Integrantes:

- Paula Valentina Mora Cárdenas - 202211158
- Julián Roberto Ramírez Alemán - 202310826

RF de actualización terminados de la entrega 1

1. **Actualización Proveedor:**
   Para la actualización se solicita el nit y el nombre de contacto nuevo. Se puede ver que funciona en su prueba respectiva en postman.
2. **Actualización Producto:**
   Para la actualización se solicita el código de barras y la cantidad de la presentación de dicho producto. En la prueba de postaman se cambia de 1 a 2 la cantidad de la presentación.

Documentación RF10 – Recepción orden de compra.

1. En el controller tenemos el siguiente código, el cual es el que redirecciona la solicitud al service.

```
@PostMapping("/RecepcionProductos/new/save")
public ResponseEntity<String> RecepcionProductosGuardar(@RequestBody RecepcionProductos recepcionProductos) {
    try {
        recepcionProductosServicio.InsertarRecepcion(recepcionProductos);
    return new ResponseEntity<>("RecepcionProductos creado exitosamente", HttpStatus.CREATED);
    } catch(Exception e) {
        return new ResponseEntity<>(body:"Error al crear el RecepcionProductos", HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Como se puede ver se recibe toda la información por el body en postaman del cual se sacará toda la información necesaria para poder hacer todos los pasos que se debe realizar al recibir la orden de compra

2. El service es el que usa recursos de los repositorios para lograr su proceso.

```
@Transactional (isolation = Isolation.SERIALIZABLE, rollbackFor = Exception.class)
public void InsertarRecepcion( RecepcionProductos documento) throws Exception {
    try {
        recepcionProductosRepository.insertRecepcionProductos(documento.getId(),documento.getFechaRecepcion(), documento.getId_Bodega().getId(), documento.getId_OrdenCompra().getId());
        Collection<InfoExtraOrden> productos = infoExtraOrdenRepository.getInfoExtraOrdenpororden(documento.getId_OrdenCompra().getId());

        for (InfoExtraOrden producto : productos){
            infoExtraBodegaRepository.ActualizarCantidadyprecio(producto.getPk().getCantidad(), producto.getPk().getCostoUnitarioCompra(), producto.getPk().getCodigoBarras().getCodigoBarras());
        }
        ordenCompraRepository.Entregada(documento.getId_Bodega().getId());

    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
```

Lo que se tiene es que el nivel de aislamiento es Serializable, y hace RollBack si no se cumple algo. Ya dentro de la función hace 3 pasos para lograr el adecuado ingreso. Primero insertar la información en la tabla de Recepcion_productos. Segundo, luego de encontrar los productos relacionados con la orden de compra recorre esa lista y por

cada producto inserta la nueva información. Por último, actualiza al orden de compra como entregada.

RFC6 – Consulta de documentos de ingreso de productos a bodega

Este requerimiento de consulta está en un nivel de aislamiento SERIALIZABLE por lo que, recibiendo el id de la bodega y la sucursal deseada, hace la consulta de los documentos de aquellos que cumplan con los parámetros recibidos.

```
@Transactional(readOnly = true, isolation = Isolation.SERIALIZABLE)
public Collection<RecepcionProductos> docPorId( Long idBodega, Long idSucursal) throws InterruptedException{
    Collection<RecepcionProductos> docs= recepcionProductosRepository.documentosporId(idBodega, idSucursal);
    System.out.println(docs.size());
    Thread.sleep(millis:30000);
    docs = recepcionProductosRepository.documentosporId(idBodega, idSucursal);
    return docs;
}
```

RFC7 – Consulta de documento de ingreso de productos bodega

Este requerimiento de consulta esta dado por READ-COMMITED. Lo que hace es consultar los documentos dándonos el id de la bodega y de la sucursal. Usando un service y durmiendolo por 30 seg.

```
@Transactional(readOnly = true, isolation = Isolation.READ_COMMITTED)
public Collection<RecepcionProductos> documentosPorIdespecifico( Long idBodega, Long idSucursal) throws InterruptedException{
    Collection<RecepcionProductos> docs= recepcionProductosRepository.documentosporId(idBodega, idSucursal);
    System.out.println(docs.size());
    Thread.sleep(30000);
    docs = recepcionProductosRepository.documentosporId(idBodega,idSucursal);
    return docs;
}
```

5. Escenario de prueba de concurrencia 1

Pasos de la ejecución de la transacción:

1. Se inicia la ejecución de la consulta del RFC6 (SERIALIZABLE).
2. Después de que se empieza a ejecutar la consulta, se inicia el RF10 (registro de ingreso de productos).
3. El RFC6 sigue procesando la consulta ya que toma 30 segundos para completarse.
4. El RF10 intenta ejecutar su inserción.

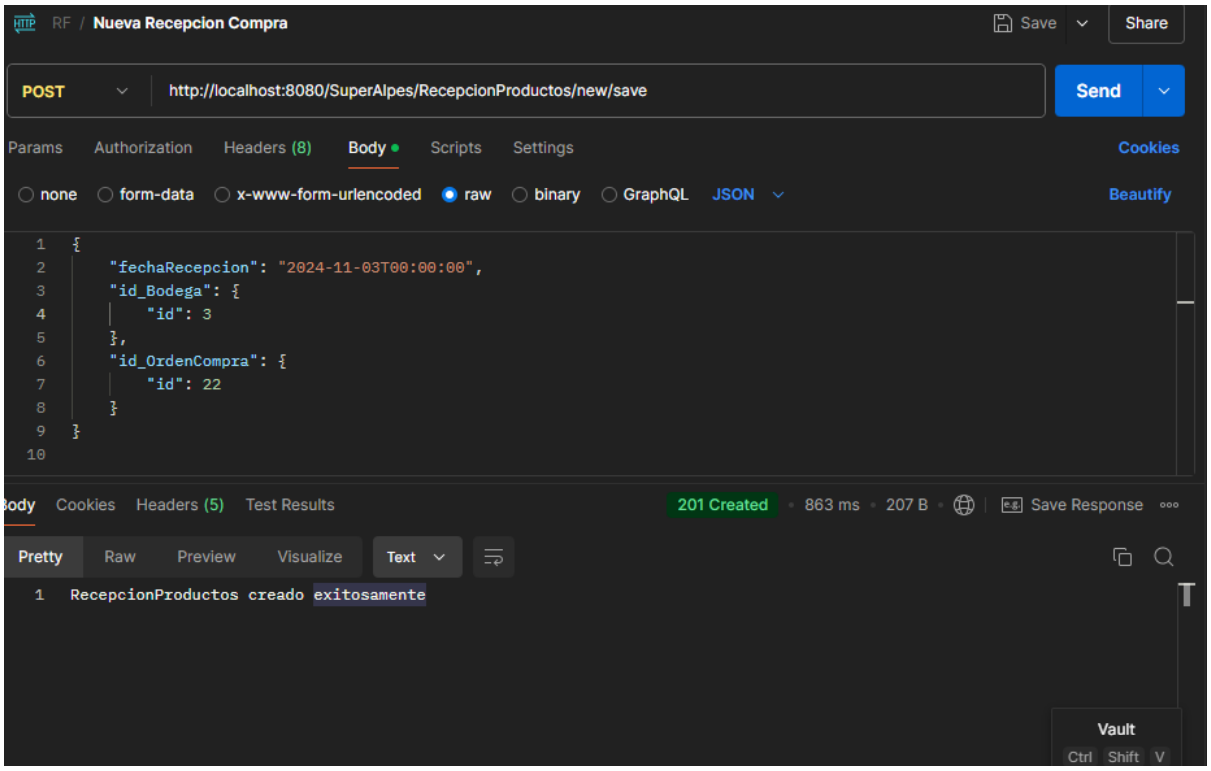
Descripción de lo sucedido:

Teniendo en cuenta que el RFC6 es SERIALIZABLE, este mantiene un bloqueo en los datos que está consultando. Por lo tanto, la inserción que intenta hacer el RF10 debe

esperar a que la consulta del RFC6 se termine antes de poder registrar un nuevo ingreso de productos. Todo lo anterior ya que el nivel de aislamiento del RFC6 impide que se realicen cambios en los datos que están siendo leídos hasta que la transacción de consulta se termine.

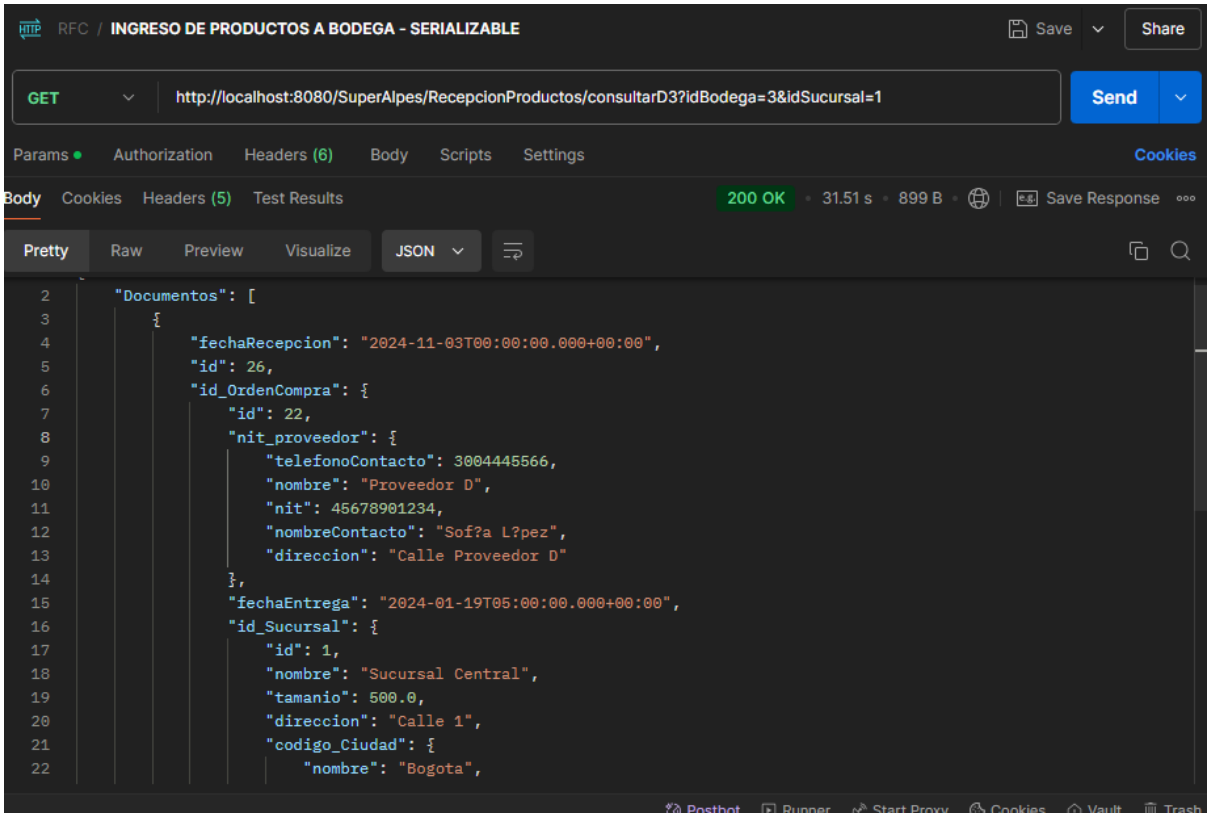
Resultado

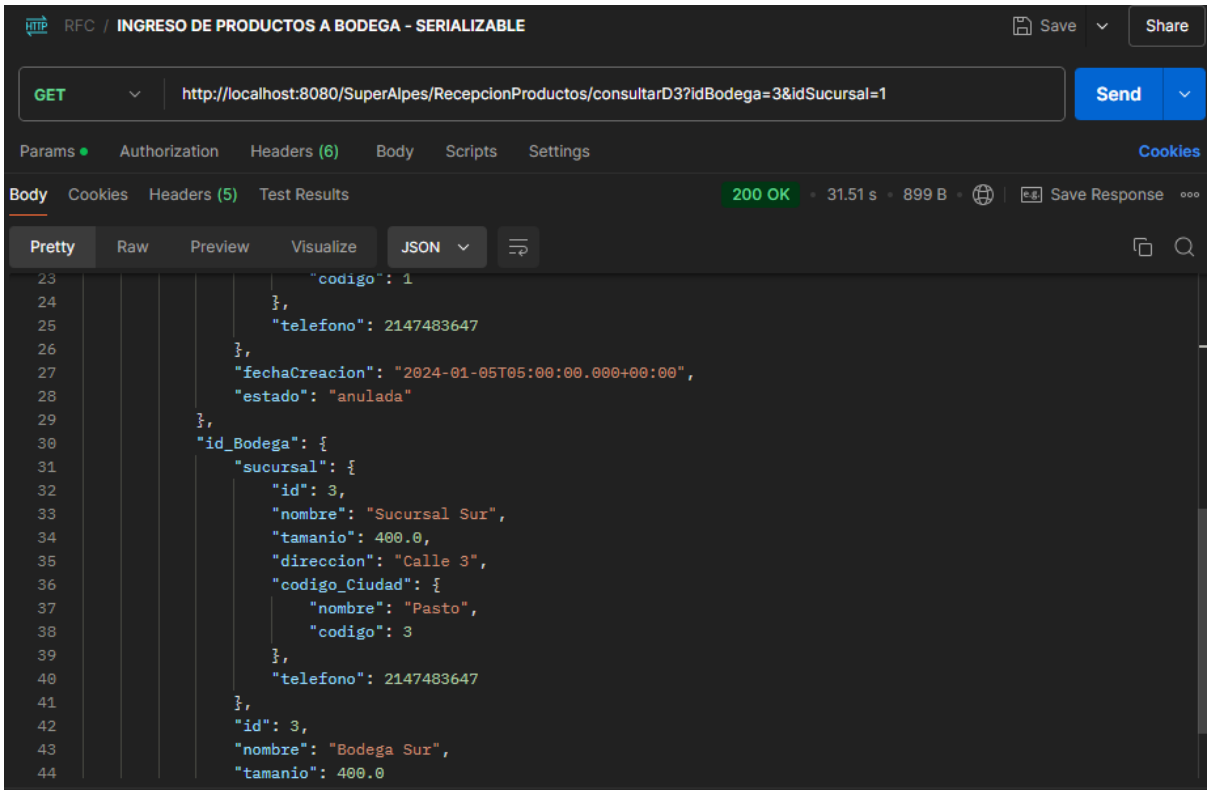
Inserción de nueva RecepcionProducto:



Outcome de la consulta RFC6 después de la inserción del RF10:

No se ven los documentos de la inserción de RecepcionProductos en la bodega con id=3.





6. Escenario de prueba de concurrencia 2

Pasos de la ejecución de la transacción:

1. Se inicia la ejecución de la consulta del RFC7 (READ-COMMITED)

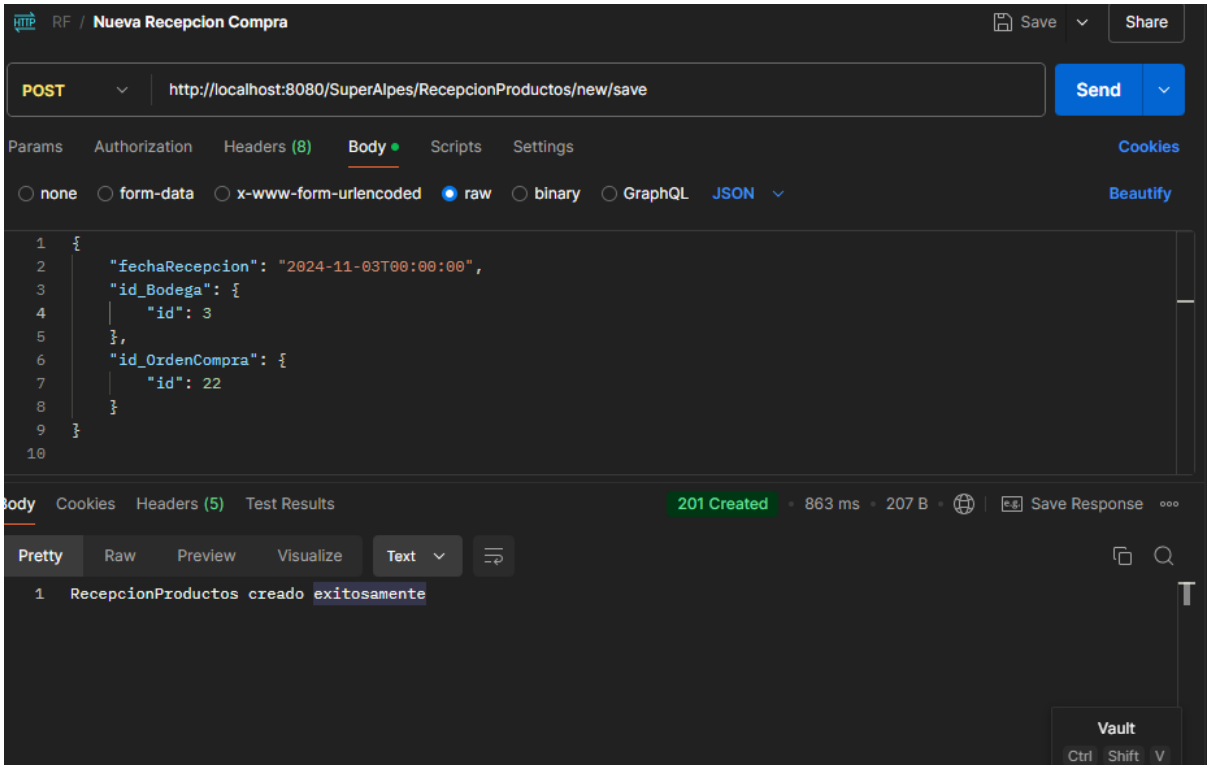
2. Una vez iniciada la consulta del RFC7, se inicia la inserción de un nuevo registro de ingreso de productos por parte del RF10
3. El RFC7 sigue procesando la consulta ya que toma 30 segundos para completarse.
4. El RF10 intenta ejecutar su inserción.

Descripción de lo sucedido:

Teniendo en cuenta que el RFC7 es READ-COMMITED, este no mantiene bloqueos prolongados como en el modo SERIALIZABLE. Por lo tanto, RF10 puede ejecutarse concurrentemente y registrar el ingreso de productos sin necesidad de esperar a que RFC7 termine. RF10 puede realizar su operación y hacer commit, lo que significa que los cambios se reflejarán en la base de datos.

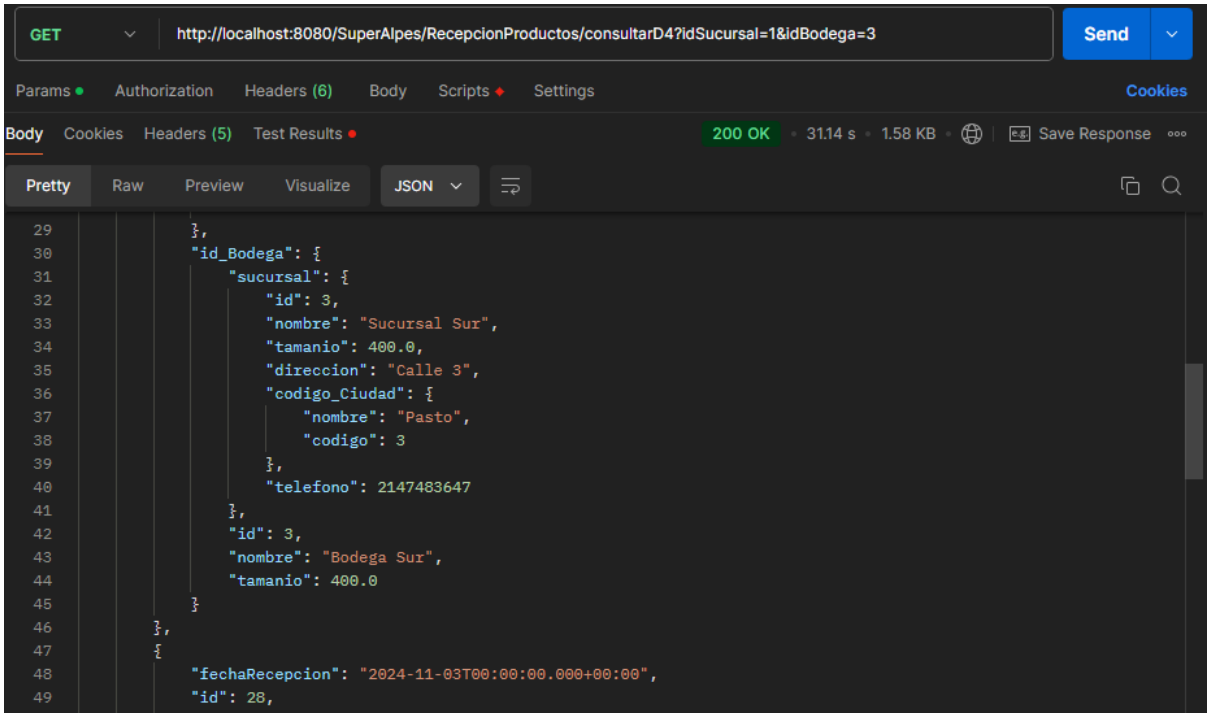
Resultado:

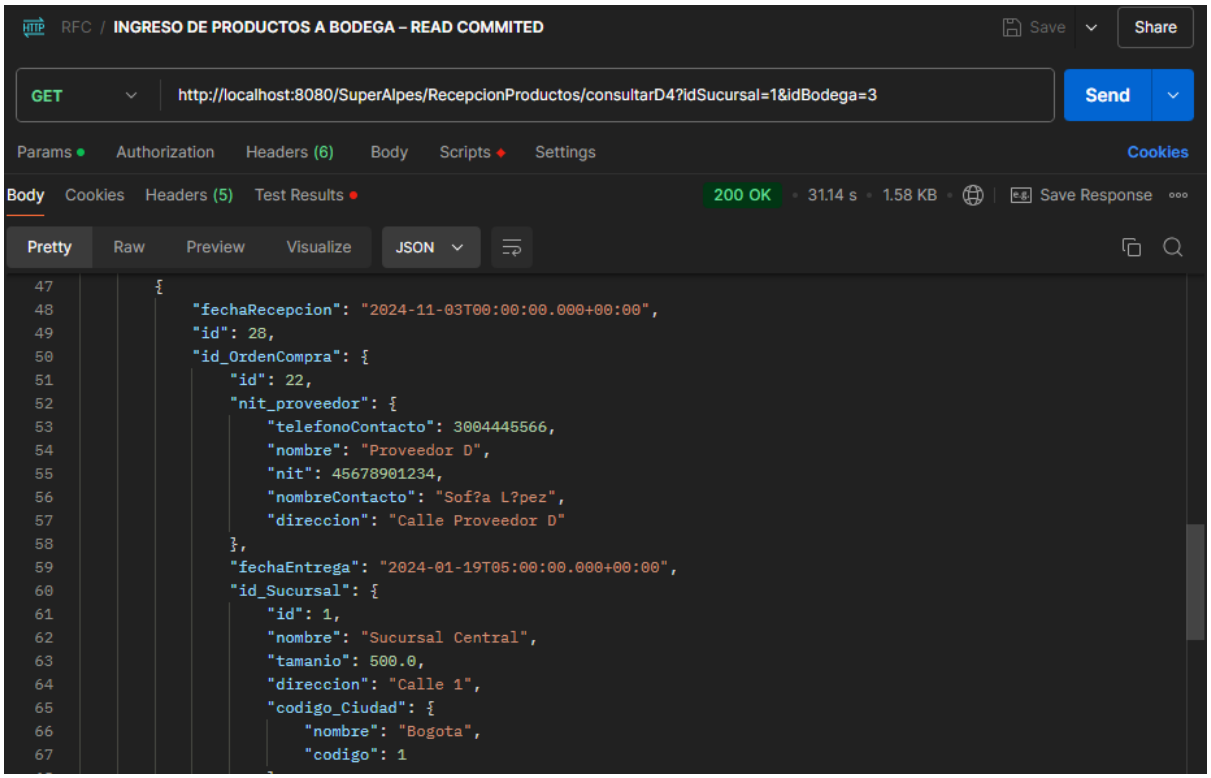
Inserción de nueva RecepcionProducto:



Outcome de la consulta RFC7 después de la inserción del RF10:

Se puede ver el registro de la RecepcionProducto en la bodega con id=3 y con la orden compra con id=22.


GET
http://localhost:8080/SuperAlpes/RecepcionProductos/consultarD4?idSucursal=1&idBodega=3
Send
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Body
Cookies
Headers (5)
Test Results
200 OK
31.14 s
1.58 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
},
"id_Bodega": {
"sucursal": {
"id": 3,
"nombre": "Sucursal Sur",
"tamanio": 400.0,
"direccion": "Calle 3",
"codigo_Ciudad": {
"nombre": "Pasto",
"codigo": 3
},
"telefono": 2147483647
},
"id": 3,
"nombre": "Bodega Sur",
"tamanio": 400.0
}
},
{
"fechaRecepcion": "2024-11-03T00:00:00.000+00:00",
"id": 28,


RFC / INGRESO DE PRODUCTOS A BODEGA – READ COMMITED
Save
Share
GET
http://localhost:8080/SuperAlpes/RecepcionProductos/consultarD4?idSucursal=1&idBodega=3
Send
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Body
Cookies
Headers (5)
Test Results
200 OK
31.14 s
1.58 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"fechaRecepcion": "2024-11-03T00:00:00.000+00:00",
"id": 28,
"id_OrdenCompra": {
"id": 22,
"nit_proveedor": {
"telefonoContacto": 3004445566,
"nombre": "Proveedor D",
"nit": 45678901234,
"nombreContacto": "Sof?a L?pez",
"direccion": "Calle Proveedor D"
},
"fechaEntrega": "2024-01-19T05:00:00.000+00:00",
"id_Sucursal": {
"id": 1,
"nombre": "Sucursal Central",
"tamanio": 500.0,
"direccion": "Calle 1",
"codigo_Ciudad": {
"nombre": "Bogota",
"codigo": 1