyToOne
    @JoinColumn(name="nit_proveedor", referencedColumnName="nit")
    private Proveedor nitProveedor;

    public OrdenDeCompra(LocalDate fechaEntrega, String estado, LocalDate fechaCreacion, Sucursal idSucursal, Proveedor nitProveedor){
        this.fechaEntrega = fechaEntrega;
        this.estado = estado;
        this.fechaCreacion = fechaCreacion;
        this.idSucursal = idSucursal;
        this.nitProveedor = nitProveedor;
    }
```

Clase Producto

La clase Producto se encarga de mapear la tabla producto en la base de datos. Representa un producto con atributos como identificador, nombre, costo en bodega (este se asume como el precio del producto), presentación, cantidad por presentación, unidad de medida, volumen del empaque, peso del empaque, fecha de expiración y código de barras. El identificador es la llave primaria y se genera automáticamente usando JPA. Además, la clase tiene una relación de muchos a uno con la tabla Categoria, lo que permite clasificar el producto según su categoría. Los métodos getters y setters permiten acceder y modificar los atributos del producto.

```java
//Clase que mapea la tabla Producto en la base de datos
@Entity
@Table(name="producto")
public class Producto {

    @Id //Indica que es la llave primaria
    @GeneratedValue(strategy=GenerationType.AUTO) //Genera automaticamente el valor de la llave primaria
    private Integer identificador;

    //Atributos de la clase
    private String nombre;
    private Double costoEnBodega;
    private String presentacion;
    private Double cantidadPresentacion;
    private String unidadMedida;
    private String volumenEmpaque;
    private String pesoEmpaque;
    private LocalDate fechaExpiracion;
    private String codigoDeBarras;

    //Relacion con la tabla Categoria
    @ManyToOne
    @JoinColumn(name="clasificacion_categoria", referencedColumnName = "codigo")
    private Categoria clasificacionCategoria;

    public Producto(String nombre, Double costoEnBodega, String presentacion,
        Double cantidadPresentacion, String unidadMedida, String volumenEmpaque,
        String pesoEmpaque, LocalDate fechaExpiracion, String codigoDeBarras,
        Categoria clasificacionCategoria
    ) {
        this.nombre = nombre;
        this.costoEnBodega = costoEnBodega;
        this.presentacion = presentacion;
        this.cantidadPresentacion = cantidadPresentacion;
        this.unidadMedida = unidadMedida;
        this.volumenEmpaque = volumenEmpaque;
        this.pesoEmpaque = pesoEmpaque;
        this.fechaExpiracion = fechaExpiracion;
        this.codigoDeBarras = codigoDeBarras;
        this.clasificacionCategoria = clasificacionCategoria;
    }
```

Clase ProductoEnBodega

La clase ProductoEnBodega mapea la tabla producto_en_bodega en la base de datos y representa la relación entre un producto y una bodega. Usa una llave primaria compuesta, definida en la clase ProductoEnBodegaPK, que incluye las relaciones con las entidades Producto y Bodega. Los atributos adicionales de la clase incluye nivelMinimoReorden, costoPromedio, capacidadAlmacenarProducto y cantidadEnBodega, que describen

aspectos del inventario y almacenamiento de los productos en una bodega específica. Los métodos getters y setters permiten acceder y modificar los atributos de la clase.

La clase ProductoEnBodegaPK mapea la llave primaria compuesta de la tabla producto_en_bodega, que se forma a partir de las relaciones con las entidades Producto y Bodega. Define las asociaciones necesarias entre un producto y una bodega en el sistema, y permite identificarlos conjuntamente como una sola llave compuesta en la tabla producto_en_bodega.

```java
@Entity
@Table(name="producto_en_bodega")
public class ProductoEnBodega {

    //Llave primaria compuesta
    @EmbeddedId //Indica que es una llave primaria compuesta
    private ProductoEnBodegaPK pk;

    //Atributos de la clase
    private Integer nivelMinimoReorden;
    private Double costoPromedio;
    private Integer capacidadAlmacenarProducto;
    private Integer cantidadEnBodega;



    public ProductoEnBodega(Producto idProducto, Bodega idBodega,Integer nivelMinimoReorden, Double costoPromedio,
                            Integer capacidadAlmacenarProducto, Integer cantidadEnBodega)
    {
        this.pk = new ProductoEnBodegaPK(idProducto, idBodega);
        this.nivelMinimoReorden = nivelMinimoReorden;
        this.costoPromedio = costoPromedio;
        this.capacidadAlmacenarProducto = capacidadAlmacenarProducto;
        this.cantidadEnBodega = cantidadEnBodega;
    }

    public ProductoEnBodega()
    {;}
```

```java
//Clase que mapea la llave primaria de la tabla ProductoEnBodega en la base de datos
@Embeddable //Indica que es una llave primaria compuesta
public class ProductoEnBodegaPK implements Serializable{

    //Relacion con la tabla Producto
    @ManyToOne
    @JoinColumn(name="identificador_producto", referencedColumnName="identificador")
    private Producto idProducto;

    //Relacion con la tabla Bodega
    @ManyToOne
    @JoinColumn(name="id_bodega", referencedColumnName="id")
    private Bodega idBodega;

    public ProductoEnBodegaPK(Producto idProducto, Bodega idBodega) {
        super();
        this.idProducto = idProducto;
        this.idBodega = idBodega;
    }
```

Clase ProductoPedido

La clase ProductoPedido mapea la tabla producto_pedido en la base de datos y representa la relación entre un producto y una orden de compra. Utiliza una llave primaria compuesta, definida por la clase ProductoPedidoPK, que incluye las relaciones con las entidades Producto y OrdenDeCompra. Además, contiene el atributo cantidadEnOrden, que indica la cantidad de un producto específico en una orden de compra. Los métodos getters y setters permiten acceder y modificar los atributos de la clase.

La clase ProductoPedidoPK mapea la llave primaria compuesta de la tabla producto_pedido, que está formada por las relaciones entre Producto y OrdenDeCompra. Esta llave permite identificar de manera única un producto dentro de una orden de compra específica. Los métodos getters y setters facilitan la gestión de las relaciones entre las dos entidades.

```java
//Clase que mapea la tabla ProductoPedido en la base de datos
@Entity
@Table(name="producto_pedido")
public class ProductoPedido {

    //Llave primaria compuesta
    @EmbeddedId
    private ProductoPedidoPK pk;

    //Atributos de la clase
    private Integer cantidadEnOrden;

    public ProductoPedido(Producto identificadorProducto, OrdenDeCompra numeroOrdenDeCompra, Integer cantidadEnOrden){
        this.pk = new ProductoPedidoPK(identificadorProducto, numeroOrdenDeCompra);
        this.cantidadEnOrden = cantidadEnOrden;
    }

    public ProductoPedido(){
    ;}
```

```java
//Clase que mapea la llave primaria de la tabla ProductoPedido en la base de datos
@Embeddable
public class ProductoPedidoPK implements Serializable {

    //Relacion con la tabla Producto
    @ManyToOne
    @JoinColumn(name="identificador_producto", referencedColumnName="identificador")
    private Producto identificadorProducto;

    //Relacion con la tabla OrdenDeCompra
    @ManyToOne
    @JoinColumn(name="numero_orden_de_compra", referencedColumnName="numero")
    private OrdenDeCompra numeroOrdenDeCompra;

    public ProductoPedidoPK(Producto identificadorProducto, OrdenDeCompra numeroOrdenDeCompra){
        super();
        this.identificadorProducto = identificadorProducto;
        this.numeroOrdenDeCompra = numeroOrdenDeCompra;
    }
```

Clase ProductoProveedor

La clase ProductoProveedor mapea la tabla producto_proveedor en la base de datos y representa la relación entre un producto y un proveedor. Utiliza una llave primaria compuesta, definida por la clase ProductoProveedorPK, que incluye las relaciones con las entidades Producto y Proveedor. Los métodos getters y setters permiten acceder y modificar la llave primaria compuesta, facilitando la gestión de la relación entre productos y proveedores en la base de datos.

La clase ProductoProveedorPK mapea la llave primaria compuesta de la tabla producto_proveedor, que está formada por las relaciones entre Producto y Proveedor. Esta llave permite identificar de manera única la relación entre un producto y su proveedor. Los métodos getters y setters facilitan la manipulación de las entidades asociadas en la base de datos.

```java
//Clase que mapea la tabla ProductoProveedor en la base de datos
@Entity
@Table(name="producto_proveedor")
public class ProductoProveedor {

    //Llave primaria compuesta
    @EmbeddedId
    private ProductoProveedorPK pk;

    public ProductoProveedor(Producto identificadorProducto, Proveedor nitProveedor){
        this.pk = new ProductoProveedorPK(identificadorProducto, nitProveedor);
    }

    public ProductoProveedor()
    {;}
```

```java
//Clase que mapea la llave primaria de la tabla ProductoProveedor en la base de datos
@Embeddable
public class ProductoProveedorPK implements Serializable{

    //Relacion con la tabla Producto
    @ManyToOne
    @JoinColumn(name="identificador_producto", referencedColumnName = "identificador")
    private Producto identificadorProducto;

    //Relacion con la tabla Proveedor
    @ManyToOne
    @JoinColumn(name="nit_proveedor", referencedColumnName = "nit")
    private Proveedor nitProveedor;

    public ProductoProveedorPK(Producto identificadorProducto, Proveedor nitProveedor){
        super();
        this.identificadorProducto = identificadorProducto;
        this.nitProveedor = nitProveedor;
    }
```

Clase Proveedor

La clase Proveedor mapea la tabla proveedor en la base de datos y representa a los proveedores que pueden asociarse a productos. El atributo nit es la llave primaria, definida por el usuario. Los demás atributos incluyen nombre, dirección, nombre de la persona de contacto y el teléfono de contacto del proveedor. Los métodos getters y setters permiten acceder y modificar los valores de estos atributos, facilitando la gestión de los proveedores en la base de datos.

```java
//Clase que mapea la tabla Proveedor en la base de datos
@Entity
@Table(name="proveedor")
public class Proveedor {

    //Atributos de la clase
    @Id
    private String nit; //Indica que es la llave primaria asignada por el usuario

    //Atributos de la clase
    private String nombre;
    private String direccion;
    private String nombrePersonaContacto;
    private String telefonoPersonaContacto;

    public Proveedor(String nit, String nombre, String direccion,
                     String nombrePersonaContacto,
                     String telefonoPersonaContacto)
    {
        this.nit = nit;
        this.nombre = nombre;
        this.direccion = direccion;
        this.nombrePersonaContacto = nombrePersonaContacto;
        this.telefonoPersonaContacto = telefonoPersonaContacto;
    }

    public Proveedor()
    {;}
```

Clase Sucursal

La clase Sucursal mapea la tabla sucursal en la base de datos y representa una sucursal con atributos como id, nombre, tamaño, dirección y teléfono. El atributo id es la llave primaria y se genera automáticamente usando JPA. La clase también tiene una relación de muchos a uno con la entidad Ciudad, lo que significa que cada sucursal está asociada a una ciudad. Los métodos getters y setters permiten acceder y modificar los valores de los atributos, facilitando la gestión de las sucursales en la base de datos.

```java
//Clase que mapea la tabla Sucursal en la base de datos
@Entity
@Table(name="sucursal")
public class Sucursal {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO) //Genera automaticamente el valor de la llave primaria
    private Integer id; //Indica que es la llave primaria

    //Atributos de la clase
    private String nombre;
    private Double tamanio;
    private String direccion;
    private String telefono;

    //Relacion con la tabla Ciudad
    @ManyToOne
    @JoinColumn(name="codigo_ciudad", referencedColumnName="codigo")
    private Ciudad codigoCiudad;

    public Sucursal(String nombre, Double tamanio, String direccion,
                    String telefono, Ciudad codigoCiudad)
    {
        this.nombre = nombre;
        this.tamanio = tamanio;
        this.direccion = direccion;
        this.telefono = telefono;
        this.codigoCiudad = codigoCiudad;
    }

    public Sucursal()
    {;}
```

Sobre los repositorios

A continuación se presenta la estructura general de los repositorios implementados. La misma estructura se aplicó para todas las entidades mencionadas previamente. Solo se detallarán las consultas avanzadas en las que haya aspectos importantes a destacar.

Las consultas CRUD en los repositorios siguen una estructura estándar:

1. **Lectura (SELECT):** Usan @Query con nativeQuery = true y @Param para seleccionar registros.
2. **Inserción (INSERT):** Utilizan @Modifying y @Transactional, con INSERT INTO tabla (columnas) VALUES (:valores).
3. **Actualización (UPDATE):** También usan @Modifying y @Transactional, actualizando registros con UPDATE tabla SET columna = :valor WHERE columna = :parámetro.
4. **Eliminación (DELETE):** Siguen el formato DELETE FROM tabla WHERE columna = :parámetro, modificando la base de datos.

Ejemplo de estructura:

```
//Consulta CRUD de darCiudades
@Query(value = "SELECT * FROM ciudad", nativeQuery=true)
Collection<Ciudad> darCiudades();

//Consulta CRUD de darCiudad
@Query(value = "SELECT * FROM ciudad WHERE codigo= :codigo", nativeQuery=true)
Ciudad darCiudad(@Param("codigo") int id);

//Consulta CRUD de insertarCiudad
@Modifying //Indica que se va a realizar una modificacion en la base de datos
@Transactional //Indica que es una transaccion
@Query(value = "INSERT INTO ciudad(codigo, nombre) VALUES(secuencia_ciudad.nextval, :nombre)", nativeQuery = true)
void insertarCiudad(@Param("nombre") String nombre);
```

Requerimientos importantes

RFC1: Esta consulta avanzada calcula el porcentaje de ocupación de las bodegas de una sucursal específica, basado en la cantidad de productos almacenados. Utiliza un inner join entre las tablas Bodega, Producto_En_Bodega y Producto. Filtra por la sucursal utilizando el parámetro idSucursalU y por una lista de productos con listaProductosU, agrupando los resultados por bodega. El resultado devuelve el id, nombre de la bodega y el porcentaje de ocupación.

```
//Consulta Avanzada No. 1
//Consulta que permite obtener la ocupacion de las bodegas de una sucursal dados unos productos
@Query(value = "SELECT Bodega.id, Bodega.nombre, (SUM(Producto_En_Bodega.cantidad_En_Bodega) / Bodega.capacidad) AS porcentajeOcupacion FROM Bodega\r\n" +
"INNER JOIN Producto_En_Bodega ON Producto_En_Bodega.id_Bodega = Bodega.id\r\n" +
"INNER JOIN Producto ON Producto.identificador = Producto_En_Bodega.identificador_Producto\r\n" +
"WHERE Bodega.id_sucursal = :idSucursalU\r\n" +
"AND Producto.identificador IN :listaProductosU\r\n" +
"GROUP BY Bodega.id, Bodega.nombre, Bodega.capacidad", nativeQuery = true)
Collection<Object[]> obtenerOcupacionBodegas(@Param("idSucursalU") Integer idSucursalU, @Param("listaProductosU") Collection<Integer> listaProductosU);
```

RFC2: Esta consulta filtra productos según varios criterios opcionales: rango de precio, fecha de expiración, sucursal y categoría. Utiliza varios INNER JOIN entre las tablas Producto, Producto_En_Bodega, Bodega, Sucursal y Categoria. Si un parámetro es nulo, no se aplica el filtro correspondiente. Devuelve los productos que cumplen con las condiciones dadas.

```
//Consulta Avanzada No. 2
//Consulta que permite obtener los productos que cumplen con los filtros dados
@Query(value = "SELECT DISTINCT Producto.* FROM Producto\r\n " +
"INNER JOIN Producto_En_Bodega ON Producto_En_Bodega.identificador_Producto = Producto.identificador\r\n " +
"INNER JOIN Bodega ON Bodega.id = Producto_En_Bodega.id_Bodega\r\n " +
"INNER JOIN Sucursal ON Sucursal.id = Bodega.id_sucursal\r\n " +
"INNER JOIN Categoria ON Categoria.codigo = Producto.clasificacion_categoria\r\n " +
"WHERE (:precioMinU IS NULL OR Producto.costo_en_bodega >= :precioMinU)\r\n " +
"AND (:precioMaxU IS NULL OR Producto.costo_en_bodega <= :precioMaxU)\r\n " +
"AND (:fechaSuperiorU IS NULL OR Producto.fecha_expiracion < TO_DATE(:fechaSuperiorU, 'YYYY-MM-DD'))\r\n " +
"AND (:fechaInferiorU IS NULL OR Producto.fecha_expiracion > TO_DATE(:fechaInferiorU, 'YYYY-MM-DD'))\r\n " +
"AND (:sucursalIdU IS NULL OR Bodega.id_sucursal = :sucursalIdU)\r\n " +
"AND (:categoriaNombreU IS NULL OR Categoria.nombre = :categoriaNombreU)",
nativeQuery = true)
Collection<Producto> darProductosFiltrados(@Param("precioMinU") Double precioMinU, @Param("precioMaxU") Double precioMaxU,
```

RFC3: Esta consulta obtiene la información de los productos almacenados en una bodega específica de una sucursal dada. Realiza varios INNER JOIN entre las tablas Producto,

Producto_En_Bodega, Bodega y Sucursal, filtrando por el ID de la sucursal y el ID de la bodega. Devuelve datos como el identificador, nombre, nivel mínimo de reorden, costo promedio y cantidad en bodega de los productos.

```
//Consulta Avanzada No. 3
//Consulta que permite obtener los productos y su informacion en una bodega dada de una sucursal dada
@Query(value = "SELECT PRODUCTO.IDENTIFICADOR, PRODUCTO.NOMBRE, PRODUCTO_EN_BODEGA.NIVEL_MINIMO_REORDEN,\r\n" +
"PRODUCTO_EN_BODEGA.COSTO_PROMEDIO, PRODUCTO_EN_BODEGA.CANTIDAD_EN_BODEGA\r\n" +
"FROM PRODUCTO\r\n" +
"INNER JOIN PRODUCTO_EN_BODEGA ON PRODUCTO_EN_BODEGA.IDENTIFICADOR_PRODUCTO = PRODUCTO.IDENTIFICADOR\r\n" +
"INNER JOIN BODEGA ON BODEGA.ID = PRODUCTO_EN_BODEGA.ID_BODEGA\r\n" +
"INNER JOIN SUCURSAL ON SUCURSAL.ID = BODEGA.ID_SUCURSAL\r\n" +
"WHERE SUCURSAL.ID =:idSucursal\r\n" +
"AND BODEGA.id = :idBodega", nativeQuery = true)
Collection<Object[]> darProductosBodega(@Param("idSucursal") Integer idSucursal, @Param("idBodega") Integer idBodega);
```

RFC4: Estas tres consultas avanzadas tienen como objetivo obtener las sucursales que tienen en bodega un producto específico y que además tienen existencias disponibles, es decir, con una cantidad en bodega mayor a cero. Para ello, cada consulta utiliza varias relaciones inner join entre las tablas Sucursal, Bodega, Producto_En_Bodega y Producto. Esto permite conectar la información de la sucursal con la bodega y finalmente con los productos almacenados en dicha bodega.

La consulta **darSucursalesConProducto** busca las sucursales filtrando por el identificador o por el nombre del producto. Si un producto se encuentra en una bodega de la sucursal, ya sea por su id o por su nombre, y tiene existencias, la consulta devolverá esa sucursal.

La consulta **darSucursalesConProductoIdentificador** es más específica y solo filtra por el identificador del producto. Esta consulta es útil cuando se busca un producto único y conocido por su id dentro del sistema.

Finalmente, la consulta **darSucursalesConProductoNombre** se enfoca en buscar productos por su nombre. Si el producto con el nombre indicado está almacenado en una bodega con existencias, la sucursal correspondiente se devolverá como resultado. Esta consulta es útil cuando no se tiene el identificador exacto del producto, pero sí su nombre.

```
//Consulta Avanzada No. 4
//Consulta que retorna las sucursales que tienen un cierto producto en bodega buscandolo por identificador o nombre
@Query(value = "SELECT DISTINCT Sucursal.*\r\n" +
    "FROM Sucursal INNER JOIN Bodega ON Sucursal.id = Bodega.id_sucursal\r\n" +
    "INNER JOIN Producto_En_Bodega ON Bodega.id = Producto_En_Bodega.id_bodega\r\n" +
    "INNER JOIN Producto ON Producto_En_Bodega.identificador_producto = Producto.identificador\r\n" +
    "WHERE (Producto.identificador = :idProductoU OR Producto.nombre = :nombreProductoU)\r\n" +
    "AND Producto_En_Bodega.cantidad_En_Bodega > 0", nativeQuery=true)
Collection<Sucursal> darSucursalesConProducto(@Param("idProductoU") int idProductoU, @Param("nombreProductoU") String nombreProductoU);

//Consulta Avanzada No. 4
//Consulta que retorna las sucursales que tienen un cierto producto en bodega buscandolo por identificador
@Query(value = "SELECT DISTINCT Sucursal.*\r\n" +
    "FROM Sucursal INNER JOIN Bodega ON Sucursal.id = Bodega.id_sucursal\r\n" +
    "INNER JOIN Producto_En_Bodega ON Bodega.id = Producto_En_Bodega.id_bodega\r\n" +
    "INNER JOIN Producto ON Producto_En_Bodega.identificador_producto = Producto.identificador\r\n" +
    "WHERE Producto.identificador = :idProductoU\r\n" +
    "AND Producto_En_Bodega.cantidad_En_Bodega > 0", nativeQuery=true)
Collection<Sucursal> darSucursalesConProductoIdentificador(@Param("idProductoU") int idProductoU);

//Consulta Avanzada No. 4
//Consulta que retorna las sucursales que tienen un cierto producto en bodega buscandolo por nombre
@Query(value = "SELECT DISTINCT Sucursal.*\r\n" +
    "FROM Sucursal INNER JOIN Bodega ON Sucursal.id = Bodega.id_sucursal\r\n" +
    "INNER JOIN Producto_En_Bodega ON Bodega.id = Producto_En_Bodega.id_bodega\r\n" +
    "INNER JOIN Producto ON Producto_En_Bodega.identificador_producto = Producto.identificador\r\n" +
    "WHERE Producto.nombre = :nombreProductoU\r\n" +
    "AND Producto_En_Bodega.cantidad_En_Bodega > 0", nativeQuery=true)
Collection<Sucursal> darSucursalesConProductoNombre(@Param("nombreProductoU") String nombreProductoU);
```

RFC5: La consulta incluye un filtro para seleccionar solo los productos cuya cantidad en bodega es menor al nivel mínimo de reorden, asegurándose de que se retornen únicamente aquellos productos que necesitan ser reabastecidos. Los resultados se ordenan alfabéticamente por el nombre del producto y luego por el nombre de la bodega.

```
//Consulta Avanzada No. 5
//Consulta que permite obtener los productos que estan por debajo del nivel de reorden
@Query(value = "SELECT PRODUCTO.IDENTIFICADOR, PRODUCTO.NOMBRE, BODEGA.NOMBRE AS BODEGA_NOMBRE,\r\n" +
"SUCURSAL.NOMBRE AS SUCURSAL_NOMBRE, PROVEEDOR.NOMBRE AS PROVEEDOR_NOMBRE, PRODUCTO_EN_BODEGA.CANTIDAD_EN_BODEGA, PRODUCTO_EN_BODEGA.NIVEL_MINIMO_REORDEN \r\n" +
"FROM PRODUCTO \r\n" +
"INNER JOIN PRODUCTO_EN_BODEGA ON PRODUCTO.IDENTIFICADOR = PRODUCTO_EN_BODEGA.IDENTIFICADOR_PRODUCTO \r\n" +
"INNER JOIN BODEGA ON PRODUCTO_EN_BODEGA.ID_BODEGA = BODEGA.ID \r\n" +
"INNER JOIN SUCURSAL ON BODEGA.ID_SUCURSAL = SUCURSAL.ID \r\n" +
"INNER JOIN PRODUCTO_PROVEEDOR ON PRODUCTO.IDENTIFICADOR = PRODUCTO_PROVEEDOR.IDENTIFICADOR_PRODUCTO \r\n" +
"INNER JOIN PROVEEDOR ON PRODUCTO_PROVEEDOR.NIT_PROVEEDOR = PROVEEDOR.NIT \r\n" +
"WHERE PRODUCTO_EN_BODEGA.CANTIDAD_EN_BODEGA < PRODUCTO_EN_BODEGA.NIVEL_MINIMO_REORDEN \r\n" +
"ORDER BY PRODUCTO.NOMBRE, BODEGA.NOMBRE", nativeQuery = true)
Collection<Object[]> obtenerProductosBajoNivelReorden();
```

Sobre los controllers

En este proyecto, los controllers gestionan las solicitudes HTTP que interactúan con las entidades del sistema, permitiendo realizar operaciones CRUD como crear, leer, actualizar y eliminar. Actúan como intermediarios entre el cliente y la base de datos, recibiendo las peticiones, validando la información, y utilizando los repositorios para acceder o modificar los datos. Además, los controllers aseguran que las respuestas se envíen correctamente al cliente, usualmente en formato JSON, gestionando errores y proporcionando el estado adecuado en cada respuesta.

La estructura de una petición en un controller sigue un patrón definido, comenzando con una anotación HTTP como @PostMapping, @GetMapping, @PutMapping, o @DeleteMapping. Estas anotaciones definen el tipo de operación HTTP que manejará el método (crear, leer, actualizar o eliminar) y la ruta específica a la que el cliente debe hacer la solicitud. Por ejemplo, en el método bodegaGuardar, se usa @PostMapping("/bodegas/new/save") para procesar una solicitud POST a la URL que crea una nueva bodega.

El método puede recibir parámetros de la solicitud de varias formas. Por ejemplo @RequestBody captura un objeto JSON enviado en el cuerpo de la solicitud, que luego se convierte en una entidad del sistema, como una Bodega. Alternativamente, se pueden usar @PathVariable para obtener parámetros directamente de la URL o @RequestParam para capturar parámetros desde la query string.

Una vez recibidos los datos, el método realiza la lógica necesaria, como interactuar con el repositorio correspondiente para consultar, modificar o almacenar información en la base de datos. En el ejemplo de bodegaGuardar, el método llama a bodegaRepository.insertarBodega() para insertar los datos de la nueva bodega en la base de datos. Este paso garantiza que se ejecuten las operaciones adecuadas de acuerdo con la lógica de negocio definida.

Finalmente, el método responde al cliente con un ResponseEntity, que incluye tanto el resultado de la operación como el código de estado HTTP correspondiente. Por ejemplo, si la inserción es exitosa, el método devuelve un mensaje de éxito y el estado 201 (Created). Si ocurre un error, responde con un estado 500 (Internal Server Error). Esta estructura asegura que el cliente reciba una respuesta clara sobre el éxito o fallo de la operación.

```java
//Metodo que se encarga de crear una bodega
@PostMapping("/bodegas/new/save") //Indica que el metodo se activa cuando se hace una peticion POST a la URL /bodegas/new/save
public ResponseEntity<String> bodegaGuardar(@RequestBody Bodega bodega){
    try{
        //Insertar la bodega en la base de datos
        bodegaRepository.insertarBodega(bodega.getNombre(), bodega.getTamanio(), bodega.getCapacidad(), bodega.getIdSucursal().getId());
        return new ResponseEntity<>(body:"Bodega creada exitosamente", HttpStatus.CREATED);
    }
    catch (Exception e){
        return new ResponseEntity<>(body:"Error al crear la bodega", HttpStatus.INTERNAL_SERVER_ERROR);
    }
```

Es de aclarar que en algunos requerimientos avanzados tuvimos que usar ArrayList para poder instanciar una Collection y lograr llenarla de objetos de tipo HashMap, así podíamos retornar toda la información solicitada en las peticiones. Se deja como ejemplo el requerimiento de consulta número cuatro:

```java
//Para RCF4:
//Metodo que se encarga de obtener los productos en una bodega dada de una sucursal dada
@GetMapping("/productos/productosEnBodega")
public Collection<Map<String, Object>> obtenerProductosEnBodega(
        @RequestParam Integer idSucursal,
        @RequestParam Integer idBodega) {
    Collection<Object[]> resultado = productoRepository.darProductosBodega(idSucursal, idBodega);

    // Convertir los resultados en una colección de Map<String, Object> para mayor claridad en el retorno.
    Collection<Map<String, Object>> listaproductos = new ArrayList<>();
    for (Object[] fila : resultado) {
        Map<String, Object> productos = new HashMap<>();
        productos.put("id", fila[0]);
        productos.put("nombre", fila[1]);
        productos.put("nivelMinimoReorden", fila[2]);
        productos.put("costoPromedio", fila[3]);
        productos.put("cantidadEnBodega", fila[4]);
        listaproductos.add(productos);
    }
    return listaproductos;
}
```

Adicionalmente, es importante aclarar que para ciertos requerimientos se manejaron las restricciones de integridad desde el controller para desarrollar las pruebas de Postman. Por ejemplo el check de NC para actualizar un producto y que el usuario no modifique su fecha de Entrega se realizó de la siguiente manera:

```java
    // Metodo que se encarga de editar un producto y lanza error si el usuario intenta modificar la fecha de expiración
    @PostMapping("/productos/{identificador}/edit/save")
    public ResponseEntity<String> productoEditarGuardar(@PathVariable("identificador") Integer identificador, @RequestBody Producto producto) {
    try {
        // Verificar si el producto con el identificador existe
        if (!productoRepository.existeProducto(identificador)) {
            return new ResponseEntity<>("El producto con identificador " + identificador + " no existe", HttpStatus.NOT_FOUND);
        }

        // Verificar que ninguno de los campos obligatorios sea null (sin verificar fecha de expiración)
        if (producto.getNombre() == null || producto.getCostoEnBodega() == null || producto.getPresentacion() == null ||
            producto.getCantidadPresentacion() == null || producto.getUnidadMedida() == null ||
            producto.getVolumenEmpaque() == null || producto.getPesoEmpaque() == null || producto.getCodigoDeBarras() == null ||
            producto.getClasificacionCategoria() == null || producto.getClasificacionCategoria().getCodigo() == null) {

            return new ResponseEntity<>(body:"Uno o más campos obligatorios están vacíos o nulos.", HttpStatus.BAD_REQUEST);
        }

        // Obtener el producto actual desde la base de datos
        Producto productoActual = productoRepository.darProducto(identificador);
        if (productoActual == null) {
            return new ResponseEntity<>("El producto con identificador " + identificador + " no existe", HttpStatus.NOT_FOUND);
        }

        // Verificar si el usuario está intentando modificar la fecha de expiración
        if (producto.getFechaExpiracion() != null && !producto.getFechaExpiracion().equals(productoActual.getFechaExpiracion())) {
            return new ResponseEntity<>(body:"No está permitido modificar la fecha de expiración del producto.", HttpStatus.BAD_REQUEST);
        }

        // Actualizar el producto sin modificar la fecha de expiración
        productoRepository.actualizarProducto(identificador, producto.getNombre(), producto.getCostoEnBodega(),
                                              producto.getPresentacion(), producto.getCantidadPresentacion(),
                                              producto.getUnidadMedida(), producto.getVolumenEmpaque(),
                                              producto.getPesoEmpaque(), productoActual.getFechaExpiracion(),
                                              producto.getCodigoDeBarras(), producto.getClasificacionCategoria().getCodigo());

        return new ResponseEntity<>(body:"Producto actualizado exitosamente", HttpStatus.OK);
    } catch (Exception e) {
        return new ResponseEntity<>(body:"Error al actualizar el producto", HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```