Integrantes: Luis Sebastián Contreras

Jessica Sofía Garay Acosta

Juan David Pérez Díaz

Informe proyecto – Entrega 2

1. A continuación, se describen las correcciones que se realizaron a los requerimientos que fallaron en la entrega anterior.

RF 2 CREAR UNA SUCURSAL

Para asegurar el funcionamiento adecuado de los métodos de creación y actualización de sucursales, se realizaron ajustes específicos en la clase Ciudad. Primero, se eliminó la anotación @GeneratedValue del ID de ciudad, permitiendo que este valor sea controlado directamente en el proceso de creación. Además, se agregó una validación en el código para verificar que el código de ciudad proporcionado sea efectivamente un entero antes de realizar la operación, evitando errores por tipos de datos incorrectos. Estos cambios se integraron en los métodos insertarSucursal y actualizarSucursal del repositorio, donde ahora se incluye el código de ciudad como un parámetro int, asegurando que las operaciones dependientes de este identificador se ejecuten sin inconveniente.

```
int codigoCiudad = sucursal.getCiudad().getCodigoCiudad().intValue();

sucursalRepository.insertarSucursal(
    sucursal.getIdSucursal(),
    sucursal.getNombreSucursal(),
    sucursal.getTamanioInstalacion(),
    sucursal.getDireccion(),
    sucursal.getTelefono(),
    codigoCiudad
);
```

RF 3 CREAR Y BORRAR UNA BODEGA

Se implementaron los métodos para crear y eliminar una bodega en el sistema, agregando las verificaciones y operaciones necesarias. Para la creación de una bodega, en el método crearBodega, se valida que la sucursal asociada tenga un ID

válido antes de proceder. Si esta validación es exitosa, se obtienen los datos de la bodega y se llama al repositorio mediante el método insertarBodega, el cual ejecuta la consulta SQL para insertar la nueva bodega con los valores IDBODEGA, IDSUCURSAL, NOMBREBODEGA y TAMANIOBODEGA en la tabla correspondiente.

Para la eliminación de una bodega, el método eliminarBodega en el controlador utiliza el método eliminarBodega del repositorio, el cual ejecuta una consulta SQL DELETE que elimina la bodega según su ID.

Para implementar la creación de una orden de compra en una sucursal, se desarrolló el método insertarOrden en el controlador, el cual recibe una instancia de OrdenCompra en el cuerpo de la solicitud. Este método se encarga de manejar los datos de la orden de compra, asegurando que los campos de fecha (fechaCreacion y fechaEntrega) se formateen correctamente en el formato yyyy-MM-dd HH:mm:ss utilizando SimpleDateFormat. Esto permite convertir las fechas adecuadamente para que puedan almacenarse en la base de datos.

RF7 - CREAR UNA ORDEN DE COMPRA PARA UNA SUCURSAL

La lógica del método consiste en llamar al repositorio mediante insertarOrden, que ejecuta una consulta SQL INSERT en la tabla ORDENCOMPRA. Esta consulta inserta los campos IDORDEN, FECHACREACION, ESTADO, FECHAENTREGA, IDSUCURSAL, NIT y CODIGOBARRAS, enlazando correctamente la orden con la sucursal (sucursalId), el producto (codigoBarras) y el proveedor (nit). En caso de éxito, el método devuelve una respuesta con un mensaje de confirmación; si ocurre algún error, se captura la excepción y se retorna una respuesta con el error detallado, asegurando una comunicación clara y robusta con el cliente.

RF8 - ACTUALIZAR UNA ORDEN DE COMPRA CAMBIANDO SU ESTADO A ANULADA

Para actualizar el estado de una orden de compra y cambiarlo a "ANULADA" (requisito funcional RF8), se implementó el método actualizarOrden en el controlador. Este método utiliza una solicitud PUT y recibe el identificador de la orden (id) junto con los detalles de la orden en el cuerpo de la solicitud. La función permite actualizar varios campos de la orden, incluidos fechaCreacion, estado, fechaEntrega, sucursalId, codigoBarras y nit.

Para realizar esta operación, se creó la consulta UPDATE en el repositorio ordenCompraRepository. En esta consulta, se actualizan los campos de la orden en la tabla ORDENCOMPRA con los valores proporcionados, garantizando la correcta

asociación de la orden con la sucursal, producto y proveedor. La operación de actualización asegura que el campo estado se modifique adecuadamente a "ANULADA" o a otro estado si es necesario.

RF9 - MOSTRAR TODAS LAS ÓRDENES DE COMPRA

Para cumplir con el requerimiento funcional RF9, que solicita mostrar todas las órdenes de compra, se implementó un método en el repositorio y otro en el controlador para recuperar y exponer esta información.

En el repositorio, se define la consulta @Query que ejecuta una selección completa de la tabla ORDENCOMPRA, recuperando todas las órdenes disponibles en la base de datos. Este método, obtenerTodasLasOrdenes, devuelve una colección de objetos OrdenCompra.

En el controlador, el método obtenerTodasLasOrdenes está expuesto mediante una solicitud GET. Cuando se invoca esta ruta, se llama al método del repositorio, lo cual permite obtener todas las órdenes y devolverlas en el cuerpo de la respuesta, listándolas directamente en formato JSON.

RFC1 - MOSTRAR EL ÍNDICE DE OCUPACIÓN DE CADA UNA DE BODEGAS DE UNA SUCURSAL

Para cumplir con el requerimiento funcional RFC1, que solicita mostrar el índice de ocupación de las bodegas de una sucursal, se implementó un método en el repositorio y otro en el controlador para exponer esta información.

1. **En el repositorio**, se define una consulta @Query que calcula el porcentaje de ocupación de cada bodega de una sucursal específica (idSucursal). Para cada bodega, la consulta realiza una operación de suma sobre el producto del CANTIDADPRESENTACION de cada Producto y el TOTALEXISTENCIAS en la bodega (INFOEXTRABODEGA), y luego divide este valor entre el TAMANIOBODEGA de la bodega para calcular el índice de ocupación como un porcentaje. El filtro permite especificar una lista de productos que se quiere analizar en este cálculo.

2. **En el controlador**, se creó el método obtenerIndiceOcupacionBodegas, el cual expone esta información a través de un GET en la ruta /{idSucursal}/ocupacion. Este método recibe el idSucursal como parámetro y una lista de productos en el cuerpo de la solicitud para calcular el índice solo de esos productos. En caso de que la lista de productos esté vacía o nula, se devuelve un mensaje de error indicando la necesidad de incluir productos. Si la operación es exitosa, el

método devuelve un ResponseEntity con la colección de IndiceOcupacionBodegaInfo, que representa el índice de ocupación de cada bodega de la sucursal solicitada.

RFC3 - INVENTARIO DE PRODUCTOS EN UNA BODEGA

Para el requerimiento funcional RFC3, que solicita obtener el inventario de productos en una bodega específica, se implementaron métodos en el repositorio y en el controlador para exponer esta información:

1. **En el repositorio**, se define una consulta @Query que selecciona los campos esenciales del inventario de productos dentro de la tabla INFOEXTRABODEGA para una idBodega específica. La consulta incluye:
    - CODIGOBARRAS como identificador de cada producto en la bodega.
    - TOTALEXISTENCIAS para indicar la cantidad disponible.
    - COSTOPROMEDIO, el costo promedio del producto en la bodega.
    - CAPACIDADALMACENAMIENTO, que indica la capacidad total de almacenamiento.
    - NIVELMINIMO, que representa el nivel mínimo de existencias recomendado.

El resultado de esta consulta es una colección de mapas (Collection<Map<String, Object>>), donde cada mapa representa un producto y su inventario en la bodega.

2. **En el controlador**, se crea el método obtenerInventarioBodega, que está expuesto como un GET en la ruta /{idBodega}/inventario. Este método recibe el idBodega como parámetro de la URL y utiliza el repositorio para obtener la información del inventario. Si la consulta devuelve resultados, el método devuelve un ResponseEntity con el inventario y un estado OK. Si no hay productos en el inventario para esa bodega, se devuelve un estado NOT_FOUND. En caso de algún error durante la operación, se captura la excepción y se devuelve un estado de error interno.

RFC4 - MOSTRAR LAS SUCURSALES EN LAS QUE HAY DISPONIBILIDAD DE UN PRODUCTO

Para RFC4, el objetivo es obtener las sucursales en las que hay disponibilidad de un producto específico, ya sea mediante su ID (productoId) o nombre. Para lograr esta

funcionalidad, se implementaron las siguientes modificaciones en el repositorio y el controlador:

1. **Controlador obtenerSucursalesConProductoDisponiblePorId**:
    - Se expone una ruta GET en /disponibilidad-producto/{productoId} que recibe productoId como parámetro en la URL.
    - Dentro del método, se llama al repositorio sucursalRepository para buscar las sucursales que tienen disponibilidad del producto correspondiente al productoId. La consulta se realiza pasando productoId como un parámetro de tipo String.
    - Si no hay resultados, se devuelve un estado NOT_FOUND. En caso de éxito, se retorna una respuesta OK con la lista de sucursales. Si ocurre algún error en el proceso, se captura la excepción y se devuelve un estado de error interno (INTERNAL_SERVER_ERROR).
2. **Consulta en SucursalRepository**:
    - En el repositorio, se define la consulta obtenerSucursalesConProductoDisponible. La consulta SQL utiliza una combinación de JOIN entre las tablas SUCURSAL, BODEGA, INFOEXTRABODEGA y PRODUCTO para identificar las sucursales donde hay disponibilidad del producto.
    - La consulta verifica la disponibilidad basándose en el CODIGOBARRAS del producto o en el NOMBRE del producto, si está disponible.
    - La respuesta es una colección de objetos Sucursal, que representan las sucursales con el producto en inventario.

Para RFC5, se implementó una funcionalidad que permite obtener todos los productos que requieren una orden de compra debido a que su inventario está por debajo del nivel mínimo establecido. A continuación, se detalla cómo se logró esta funcionalidad en el controlador y en el repositorio:

1. **Método obtenerProductosParaOrdenDeCompra en el Controlador**:
    - Se definió una ruta GET en /ordenes-compra que devuelve todos los productos con niveles de inventario bajos, indicando que necesitan una orden de compra.

- El método llama a la consulta obtenerProductosParaOrdenDeCompra en el repositorio productoRepository.
- Si no se encuentran productos que requieran una orden de compra, devuelve un estado NOT_FOUND. Si la consulta es exitosa y se encuentran productos, se retorna una respuesta OK con los datos. En caso de un error, se captura la excepción y se devuelve un estado de error interno (INTERNAL_SERVER_ERROR).

2. **Consulta obtenerProductosParaOrdenDeCompra en ProductoRepository**:
   - La consulta SQL selecciona productos de la tabla PRODUCTO junto con sus detalles como el CODIGOBARRAS, NOMBRE, NOMBREBODEGA, NOMBRESUCURSAL, NOMBREPROVEEDOR, y la cantidadActual.
   - La consulta emplea JOIN para relacionar las tablas INFOEXTRABODEGA, BODEGA, SUCURSAL, INFOEXTRAPROVEEDOR, y PROVEEDOR para obtener los detalles completos de cada producto.
   - La condición WHERE ie.TOTALEXISTENCIAS < ie.NIVELMINIMO filtra los productos con cantidades actuales por debajo de su nivel mínimo, indicando que requieren una orden de compra.

2. A continuación, se presenta la descripción del desarrollo de los nuevos requerimientos:

RF10 - REGISTRAR INGRESO DE PRODUCTOS A LA BODEGA: el documento REGISTRO DE INGRESO

Para implementar el registro de ingreso de productos a la bodega y gestionar la transacción requerida en RF10, se agregaron varios métodos que garantizan la ejecución completa de las tres operaciones involucradas: registro de ingreso, actualización de inventario y actualización del estado de la orden de compra.

Primero, en el controlador RecepcionProductosController, se incluyó un método que recibe los datos necesarios para procesar la recepción de productos. Este método realiza una verificación inicial de la orden de compra, asegurando que esté pendiente y que aún no haya sido entregada. Si la orden ya está entregada o no se encuentra, se devuelve un mensaje de error al cliente.

Posteriormente, el método registra el ingreso del producto en la tabla RECEPCIONPRODUCTOS. La información registrada incluye detalles clave como el ID de la orden, el ID de la bodega, y la fecha de recepción, capturando toda la información relevante de la recepción.

En cuanto a la actualización de inventario y costos, se agregó un método que incrementa la cantidad de productos y actualiza el costo promedio en la tabla INFOEXTRABODEGA. Este ajuste asegura que el inventario refleje la recepción actual y que los costos se actualicen con cada nuevo ingreso.

Finalmente, se incluyó un método para actualizar el estado de la orden de compra a "ENTREGADA" en ordenCompraRepository. Este cambio de estado se ejecuta solo después de que las operaciones previas (registro de ingreso y actualización de inventario) se completan exitosamente, asegurando que la orden refleje el nuevo estado de entrega.

El método en RecepcionProductosController gestiona todo este flujo marcado con la anotación @Transactional, lo que garantiza que, si alguna de las tres operaciones falla, se realice un rollback automático de todos los cambios realizados en la base de datos. Esto protege la consistencia de los datos y asegura que la base de datos no quede en un estado intermedio en caso de error.

RFC6 - CONSULTA DE DOCUMENTOS DE INGRESO DE PRODUCTOS A BODEGA – SERIALIZABLE

Para este requerimiento se dispone del último nivel de aislamiento denominado “serializable” a través de @Transactional(isolation = Isolation.SERIALIZABLE, readOnly = true). el cual se encuentra en la carpeta de servicios “BodegaServicio”, este nivel de aislamiento permite aplicar los bloqueos más restrictivos para garantizar que ninguna otra transacción pueda modificar los datos que se están leyendo. Luego a partir de los métodos pertenecientes al repositorio de bodega se usa el método que obtiene los documentos de ingreso de productos.

```
@Transactional(isolation = Isolation.SERIALIZABLE, readOnly = true)
public Collection<Map<String, Object>> obtenerDocumentosIngresoProductos(Long idSucursal, Long idBodegaLong
    Collection<Map<String, Object>> obj = bodegaRepository.obtenerDocumentosIngresoProductos(idSucursal, id
    System.out.println(obj.size());
    Thread.sleep(30000);
    obj = bodegaRepository.obtenerDocumentosIngresoProductos(idSucursal, idBodegaLong);
    return obj;
}
```

El método mencionado anteriormente, a partir del nombre de la bodega, de la sucursal, el id de la orden de compra y el nombre del producto. Se encarga de consultar aquellos productos que cuentan con las características necesarias para ser parte del documento de ingreso. Con el uso de diversos Inner Joins, el método relaciona la información de la bodega, la sucursal, el producto y el proveedor que brinda los productos para evidenciar aquellos productos que ingresaron. El método devuelve una colección de tipo Collection<Map<String, Object>>

Finalmente, para poder manejar las excepciones correctamente se hace uso de un método en el controlador de la bodega llamado “obtenerDocIngreso” el cual si existe un bloqueo debido a transacciones simultaneas lanza un error de INTERNAL_SERVER_ERROR.

RFC7 - CONSULTA DE DOCUMENTOS DE INGRESO DE PRODUCTOS A BODEGA – READ COMMITED

Este requerimiento hace uso de los mismos métodos del RFC6. La diferencia con el anterior mencionado, es que se dispone un nivel inferior de aislamiento denominado “READ COMMITED o lectura confirmada”. Este nivel de aislamiento aplica bloqueos de lectura a los datos que son modificados por otras transacciones. A pesar de que este nivel de aislamiento no permite lecturas sucias, puede llegar a generar lecturas inconsistentes.

Para disponer este nivel de aislamiento, se hace uso de @Transactional(isolation = Isolation.READCOMMITED readOnly = true)

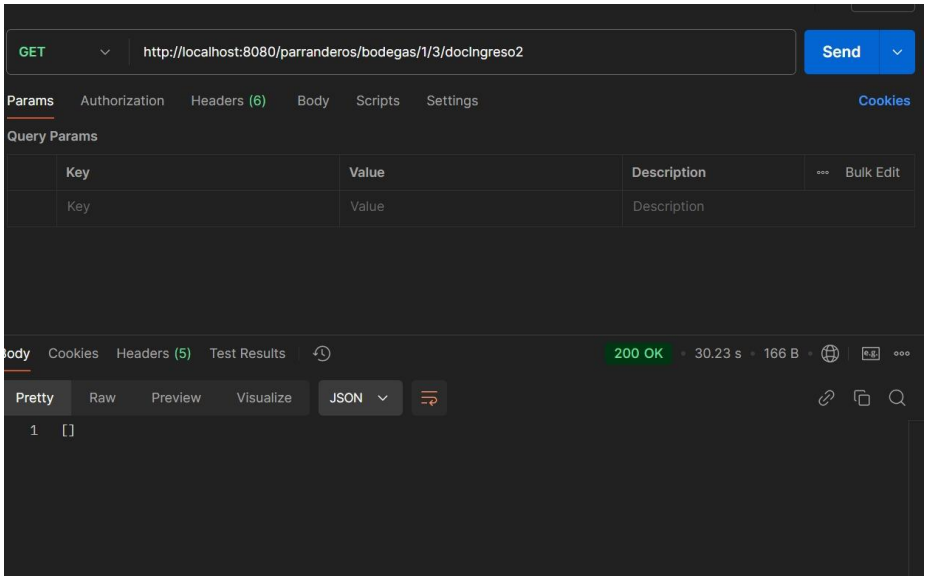
```
@Transactional(isolation = Isolation.READ_COMMITTED, readOnly = true)
public Collection<Map<String, Object>> obtenerDocumentosIngresoProductos2(Long idSucursal, Long idBodegaLong) throws InterruptedException {
    Collection<Map<String, Object>> obj = bodegaRepository.obtenerDocumentosIngresoProductos(idSucursal, idBodegaLong);
    System.out.println(obj.size());
    Thread.sleep(30000);
    obj = bodegaRepository.obtenerDocumentosIngresoProductos(idSucursal, idBodegaLong);
    return obj;
}
```

ESCENARIO DE PRUEBA DE CONCURRENCIA 1

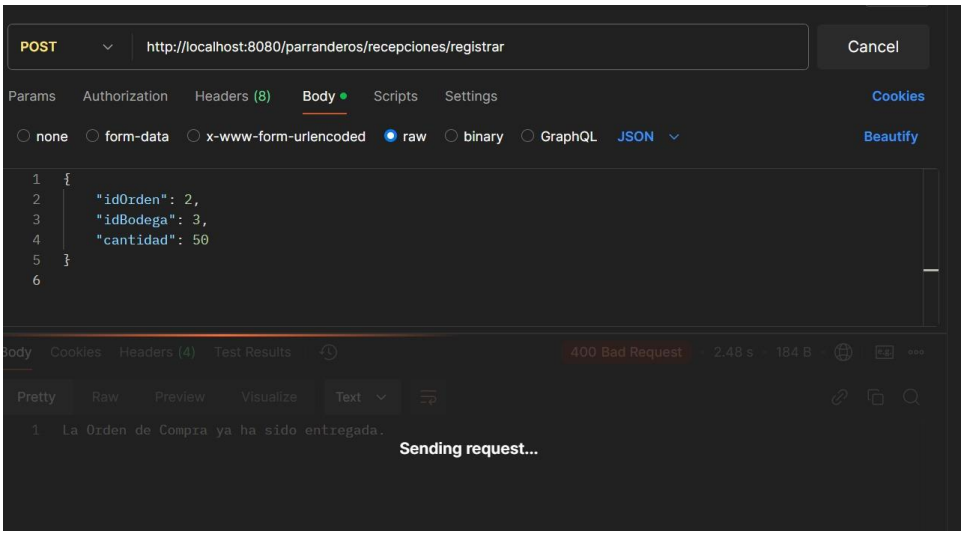
En primer lugar, se realiza la obtención del documento de ingreso de productos a través del nivel de aislamiento “Serializable”. Para esto, se hace una petición de este documento a través del nombre de la bodega, sucursal, id de la orden, la cantidad de productos, el nombre del producto y la fecha de recepción del producto. Al crear este documento y hacer la correspondiente prueba en PostMan se obtiene una lista vacía.

Luego de esto, se omiten los 30 segundos de espera para que se realice la primera transacción correctamente y se realiza el rf10.

Para el desarrollo del rf10, se registra el ingreso de un nuevo producto a través del id de la orden de compra, el id de la bodega y la cantidad de productos. Sin embargo, al realizar esta transacción de manera casi inmediata, el nivel de aislamiento se encarga de bloquear el acceso a la información de la base de datos, prohibiendo que la segunda transacción pueda modificar la información de registro que está leyendo el rfc6. Es por lo anterior, que en la prueba de PostMan se genera un error 404 y se bloquea la transacción. Para eliminar este bloqueo, es necesario que la primera transacción termine su consulta o reiniciar la aplicación.



Resultado PostMan: RFC6

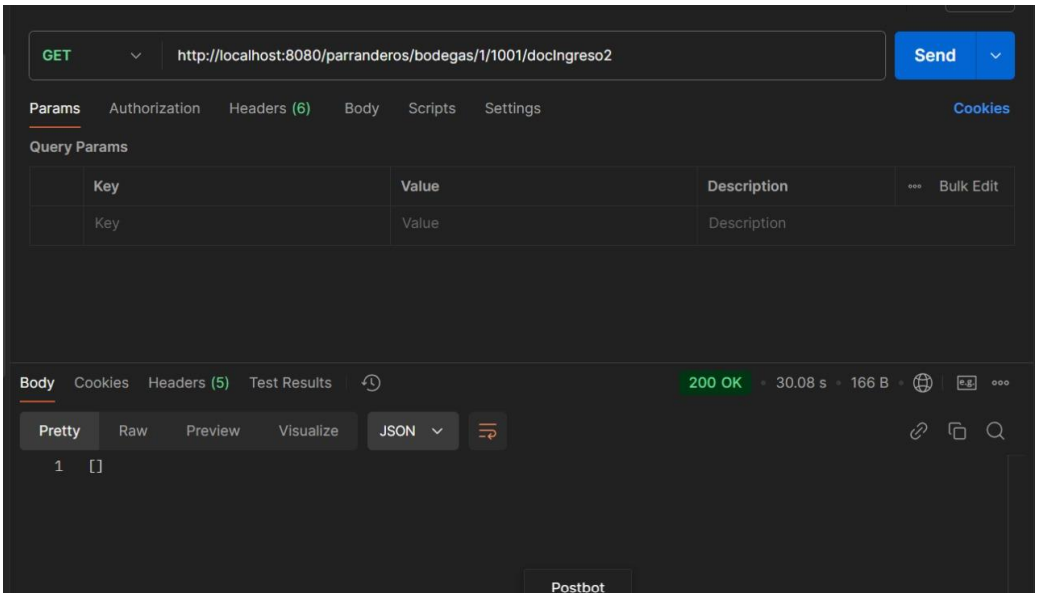


Resultado PostMan: RF10

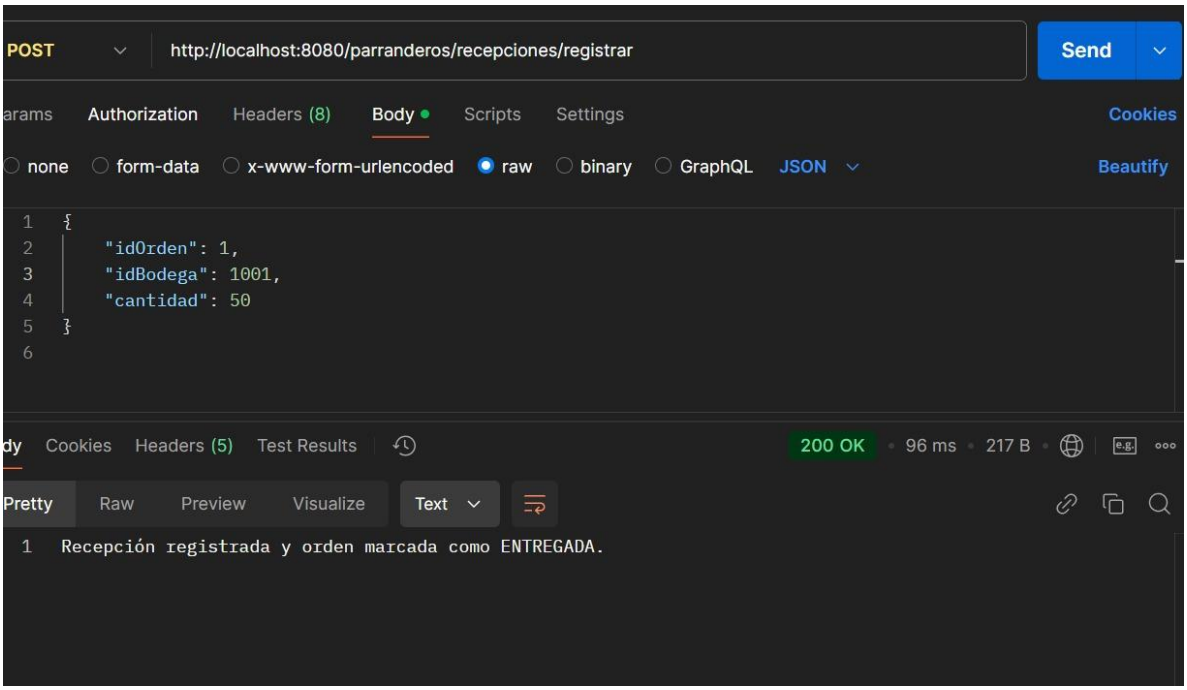
ESCENARIO DE PRUEBA DE CONCURRENCIA 2

En primer lugar, se realiza la obtención del documento de ingreso de productos a través del nivel de aislamiento “Read_commited”. Para esto, se hace una petición de este documento a través del nombre de la bodega, sucursal, id de la orden, la cantidad de productos, el nombre del producto y la fecha de recepción del producto, al igual que se realiza con el rf6. Al crear este documento y hacer la correspondiente prueba en PostMan se obtiene una lista vacía. Luego de esto, se omiten los 30 segundos de espera para que se realice la primera transacción correctamente y se realiza el rf10.

Para el desarrollo del rf10, se registra el ingreso de un nuevo producto a través del id de la orden de compra, el id de la bodega y la cantidad de productos. Al realizar esta transacción de manera casi inmediata, el nivel de aislamiento permite que la segunda transacción se realice a pesar de la simultaneidad con la primera transacción. Es por lo anterior, que en la prueba de PostMan se muestra el resultado “Recepción registrada y orden marcada como ENTREGADA” y el código 200 OK. Sin embargo, como se evidencia en los resultados los valores presentes en ambas transacciones son diferentes, esto debido a que al hacer transacciones de manera inmediata con este nivel de aislamiento se genera una lectura no repetible, lo que genera que la información sea inconsistente entre ambas transacciones.



Resultado PostMan: RFC7



Resultado PostMan: RF10