

UNIVERSIDAD DE LAS FUERZAS ARMADAS-ESPE SEDE SANTO DOMINGO

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN - DCCO-SS

CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

PERIODO : noviembre 2023 – marzo 2024

ASIGNATURA : Aplicaciones Distribuidas

TEMA : Backend Ventas OnLine

NOMBRES : Wilmer Mesias Shagñay Condo

NIVEL-PARALELO : Séptimo

DOCENTE : Ing. Luis Chica Moncayo MSc

FECHA DE ENTREGA : 27/11/2023

SANTO DOMINGO - ECUADOR

2023

Índice de contenido

1.	Introducción	5
2.	Sistemas de Objetivos	5
2.1.	Objetivo General	5
2.2.	Objetivo Especifico	5
3.	Desarrollo.....	6
3.1.	Estructura del Proyecto:	7
	Modelo:	7
	Repositorios:	8
	Servicios:.....	12
	Controladores:	15
3.2.	Resultados	20
	Almacén	21
	Cliente	23
	Producto	25
	Orden.....	28
4.	Conclusiones	31
5.	Recomendaciones	31
6.	Referencias.....	32

Índice de figuras

Figura 1 Diagrama de las tablas.....	6
Figura 2 Estructura MVC	6
Figura 3 Clase Orden	7
Figura 4 Clase Cliente.....	7
Figura 5 Clase Almacen.....	8
Figura 6 Clase Producto.....	8
Figura 7 Interfaz del repositorio	9
Figura 8 Clase Repositorio Orden	9
Figura 9 Clase Cliente Repositorio.....	10
Figura 10 Clase Almacén Repositorio	11
Figura 11 Clase Productos Repositorio	12
Figura 12 Clase Orden Servicio.....	13
Figura 13 Clase Cliente Servicio	13
Figura 14 Clase Almacen Servicio	14
Figura 15 Clase Producto Servicio	14
Figura 16 Controlador Ordenes declaraciones de servicios.....	15
Figura 17 Controlador Orden método registrar	15
Figura 18 Controlador Ordenes método listar ordenes	16
Figura 19 Controlador Cliente método clientes, buscar cliente, agregar.....	16
Figura 20 Controlador Cliente métodos eliminar y actualizar.....	17
Figura 21 Controlador Almacén métodos eliminar y actualizar.....	17
Figura 22 Controlador Almacén métodos almacenes, buscar por id y agregar	18
Figura 23 Controlador Producto métodos lista, agregar	19
Figura 24 Controlador Producto métodos eliminar, buscar por id, actualizar	19

Figura 25	Lista de carpetas del postman.....	20
Figura 26	Postman Agregar Almacén.....	21
Figura 27	Postman Listar almacenes	21
Figura 28	Postman Buscar almacén por ID	22
Figura 29	Postman Actualizar almacén	22
Figura 30	Postman Eliminar almacén por id	22
Figura 31	Postman listar almenen para verificar actualización y eliminación	23
Figura 32	Postman Agregar cliente	23
Figura 33	Postman listar clientes	24
Figura 34	Postman Buscar cliente por id	24
Figura 35	Postman actualizar cliente	24
Figura 36	Postman Eliminar cliente.....	25
Figura 37	Postman listar clientes para verificar eliminación y actualización.....	25
Figura 38	Postman agregar producto	25
Figura 39	Postman listar los productos.....	26
Figura 40	Postman buscar producto por id	26
Figura 41	Postman Actualizar producto	27
Figura 42	Postman Eliminar producto	27
Figura 43	Postman listar productos verificación de eliminar y actualizar.....	27
Figura 44	Postman registrar orden.....	28
Figura 45	Postman listar ordenes	28
Figura 46	Postman agrega otro producto con el mismo id orden	29
Figura 47	Postman listar ordenes	29
Figura 48	Postman registrar otra orden.....	30
Figura 49	Postman listar ordenes verificar	30

1. Introducción

En el siguiente informe se describe la implementación de una aplicación utilizando el framework Spring Boot la estructura sigue el patrón Modelo-Servicio-Repositorio-Controlador (MSRC), donde la capa de servicios maneja la lógica de negocio, la capa de controladores maneja las interacciones con el usuario y las solicitudes HTTP, y la capa de repositorio maneja el acceso a los datos y las operaciones de persistencia. Este patrón es similar al MVC, pero utiliza terminología diferente para reflejar una separación clara de responsabilidades. La aplicación gestionará información relacionada con órdenes, clientes, almacenes y productos.

2. Sistemas de Objetivos

2.1. Objetivo General

- Realizar la implementación de backend para ventas online.

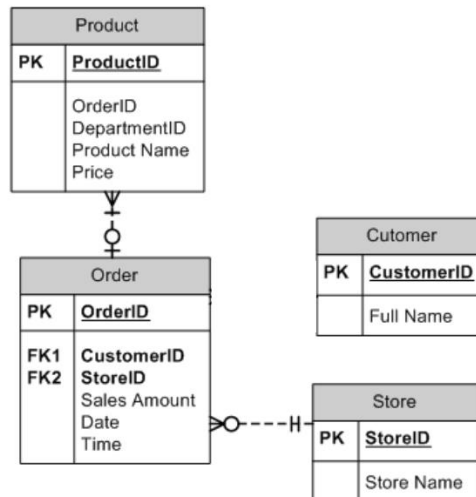
2.2. Objetivo Especifico

- Realizar Rest Api que exponga sus servicios CRUD con arquitectura N-Capas.
- Realizar pruebas de los servicios con postman.

3. Desarrollo

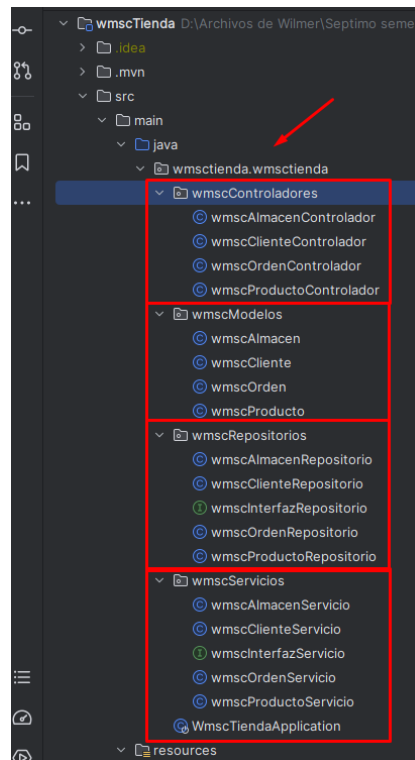
En el diagrama se visualiza las tablas para realizar un backend que exponga sus servicios CRUD. (ver Figura 1).

Figura 1 Diagrama de las tablas



En la **figura 2**, se muestra la estructura del proyecto con el modelo, repositorio, servicios y controlador en donde esta realiza las peticiones http.

Figura 2 Estructura MVC



3.1. Estructura del Proyecto:

La estructura del proyecto se organiza en la ruta "src/main/java/wmsctienda.wmsctienda/wmscModelo". Como se muestra en la **figura 2**, se describen las principales componentes en cada capa:

Modelo:

En el patrón MSPC, el "Modelo" representa la capa que maneja la lógica de negocio y los datos de la aplicación.

Ruta: **src/main/java/wmsctienda.wmsctienda/wmscModelo** y las que tiene las siguientes clases, orden, cliente, almacén y producto.

WmscOrden: Representa la entidad Orden con atributos como ordenid, clienteID, almacenID, cantidad, fecha, tiempo. Como se puede observar la **figura 3**.

Figura 3 Clase Orden

```
17 usages 4 wmierrxx
9  @Data
10  @AllArgsConstructor
11  @NoArgsConstructor
12  public class wmscOrden {
13      private int wmscOrderID;
14      private int wmscClienteID;
15      private int wmscAlmacenID;
16      private int wmscCantidadVentas;
17      private String wmscFechaIngreso;
18      private String wmscTiempo;
19      private List<wmscProducto> wmscProductosList;
20      private wmscCliente wmscCliente;
21      private wmscAlmacen wmscAlmacen;
22  }
```

WmscCliente: Representa la entidad Cliente con sus atributos clienteID y nombresCompleto como se muestra en la **figura 4**.

Figura 4 Clase Cliente

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class wmscCliente {
    private int wmscClienteID;
    private String wmscNombresCompleto;
}
```

WmscAlmacen: Representa la entidad Almacén con sus atributos almacenID y nombreAlmacen como se muestra en la **figura 5**.

Figura 5 Clase Almacen

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class wmscAlmacen {
    private int wmscAlmacenID;
    private String wmscNombreAlmacen;
}
```

WmscProducto: Representa la entidad Producto con atributos como productoID, DepartamentoID, nombre, precio. Como se muestra la **figura 6**.

Figura 6 Clase Producto

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class wmscProducto {
    private int wmscProductoID;
    private int wmscDepartamentoID;
    private String wmscNombre;
    private double wmscPrecio;
}
```

Repositorios:

En MSC, la capa de "Repositorio" se encarga del acceso a los datos y de las operaciones de persistencia al utilizar el decorador @Repository, que se aplica al principio de la clase. Ruta:

src/main/java/wmsctienda.wmsctienda/wmscModelo/repositorio

wmscInterfazRepositorio; en esta clase se crea los métodos que heredan cada clase del repositorio los métodos creados como se muestran en la **figura 7**, se puede ver los tres métodos que es agregar, eliminar y actualizar que se reutilizara en cada clase repositorio.

Figura 7 Interfaz del repositorio

```
java  wmscProducto.java  wmscInterfazRepositorio.java
package wmsctienda.wmsctienda.wmscRepositorios;
4 usages 4 implementations 1 wilmerxx
public interface wmscInterfazRepositorio {
    4 implementations 1 wilmerxx
    public void wmscAgregar(Object wmscObj);
    //public Object wmscBuscarPorID(int wmscID);
    4 implementations 1 wilmerxx
    public void wmscEliminarPorID(int wmscID);

    4 implementations 1 wilmerxx
    public void wmscActualizarPorId(Object wmscObj);
}
}
```

WmscOrdenRepositorio: Interfaz que extiende wmscInterfazRepositorio para operaciones de persistencia relacionadas con la entidad Orden de igual manera cuenta con sus métodos propios como se puede ver en la **figura 8**, el método de lista de órdenes, agregar, y buscar por el id de la orden y como se debe implementar todos los métodos en este caso solo se pide que se registre.

Figura 8 Clase Repositorio Orden

```
* wilmerxx *
@Repository
public class wmscOrdenRepositorio implements wmscInterfazRepositorio{
    3 usages
    List<wmscOrden> wmscOrdenList = new ArrayList<>();
    1 usage 1 wilmerxx
    public List<wmscOrden> wmscOrdenLista(){ return this.wmscOrdenList;}
    1 wilmerxx
    @Override
    public void wmscAgregar(Object wmscObj) {
        wmscOrden wmscOrde = (wmscOrden) wmscObj;
        wmscOrdenList.add(wmscOrde);
    }
    1 wilmerxx
    public wmscOrden wmscBuscarPorID(int wmscID) {
        wmscOrden wmscOrde = null;
        for (wmscOrden wmscOrdenBuscado:wmscOrdenList) {
            if(wmscOrdenBuscado.getWmscOrderID() == wmscID){
                wmscOrde = wmscOrdenBuscado;
            }
        }
        return wmscOrde;
    }
    1 wilmerxx
    @Override
    public void wmscEliminarPorID(int wmscID) {
    }
    1 wilmerxx
    @Override
    public void wmscActualizarPorId(Object wmscObj) {
    }
}
```

WmscClienteRepositorio: Interfaz que extiende wmscInterfazRepositorio para operaciones de persistencia relacionadas con la entidad Cliente. En la **figura 9**, se muestra cada método son su implementación de su funcionamiento.

Figura 9 Clase Cliente Repositorio

```
@Repository
public class WmscClienteRepositorio implements WmscInterfazRepositorio {
    4 usages
    List<WmscCliente> wmscClienteList = new ArrayList<>();

    1 usage
    public List<WmscCliente> wmscClienteListas() { return wmscClienteList; }

    @Override
    public void wmscAgregar(Object wmscObj) {
        WmscCliente wmscClient = (WmscCliente) wmscObj;
        wmscClienteList.add(wmscClient);
    }

    public WmscCliente wmscBuscarPorID(int wmscID) {
        WmscCliente wmscClientBuscado = null;
        for (WmscCliente wmscClient:wmscClienteList) {
            if(wmscClient.getWmscClienteID() == wmscID){wmscClientBuscado = wmscClient;}
        }
        return wmscClientBuscado;
    }

    public void wmscEliminarPorID(int wmscID) {
        WmscCliente wmscClient = wmscBuscarPorID(wmscID);
        if(wmscClient.getWmscClienteID() == wmscID){wmscClienteList.remove(wmscClient);}
    }

    @Override
    public void wmscActualizarPorId(Object wmscObj) {
        WmscCliente wmscClient = (WmscCliente) wmscObj;
        WmscCliente wmscClienteBuscado = wmscBuscarPorID(wmscClient.getWmscClienteID());
        if(!Objects.isNull(wmscClienteBuscado)){
            wmscClienteBuscado.setWmscClienteID(wmscClient.getWmscClienteID());
            wmscClienteBuscado.setWmscNombresCompleto(wmscClient.getWmscNombresCompleto());
        }
    }
}
```

WmscAlmacenRepositorio: Interfaz que extiende wmscInterfazRepositorio para operaciones de persistencia relacionadas con la entidad Almacén. En la **figura 10**, se muestra los métodos utilizados con su debida implementación.

Figura 10 Clase Almacén Repositorio

```
@Repository
public class WmscAlmacenRepositorio implements WmscInterfazRepositorio {
    4 usages
    List<WmscAlmacen> wmscAlmacenList = new ArrayList<>();
    1 usage - wilmerxx
    > public List<WmscAlmacen> wmscAlmacenListas() { return wmscAlmacenList; }
    3 usages - wilmerxx
    public WmscAlmacen wmscBuscarPorId(int wmscid){
        WmscAlmacen wmscAlmacenBuscado = null;
        for (WmscAlmacen wmscAlmacen : wmscAlmacenList) {
            if(wmscAlmacen.getWmscAlmacenID() == wmscid){ wmscAlmacenBuscado = wmscAlmacen;}
        }
        return wmscAlmacenBuscado;
    }

    @Override
    public void wmscAgregar(Object wmscObj) {
        WmscAlmacen wmscAlmacen = (WmscAlmacen) wmscObj;
        this.wmscAlmacenList.add(wmscAlmacen);
    }

    @Override
    public void wmscEliminarPorID(int wmscID) {
        WmscAlmacen wmscAlmacen = wmscBuscarPorId(wmscID);
        if(wmscAlmacen.getWmscAlmacenID() == wmscID){wmscAlmacenList.remove(wmscAlmacen);}
    }

    @Override
    public void wmscActualizarPorId(Object wmscObj) {
        WmscAlmacen wmscAlmacen = (WmscAlmacen) wmscObj;
        WmscAlmacen wmscAlmacenBuscado = wmscBuscarPorId(wmscAlmacen.getWmscAlmacenID());
        if(!Objects.isNull(wmscAlmacenBuscado)){
            wmscAlmacenBuscado.setWmscAlmacenID(wmscAlmacen.getWmscAlmacenID());
            wmscAlmacenBuscado.setWmscNombreAlmacen(wmscAlmacen.getWmscNombreAlmacen());
        }
    }
}
```

WmscProductoRepositorio: Interfaz que extiende wmscInterfazRepositorio para operaciones de persistencia relacionadas con la entidad Producto. En la **figura 11**, se muestra la implementación de los métodos.

Figura 11 Clase Productos Repositorio

```
@Repository
public class WmscProductoRepositorio implements WmscInterfazRepositorio {
    4 usages
    private List<WmscProducto> wmscListaProducto = new ArrayList<>();
    1 usage  ⚡ wilmerxx

    public List<WmscProducto> wmscListaProductos() { return this.wmscListaProducto; }
    ⚡ wilmerxx *
    @Override
    public void wmscAgregar(Object wmscObj) {
        WmscProducto wmscProduct = (WmscProducto) wmscObj;
        if(!Objects.isNull(wmscProduct)){this.wmscListaProducto.add(wmscProduct);}
    }
    ⚡ wilmerxx *
    public WmscProducto wmscBuscarPorID(int wmscID) {
        WmscProducto wmscProductBuscado = null;
        for (WmscProducto wmscProduct:wmscListaProducto) {
            if(wmscProduct.getWmscProductoID() == wmscID){wmscProductBuscado = wmscProduct;}
        }
        return wmscProductBuscado;
    }
    ⚡ wilmerxx *
    @Override
    public void wmscEliminarPorID(int wmscID) {
        WmscProducto wmscProduct = wmscBuscarPorID(wmscID);
        if(wmscProduct.getWmscProductoID() == wmscID){wmscListaProducto.remove(wmscProduct);}
    }
    ⚡ wilmerxx
    @Override
    public void wmscActualizarPorId(Object wmscObj) {
        WmscProducto wmscProduct = (WmscProducto) wmscObj;
        WmscProducto wmscProductBuscado = wmscBuscarPorID(wmscProduct.getWmscProductoID());
        if(!Objects.isNull(wmscProductBuscado)){
            wmscProductBuscado.setWmscProductoID(wmscProduct.getWmscProductoID());
            wmscProductBuscado.setWmscPrecio(wmscProduct.getWmscPrecio());
            wmscProductBuscado.setWmscNombre(wmscProduct.getWmscNombre());
            wmscProductBuscado.setWmscDepartamentoID(wmscProduct.getWmscDepartamentoID());
        }
    }
}
```

Servicios:

En las clases de servicios contienen la lógica de negocio,

Ruta: **src/main/java/wmsctienda.wmsctienda/wmscModelo/servicios**

WmscOrdenServicio: Clase que contiene la lógica de negocio relacionada con las órdenes e implementa una interfaz wmscInterfazServicio. En la **figura 12**, se muestra la utilización de los métodos del repositorio en donde se marca con rojo son los métodos que se va a implementar.

Figura 12 Clase Orden Servicio



```
@Service
public class WmscOrdenServicio implements WmscInterfazServicio{

    @Autowired
    WmscOrdenRepositorio WmscOrdenRepositori;

    // usage
    public List<WmscOrden> WmscOrdenLista() { return this.WmscOrdenRepositori.WmscOrdenLista(); }

    // Wilmerxx
    @Override
    public void WmscAgregar(Object WmscObj) { this.WmscOrdenRepositori.WmscAgregar(WmscObj); }

    // @Override
    // Wilmerxx
    public WmscOrden WmscBuscarPorID(int WmscID) { return this.WmscOrdenRepositori.WmscBuscarPorID(WmscID); }

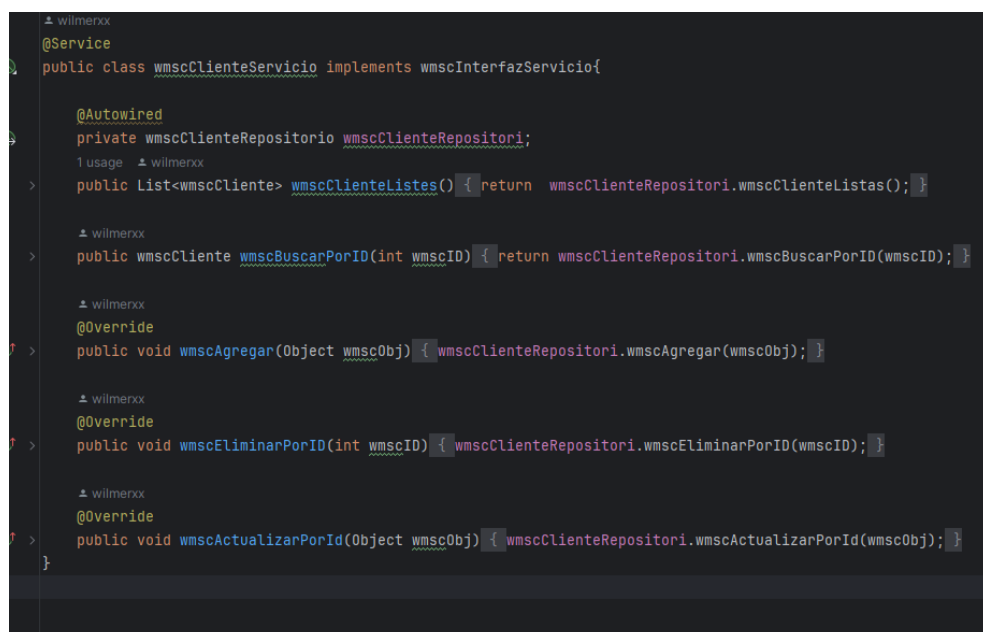
    // Wilmerxx
    @Override
    public void WmscEliminarPorID(int WmscID) { }

    // Wilmerxx
    @Override
    public void WmscActualizarPorId(Object WmscObj) { }
}
```

The screenshot shows the Java code for the `WmscOrdenServicio` class. Red boxes highlight the `@Service` annotation, the `@Autowired` annotation on the `WmscOrdenRepositorio` field, and the `WmscOrdenLista()` and `WmscAgregar()` methods. The `WmscOrdenLista()` method is also annotated with `// usage`.

WmscClienteServicio: Clase que contiene la lógica de negocio relacionada con los clientes e implementa una interfaz wmscInterfazServicio. En la **figura 13**, se muestra la utilización de los métodos del repositorio.

Figura 13 Clase Cliente Servicio



```
WmscClienteServicio
@Service
public class WmscClienteServicio implements WmscInterfazServicio{

    @Autowired
    private WmscClienteRepositorio WmscClienteRepositori;

    // usage
    public List<WmscCliente> WmscClienteListas() { return WmscClienteRepositori.WmscClienteListas(); }

    // Wilmerxx
    public WmscCliente WmscBuscarPorID(int WmscID) { return WmscClienteRepositori.WmscBuscarPorID(WmscID); }

    // Wilmerxx
    @Override
    public void WmscAgregar(Object WmscObj) { WmscClienteRepositori.WmscAgregar(WmscObj); }

    // Wilmerxx
    @Override
    public void WmscEliminarPorID(int WmscID) { WmscClienteRepositori.WmscEliminarPorID(WmscID); }

    // Wilmerxx
    @Override
    public void WmscActualizarPorId(Object WmscObj) { WmscClienteRepositori.WmscActualizarPorId(WmscObj); }
}
```

The screenshot shows the Java code for the `WmscClienteServicio` class. The `@Service` annotation is highlighted. The `WmscClienteListas()` method is annotated with `// usage`. The `WmscBuscarPorID()`, `WmscAgregar()`, `WmscEliminarPorID()`, and `WmscActualizarPorId()` methods are all annotated with `Wilmerxx`.

WmscAlmacenServicio: Clase que contiene la lógica de negocio relacionada con los almacenes e implementa una interfaz `wmscInterfazServicio`. En la **figura 14**, se muestra la utilización de los métodos del repositorio.

Figura 14 Clase Almacen Servicio

```
@Service
public class wmscAlmacenServicio implements wmscInterfazServicio{
    @Autowired
    wmscAlmacenRepositorio wmscAlmacenRepositori;
    1 usage 1 wilmerxx
    > public List<wmscAlmacen> wmscAlmacenListas() { return wmscAlmacenRepositori.wmscAlmacenListas(); }
    5 usages 1 wilmerxx
    > public wmscAlmacen wmscBuscarAlmacen(int wmscid) { return wmscAlmacenRepositori.wmscBuscarPorId(wmscid); }
    1 wilmerxx
    > @Override
    public void wmscAgregar(Object wmscObj) { wmscAlmacenRepositori.wmscAgregar(wmscObj); }
    1 wilmerxx
    > @Override
    public void wmscEliminarPorID(int wmscID) { wmscAlmacenRepositori.wmscEliminarPorID(wmscID); }
    1 wilmerxx
    > @Override
    public void wmscActualizarPorId(Object wmscObj) { wmscAlmacenRepositori.wmscActualizarPorId(wmscObj); }
    }
}
```

WmscProductoServicio: Clase que contiene la lógica de negocio relacionada con los productos e implementa una interfaz `wmscInterfazServicio`. En la **figura 15**, se muestra la utilización de los métodos del repositorio.

Figura 15 Clase Producto Servicio

```
@Service
public class wmscProductoServicio implements wmscInterfazServicio {
    @Autowired
    wmscProductoRepositorio wmscProductoRepo;
    1 usage 1 wilmerxx
    > public List<wmscProducto> wmscProductoList() { return wmscProductoRepo.wmscListaProductos(); }
    1 wilmerxx
    > @Override
    public void wmscAgregar(Object wmscObj) {
        this.wmscProductoRepo.wmscAgregar(wmscObj);
    }
    1 wilmerxx
    > public wmscProducto wmscBuscarPorID(int wmscID) { return wmscProductoRepo.wmscBuscarPorID(wmscID); }
    1 wilmerxx
    > @Override
    public void wmscEliminarPorID(int wmscID) { wmscProductoRepo.wmscEliminarPorID(wmscID); }
    1 wilmerxx
    > @Override
    public void wmscActualizarPorId(Object wmscObj) { wmscProductoRepo.wmscActualizarPorId(wmscObj); }
    }
}
```

Controladores:

Ruta: src/main/java/wmsctienda.wmsctienda/wmscModelo/controladores

WmscOrdenControlador: Controlador que maneja las peticiones HTTP relacionadas con las órdenes en la **figura 16,17,18**, se muestra la funcionalidad de cada funcion.

Figura 16 Controlador Ordenes declaraciones de servicios

```
@RestController
public class WmscOrdenControlador {
    @Autowired
    WmscOrdenServicio WmscOrdenService;
    @Autowired
    WmscProductoServicio WmscProductoService;
    @Autowired
    WmscAlmacenServicio WmscAlmacenService;
    @Autowired
    WmscClienteServicio WmscClienteService;
}
```

Figura 17 Controlador Orden método registrar

```
@PostMapping("/wmscAgregarOrden")
public ResponseEntity<?> WmscAgregarOrden(
    @RequestParam(value = "WmscOrderID") int WmscOrderID,
    @RequestParam(value = "WmscProductoID") int WmscProductoID,
    @RequestParam(value = "WmscClienteID") int WmscClienteID,
    @RequestParam(value = "WmscAlmacenID") int WmscAlmacenID,
    @RequestParam(value = "WmscCantidadVentas") int WmscCantidadVentas,
    @RequestParam(value = "WmscFechaIngreso") String WmscFechaIngreso,
    @RequestParam(value = "WmscTiempo") String WmscTiempo
){
    WmscProducto WmscProduct = WmscProductoService.WmscBuscarPorID(WmscProductoID);
    WmscAlmacen WmscAlmacenBuscado = WmscAlmacenService.WmscBuscarAlmacen(WmscAlmacenID);
    WmscCliente WmscClienteBuscado = WmscClienteService.WmscBuscarPorID(WmscClienteID);
    List<WmscProducto> WmscProductoList = new ArrayList<>();

    if(!Objects.isNull(WmscProduct)){
        WmscOrden WmscOrdenParaActualizar = WmscOrdenService.WmscBuscarPorID(WmscOrderID);
        if(WmscOrdenService.WmscBuscarPorID(WmscOrderID)==null){
            if(!Objects.isNull(WmscAlmacenBuscado)){
                if (!Objects.isNull(WmscClienteBuscado)){
                    WmscProductoList.add(WmscProduct);
                    WmscOrden WmscOrde = new WmscOrden(
                        WmscOrderID,
                        WmscClienteID,
                        WmscAlmacenID,
                        WmscCantidadVentas,
                        WmscFechaIngreso,
                        WmscTiempo,
                        WmscProductoList,
                        WmscClienteBuscado,
                        WmscAlmacenBuscado
                    );
                    WmscOrdenService.WmscAgregar(WmscOrde);
                    return ResponseEntity.ok( body: "Agregado con éxito la orden");
                }else { return ResponseEntity.ok( body: "Cliente no existe");
                }
            }else{ return ResponseEntity.ok( body: "Almacen no existe");
            }
        }else { WmscOrdenParaActualizar.getWmscProductosList().add(WmscProduct);
            return ResponseEntity.ok( body: "Producto nuevo agregado");
        }
    }
}
```

Figura 18 Controlador Ordenes método listar ordenes

```
willmerxx
@GetMapping("/{wmscListaOrdenes}")
public ResponseEntity<?> wmscListaOrdenes(){

    List<wmscOrden> wmscOrdenList = wmscOrdenService.wmscOrdenLista();
    if (!wmscOrdenList.isEmpty()) {
        return ResponseEntity.ok(wmscOrdenList);
    } else {
        return ResponseEntity.ok(Collections.emptyList()); // Devuelve una lista vacia si no hay datos
    }
}
```

WmscClienteControlador: Controlador que maneja las peticiones HTTP relacionadas con los clientes. En la **figura 19 y 20**, se muestra la implementación del controlador.

Figura 19 Controlador Cliente método clientes, buscar cliente, agregar

```
willmerxx
@RestController
public class wmscClienteControlador {

    @Autowired
    private wmscClienteServicio wmscClienteService;

    willmerxx
    @GetMapping("/{wmscClientes}")
    public ResponseEntity<?> wmscClientelistes(){
        List<wmscCliente> wmscClienteList = wmscClienteService.wmscClientelistes();
        if(!wmscClienteList.isEmpty()){
            return ResponseEntity.ok(wmscClienteList);
        }else {
            return ResponseEntity.ok(Collections.emptyList()); //devuelve la lista de los clientes
        }
    }

    willmerxx
    @GetMapping("/{wmscBuscarClientePorID}/{wmscid}")
    public ResponseEntity<?> wmscBuscarPorID(@PathVariable(value = "wmscid") int wmscid) {
        wmscCliente wmscClient = wmscClienteService.wmscBuscarPorID(wmscid);
        if(!Objects.isNull(wmscClient)){
            return ResponseEntity.ok(wmscClient);
        }
        return ResponseEntity.ok( body: "No existe el cliente");
    }

    willmerxx
    @PostMapping("/{wmscAgregarCliente}")
    public ResponseEntity<?> wmscAgregar(
        @RequestParam(value = "wmscClienteID") int wmscClienteID,
        @RequestParam(value = "wmscNombresCompleto") String wmscNombresCompleto
    ) {
        if (Objects.isNull(wmscClienteService.wmscBuscarPorID(wmscClienteID))){
            wmscCliente wmscClient = new wmscCliente(wmscClienteID,wmscNombresCompleto);
            wmscClienteService.wmscAgregar(wmscClient);
            return ResponseEntity.ok(wmscClient);
        }else {
            return ResponseEntity.ok( body: "Ya existe el cliente");
        }
    }
}
```


Figura 20 Controlador Cliente métodos eliminar y actualizar

```
@DeleteMapping(value = "/wmscEliminarCliente/{wmscid}")
public ResponseEntity<> wmscEliminarPorID(@PathVariable(value = "wmscid") int wmscid) {
    if(!Objects.isNull(wmscClienteService.wmscBuscarPorID(wmscid))) {
        wmscClienteService.wmscEliminarPorID(wmscid);
        return ResponseEntity.ok( body: "Cliente eliminado");
    } else {
        return ResponseEntity.ok( body: "No existe cliente");
    }
}

@PutMapping(value = "/wmscActualizarCliente/{wmscid}")
public ResponseEntity<> wmscActualizarPorId(
    @PathVariable(value = "wmscid") int wmscClienteID,
    @RequestParam(value = "wmscNombresCompleto") String wmscNombresCompleto
) {
    if(!Objects.isNull(wmscClienteService.wmscBuscarPorID(wmscClienteID))) {
        wmscCliente wmscClient = new wmscCliente(wmscClienteID, wmscNombresCompleto);
        wmscClienteService.wmscActualizarPorId(wmscClient);
        return ResponseEntity.ok(wmscClient);
    } else {
        return ResponseEntity.ok( body: "No existe producto");
    }
}
```

WmscAlmacenControlador: Controlador que maneja las peticiones HTTP relacionadas con los almacenes. En la **figura 21 y 22**, se muestra los métodos implementados para la realización del CRUD.

Figura 21 Controlador Almacén métodos eliminar y actualizar

```
@DeleteMapping(value = "/wmscEliminarPorID/{wmscid}")
public ResponseEntity<> wmscEliminarPorID(@PathVariable(value = "wmscid") int wmscid) {
    if(!Objects.isNull(wmscAlmacenService.wmscBuscarAlmacen(wmscid))) {
        wmscAlmacenService.wmscEliminarPorID(wmscid);
        return ResponseEntity.ok( body: "Almacen eliminado");
    } else {
        return ResponseEntity.ok( body: "No existe almacen");
    }
}

@PutMapping(value = "/wmscActualizarAlmacen/{wmscid}")
public ResponseEntity<> wmscActualizarPorId(
    @PathVariable(value = "wmscid") int wmscAlmacenID,
    @RequestParam(value = "wmscNombreAlmacen") String wmscNombreAlmacen
) {
    if(!Objects.isNull(wmscAlmacenService.wmscBuscarAlmacen(wmscAlmacenID))) {
        wmscAlmacen wmscAlmacen = new wmscAlmacen(wmscAlmacenID, wmscNombreAlmacen);
        wmscAlmacenService.wmscActualizarPorId(wmscAlmacen);
        return ResponseEntity.ok(wmscAlmacen);
    } else {
        return ResponseEntity.ok( body: "No existe almacen");
    }
}
```

Figura 22 Controlador Almacén métodos almacenes, buscar por id y agregar

```
@RestController
public class WmscAlmacenControlador {

    @Autowired
    private WmscAlmacenServicio wmscAlmacenService;

    // wilmerxx
    @GetMapping("/wmscAlmanes")
    public ResponseEntity<?> wmscAlmacenListas(){
        List<WmscAlmacen> wmscAlmacenList = wmscAlmacenService.wmscAlmacenListas();
        if(!wmscAlmacenList.isEmpty()){
            return ResponseEntity.ok(wmscAlmacenList);
        }else {
            return ResponseEntity.ok(Collections.emptyList());
        }
    }

    // wilmerxx
    @GetMapping("/wmscAlmanePorID/{wmscid}")
    public ResponseEntity<?> wmscBuscarPorID(@PathVariable(value = "wmscid") int wmscid){
        WmscAlmacen wmscAlmace = wmscAlmacenService.wmscBuscarAlmacen(wmscid);
        if(!Objects.isNull(wmscAlmace)){
            return ResponseEntity.ok(wmscAlmace);
        }else {
            return ResponseEntity.ok( body: "No existe el almacen");
        }
    }

    // wilmerxx *
    @PostMapping("/wmscAgregarAlmacen")
    public ResponseEntity<?> wmscAgregar(
        @RequestParam(value = "wmscAlmacenID") int wmscAlmacenID,
        @RequestParam(value = "wmscNombreAlmacen") String wmscNombreAlmacen
    ){
        if( Objects.isNull(wmscAlmacenService.wmscBuscarAlmacen(wmscAlmacenID))){
            WmscAlmacen wmscAlmace = new WmscAlmacen(wmscAlmacenID,wmscNombreAlmacen);
            wmscAlmacenService.wmscAgregar(wmscAlmace);
            return ResponseEntity.ok(wmscAlmace);
        }else{
            return ResponseEntity.ok( body: "Ya existe este cliente");
        }
    }
}
```

WmscProductoControlador: Controlador que maneja las peticiones HTTP relacionadas con los productos. En la **figura 23 y 24**, se muestra la implementación de los métodos y tipo de petición.

Figura 23 Controlador Producto métodos lista, agregar

```
@RestController
public class WmscProductoControlador {

    @Autowired
    WmscProductoServicio wmscProductoServicio;

    @GetMapping("/wmscListaProductos")
    public ResponseEntity<?> wmscListaProductos() {
        try {
            List<WmscProducto> wmscProductoList = wmscProductoServicio.wmscProductoList();

            if (!wmscProductoList.isEmpty()) {
                return ResponseEntity.ok(wmscProductoList);
            } else {
                return ResponseEntity.ok(Collections.emptyList()); // Devuelve una lista vacía si no hay datos
            }
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error al procesar la solicitud");
        }
    }

    @PostMapping("/wmscAgregarProducto")
    public ResponseEntity<?> wmscAgregar(
        @RequestParam(value = "wmscProductoID") int wmscProductoID,
        @RequestParam(value = "wmscDepartamentoID") int wmscDepartamentoID,
        @RequestParam(value = "wmscNombre") String wmscNombre,
        @RequestParam(value = "wmscPrecio") double wmscPrecio) {
        if (Objects.isNull(wmscProductoServicio.wmscBuscarPorID(wmscProductoID))) {
            WmscProducto wmscProduct = new WmscProducto(
                wmscProductoID,
                wmscDepartamentoID,
                wmscNombre,
                wmscPrecio
            );
            wmscProductoServicio.wmscAgregar(wmscProduct);
            return ResponseEntity.ok(wmscProduct);
        } else {
            return ResponseEntity.ok(body: "Ya existe ese producto");
        }
    }
}
```

Figura 24 Controlador Producto métodos eliminar, buscar por id, actualizar

```
@GetMapping("/wmscBuscarProducto/{wmscid}")
public ResponseEntity<?> wmscBuscarPorID(@PathVariable(value = "wmscid") int wmscid) {
    WmscProducto wmscProduct = wmscProductoServicio.wmscBuscarPorID(wmscid);
    if (!Objects.isNull(wmscProduct)) {
        return ResponseEntity.ok(wmscProduct);
    } else {
        return ResponseEntity.ok(body: "No existe ese producto");
    }
}

@DeleteMapping("/wmscEliminarProducto/{wmscid}")
public ResponseEntity<?> wmscEliminarPorID(@PathVariable(value = "wmscid") int wmscid) {
    WmscProducto wmscProduct = wmscProductoServicio.wmscBuscarPorID(wmscid);
    if (!Objects.isNull(wmscProduct)) {
        wmscProductoServicio.wmscEliminarPorID(wmscid);
        return ResponseEntity.ok(body: "Producto eliminado");
    } else {
        return ResponseEntity.ok(body: "No existe ese producto");
    }
}

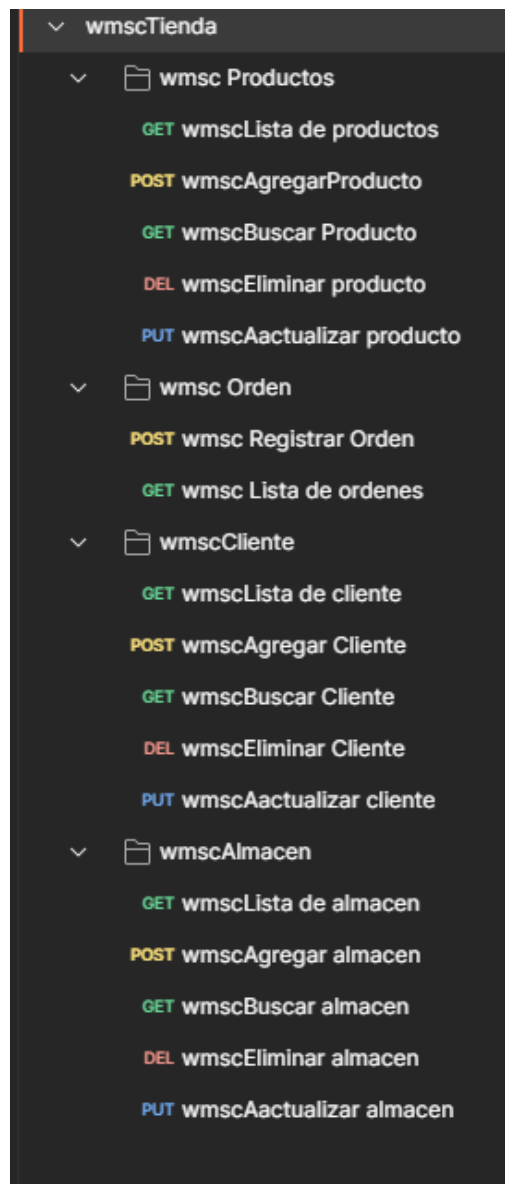
@PutMapping("/wmscActualizarProducto/{wmscid}")
public ResponseEntity<?> wmscActualizarPorID(
    @PathVariable(value = "wmscid") int wmscid,
    @RequestParam(value = "wmscDepartamentoID") int wmscDepartamentoID,
    @RequestParam(value = "wmscNombre") String wmscNombre,
    @RequestParam(value = "wmscPrecio") double wmscPrecio) {
    if (!Objects.isNull(wmscProductoServicio.wmscBuscarPorID(wmscid))) {
        WmscProducto wmscProduct = new WmscProducto(wmscDepartamentoID, wmscDepartamentoID, wmscNombre, wmscPrecio);
        wmscProductoServicio.wmscActualizarPorID(wmscProduct);
        return ResponseEntity.ok(wmscProduct);
    } else {
        return ResponseEntity.ok(body: "No existe el producto");
    }
}
```

3.2. Resultados

Para la realización de las pruebas se realiza con postman el cual permitirá realizar las peticiones de tipo get, post, put y delete.

En la **figura 25**, se puede ver las carpetas que están creadas para cada petición.

Figura 25 Lista de carpetas del postman



Almacén

En la **figura 26-31**, se muestra cada una de las funcionalidades del api.

Figura 26 Postman Agregar Almacén

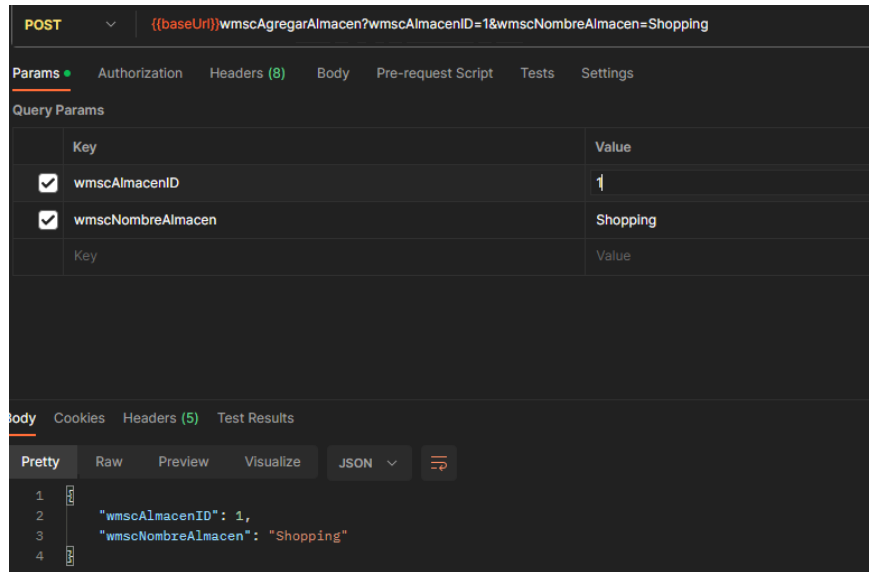


Figura 27 Postman Listar almacenes

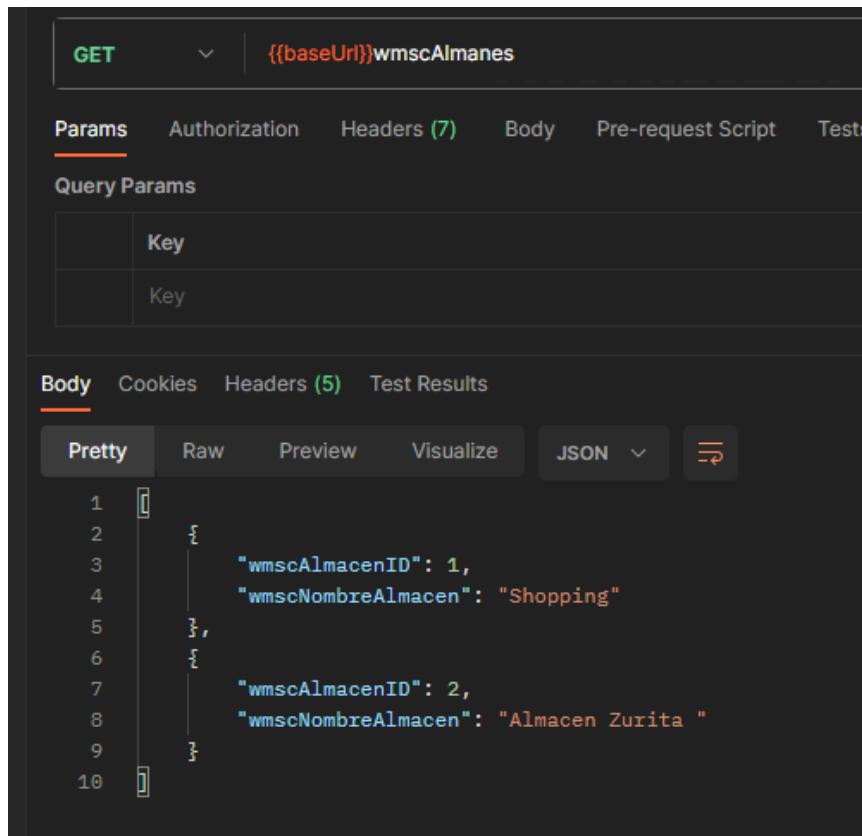


Figura 28 Postman Buscar almacén por ID

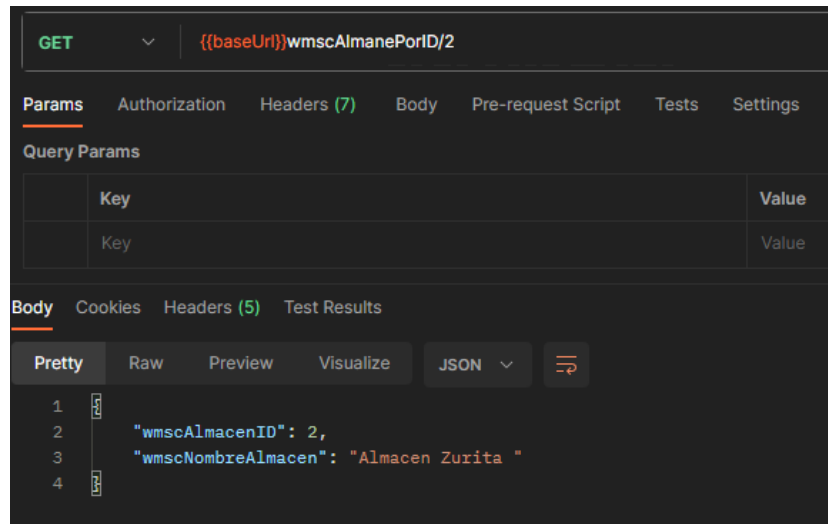


Figura 29 Postman Actualizar almacén

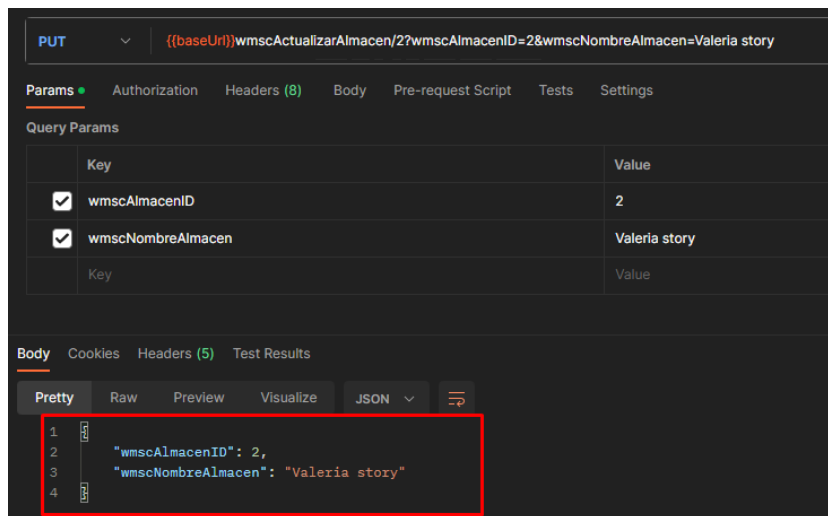


Figura 30 Postman Eliminar almacén por id

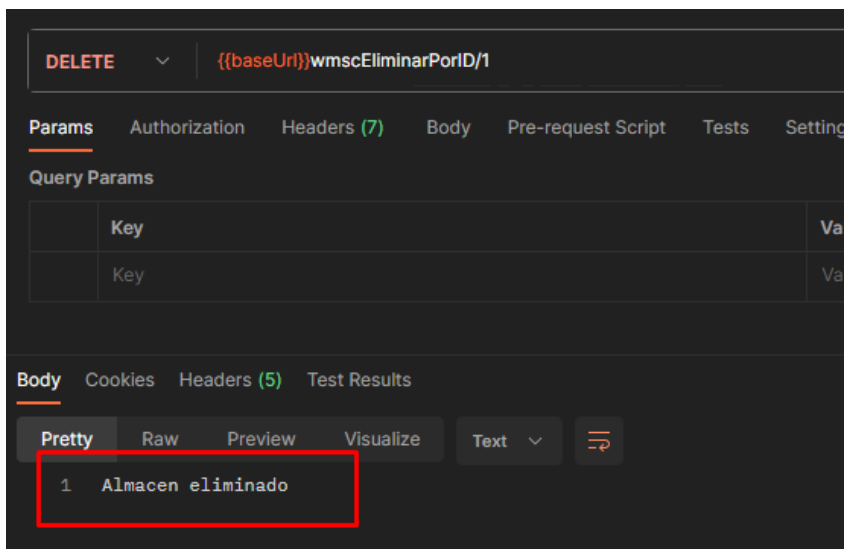
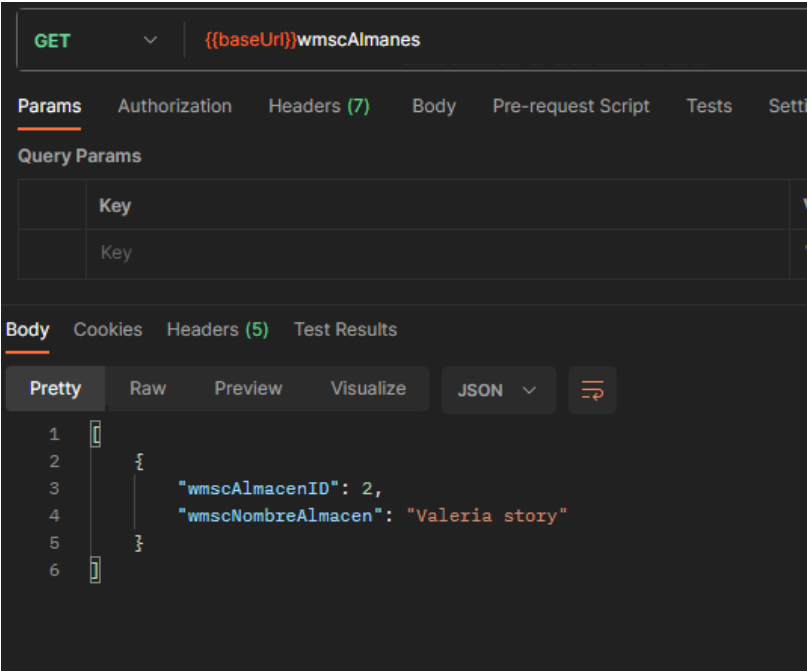


Figura 31 Postman listar almenen para verificar actualización y eliminación



Cliente

Figura 32 Postman Agregar cliente

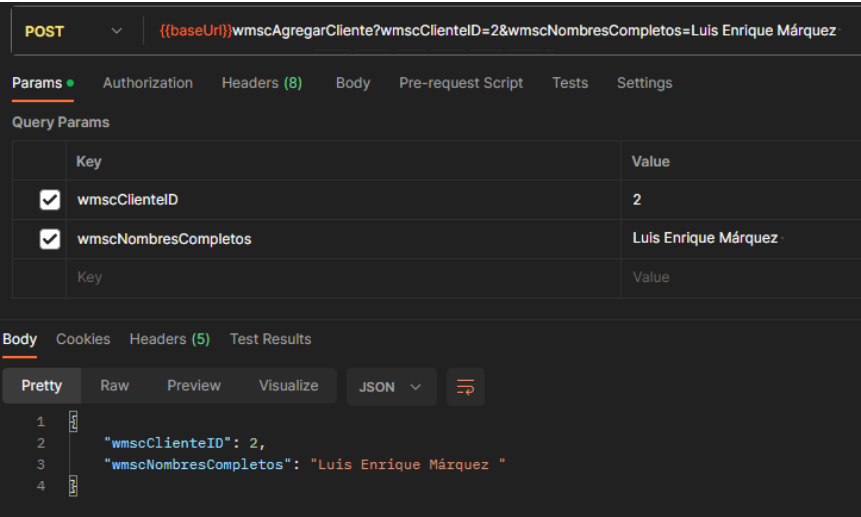


Figura 33 Postman listar clientes

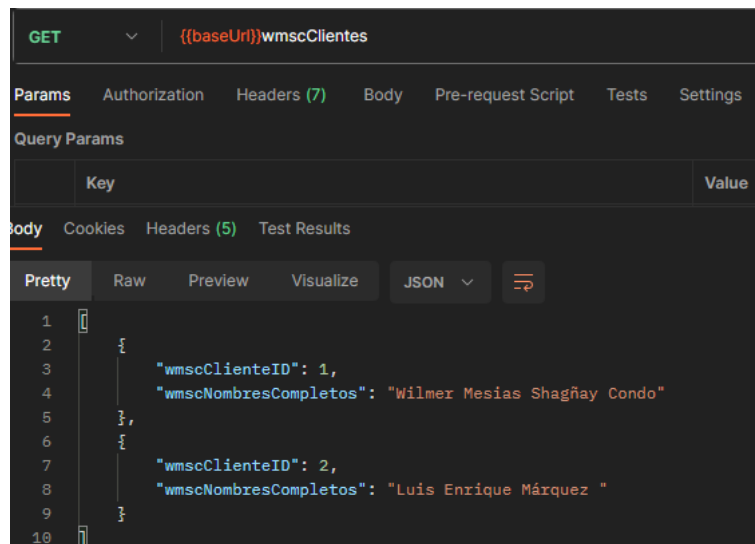


Figura 34 Postman Buscar cliente por id

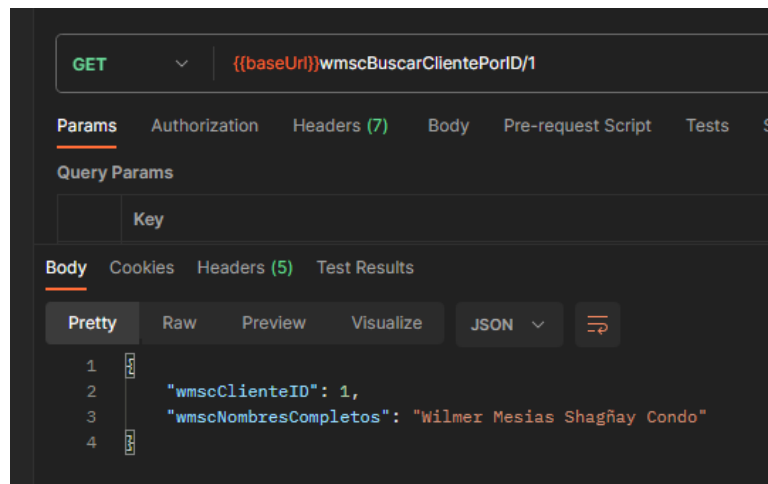


Figura 35 Postman actualizar cliente

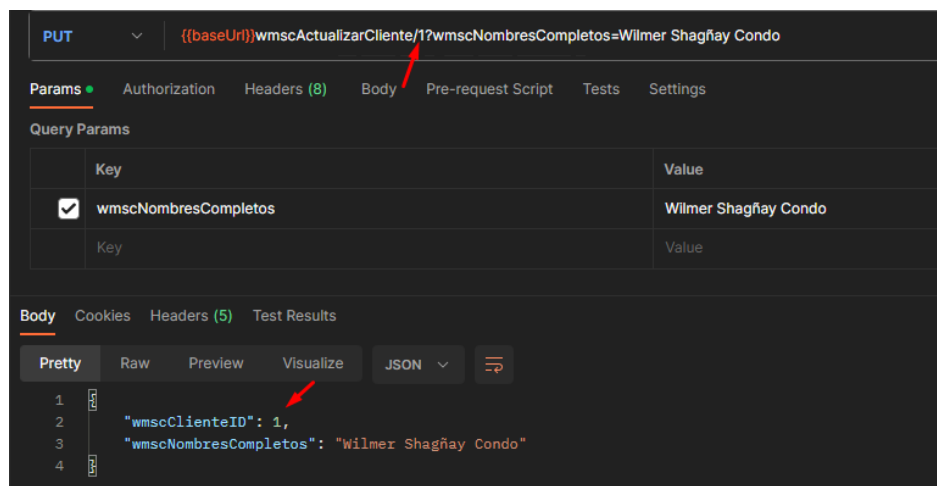


Figura 36 Postman Eliminar cliente

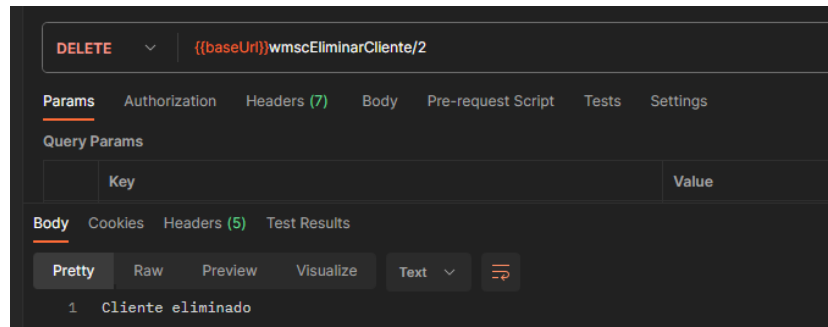
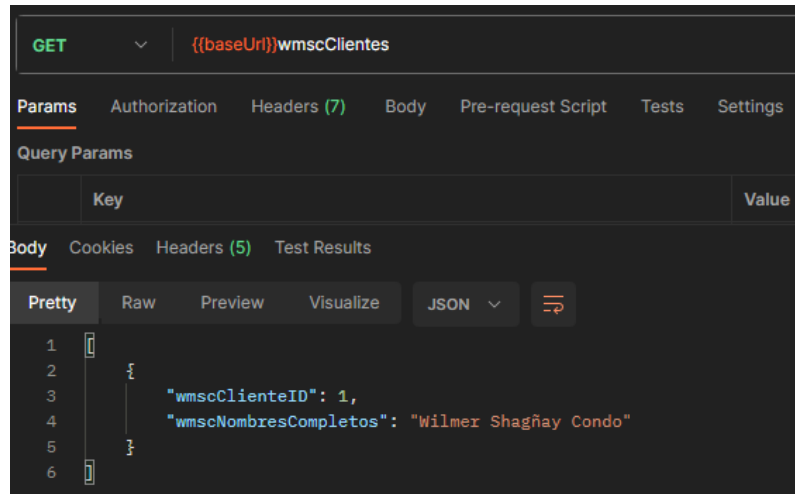
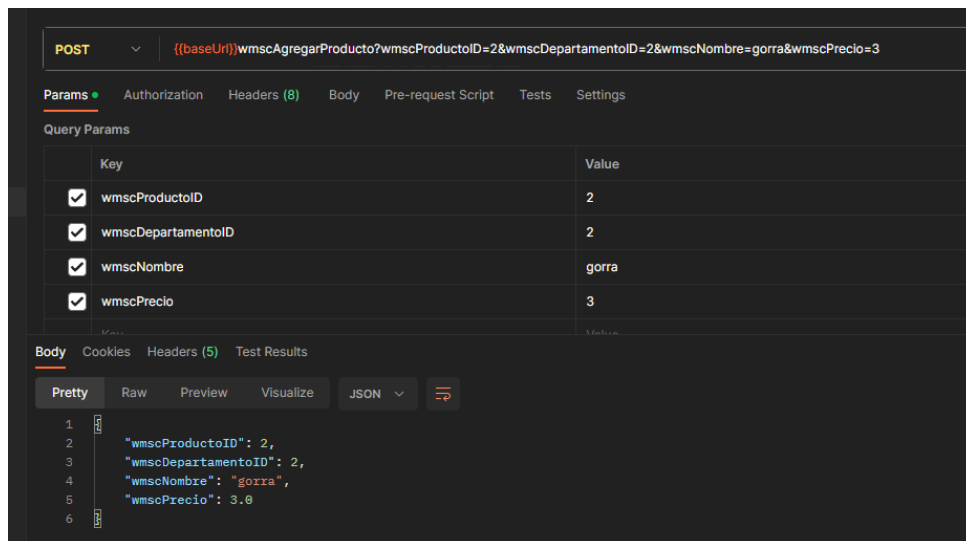


Figura 37 Postman listar clientes para verificar eliminación y actualización



Producto

Figura 38 Postman agregar producto



Se agregar tres productos para realizar la prueba de eliminar y actualizar como agregar varios productos a la orden.

Figura 39 Postman listar los productos

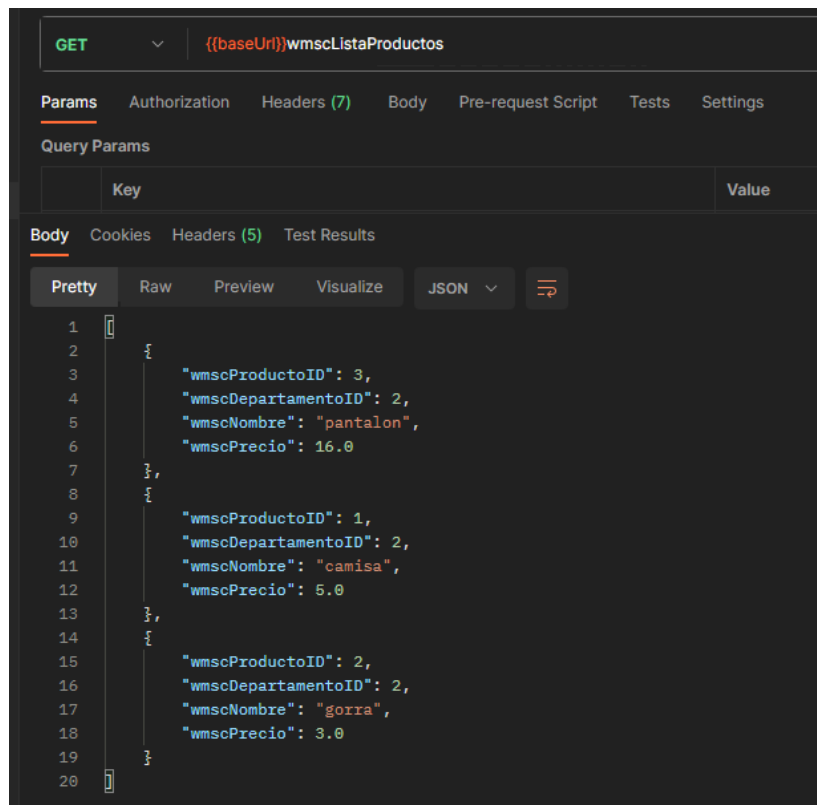


Figura 40 Postman buscar producto por id

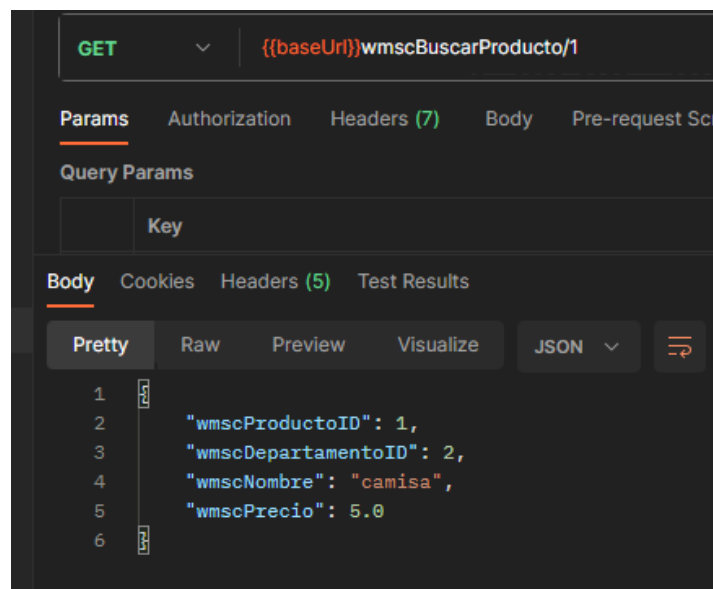


Figura 41 Postman Actualizar producto

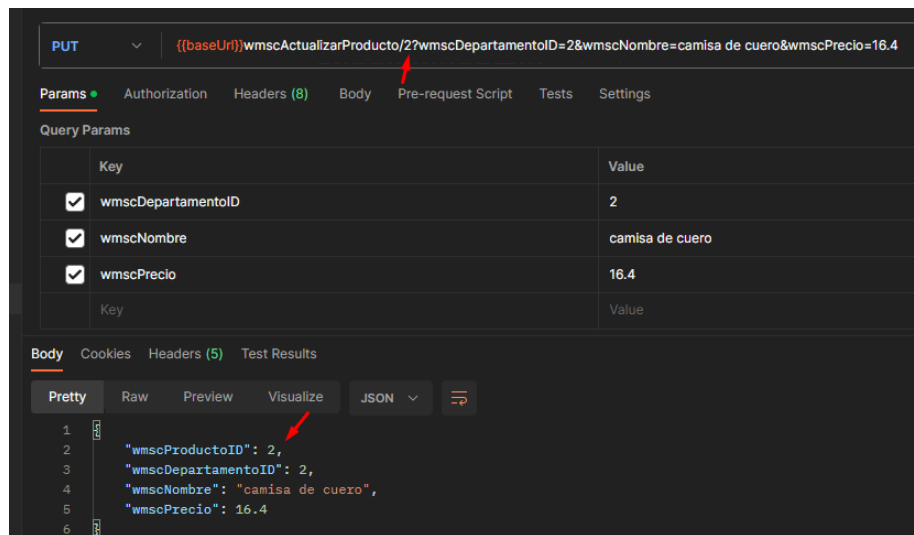


Figura 42 Postman Eliminar producto

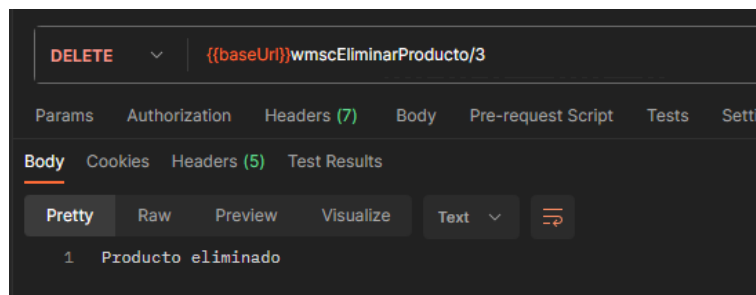
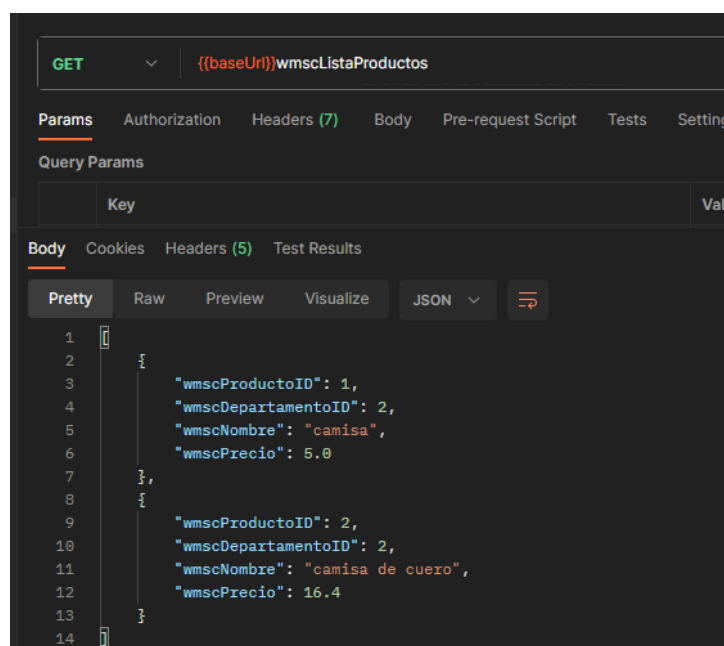


Figura 43 Postman listar productos verificación de eliminar y actualizar



Se muestran dos producto por que uno fue eliminado y se puede ver el producto actualizado.

Orden

Figura 44 Postman registrar orden

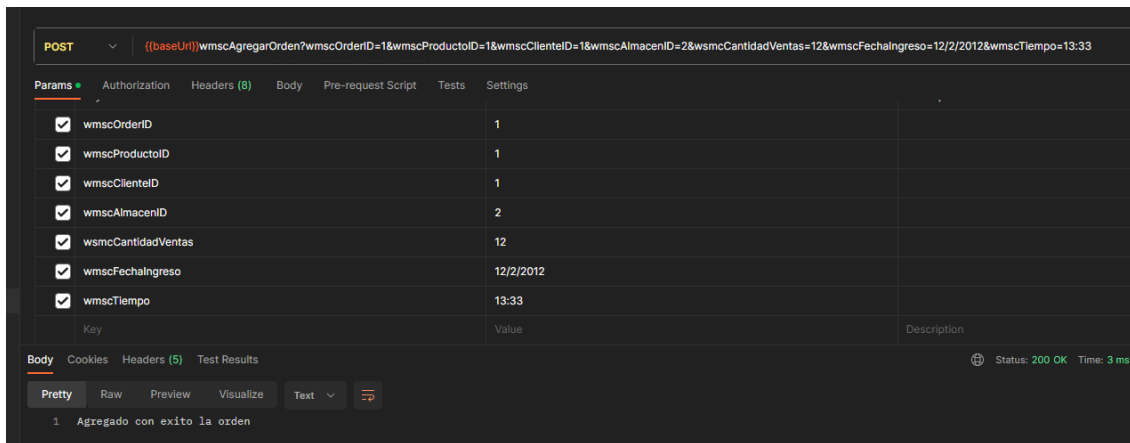


Figura 45 Postman listar ordenes

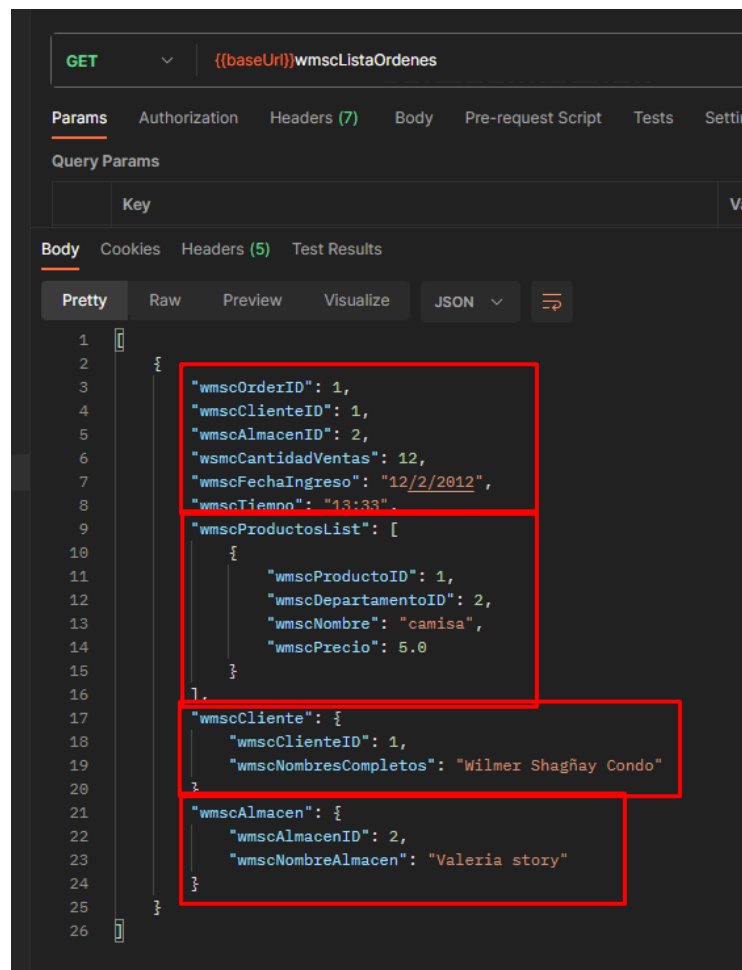


Figura 46 Postman agrega otro producto con el mismo id orden

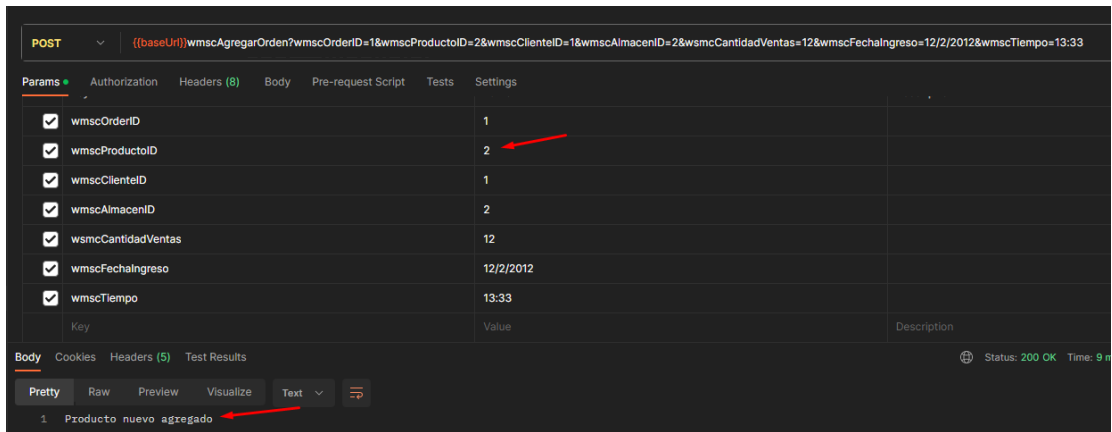
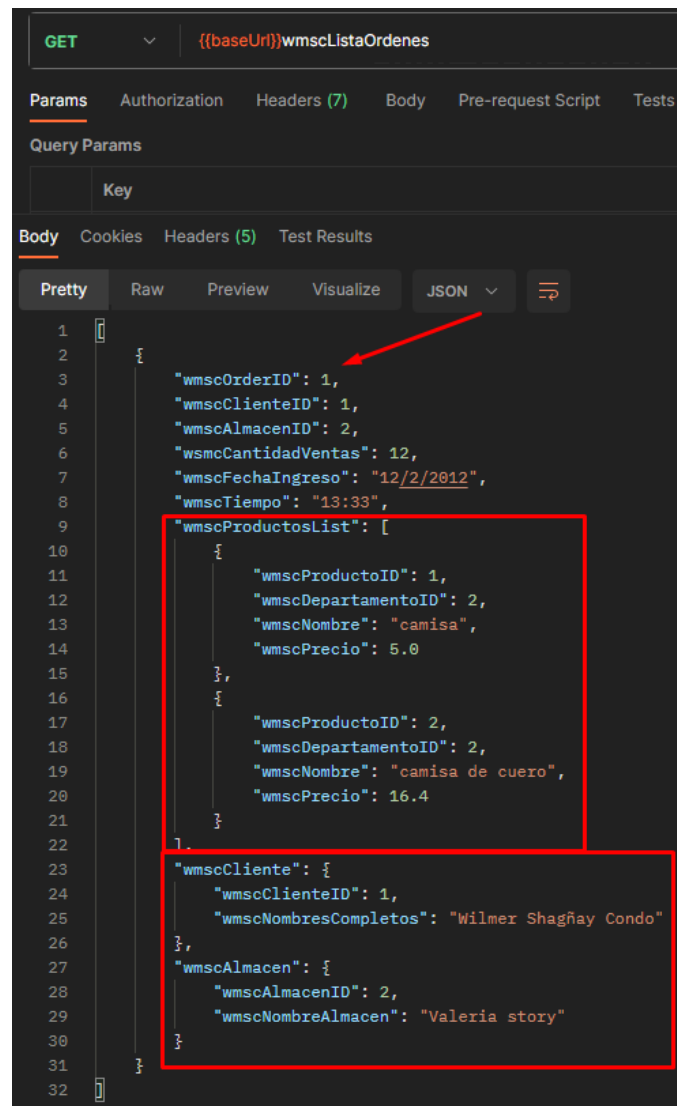


Figura 47 Postman listar ordenes



Se puede ver que en la **figura 47**, que se agrega un producto mas a la orden sin crearse otra orden porque ya existe.

Figura 48 Postman registrar otra orden

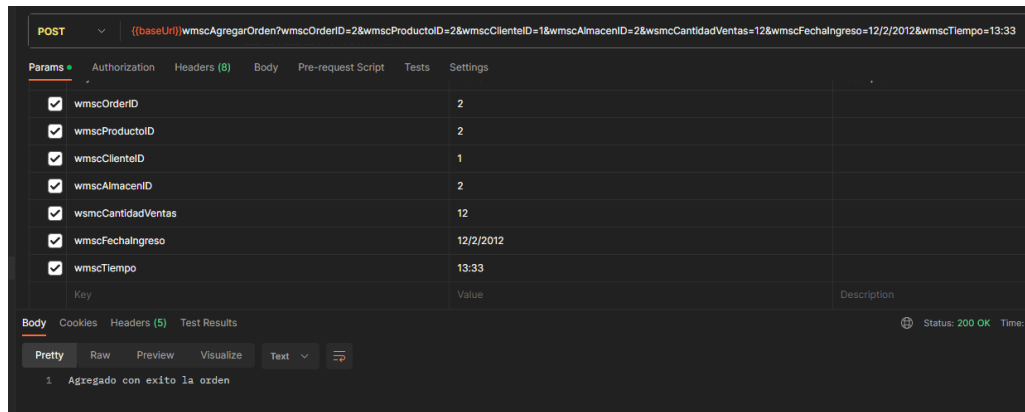
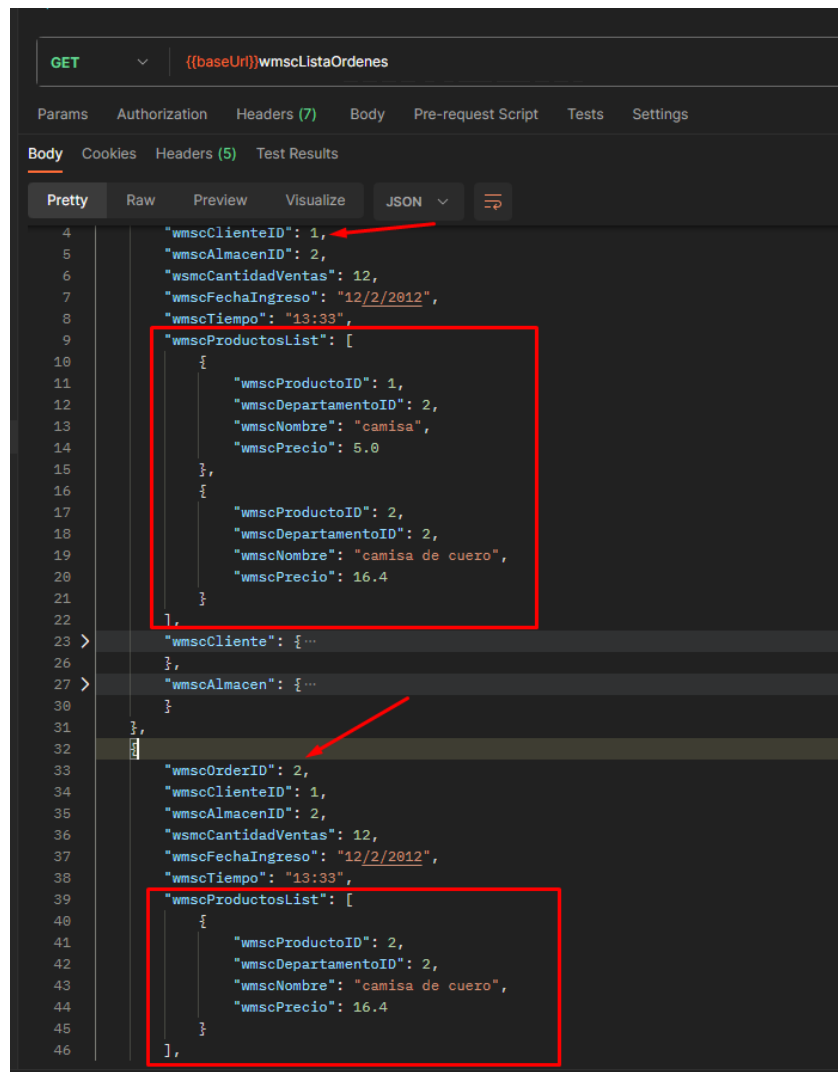


Figura 49 Postman listar ordenes verificar



Se agrega otra orden y como se puede ver en la **figura 49**, la lista de ordenes que se han creado con sus respectivos productos.

4. Conclusiones

- En la implementación de la arquitectura N-CAPAS, permite entender de como funciona y que tan útil puede ser en un sistema de negocios ya que es escalable y para un mejor manteniendo en el sistema. Y con el framework de Spring Boot se puede implementar de forma mas rápida esta solución para las empresas.
- La adopción del patrón de arquitectura Modelo-Servicio-Repositorio-Controlador (MSRC) ha proporcionado una estructura clara y coherente al proyecto ventas en online. La separación de responsabilidades entre las capas facilita el mantenimiento y la expansión del sistema.
- La gestión de la relación entre órdenes y productos se realiza de manera efectiva a través de la lógica de negocio en los servicios correspondientes. La implementación de los repositorios garantiza la persistencia de la relación de manera eficiente.
- Se realizo verificaciones en los controladores para que sigan los principios de diseño RESTful para garantizar una API coherente y fácil de consumir. Esto incluye el uso adecuado de códigos de estado HTTP, URLs significativas y la implementación de las operaciones CRUD de manera consistente al realizar pruebas con el postman y logrando así terminar con éxito la implementación y conocimiento.

5. Recomendaciones

- Mantener una documentación clara y actualizada es esencial. Esto incluye documentar las API REST, los modelos de datos, y cualquier decisión de diseño importante. Esto facilitará la colaboración y la comprensión del proyecto por parte de otros desarrolladores.
- Es importante implementar un manejo de excepciones adecuado en los servicios y controladores para garantizar una experiencia robusta para el usuario.

- Si se espera un crecimiento futuro del sistema, diseñar la arquitectura para ser escalable. Considerar la posibilidad de utilizar tecnologías de almacenamiento distribuido, así como técnicas de escalabilidad horizontal.
- Realizar validaciones en cada funcionalidad y enviar un mensaje, permitirá que se evite para el servidor por errores de datos o objetos no creados.

6. Referencias

Webb, P. (2023). *Spring Boot Reference Documentation*. Spring.io.

<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>