

María Paula Murillo 202224530

Mateo López 202220119

Adrián Arturo Suárez García 202123771

Entrega 2

Actualización RF y RFC entrega 1

No se hicieron cambios mayores a los requerimientos pasados, dado que funcionaban correctamente.

Solamente se alteró el RFC5, agregando el dato del nit del proveedor en la consulta.

RFC-5 Productos que requieren orden de compra

Dado que tenemos una entidad Inventario, ahí tenemos todos los productos de las bodegas, y como atributos la capacidad actual y la mínima requerida. Por lo tanto, el controlador llama la función del repositorio que busca en la base de datos todas las tuplas de Inventarios cuyo atributo capacidad sea menor al atributo del mínimo requerido. Ahí mismo, dado que se hace INNER JOIN con Productos y Bodegas, dentro del “select” de la consulta SQL se piden los demás atributos solicitados y se imprimen.

```

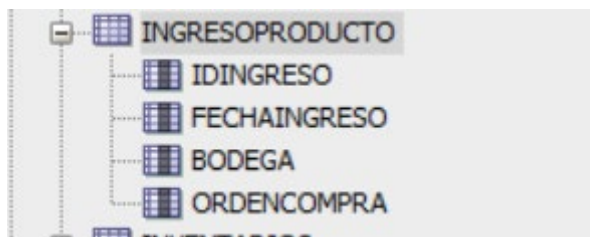
[
  {
    "productoNombre": "Crush - Cream Soda",
    "sucursal": 33,
    "productoCodigo": 48,
    "bodegaId": 34,
    "proveedor": 101253,
    "cantidadOcupada": 12
  },
  {
    "productoNombre": "Oil - Avocado",
    "sucursal": 28,
    "productoCodigo": 51,
    "bodegaId": 29,
    "proveedor": 497265,
    "cantidadOcupada": 8
  }
]

```

Documentación RF 10

Para cumplir con el Requerimiento Funcional 10, se tuvieron que hacer múltiples cambios:

- Se creó una nueva tabla en la base de datos llamada INGRESOPRODUCTO, la cual tiene los siguientes atributos y restricciones:



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	IDINGRESO	NUMBER	No	(null)	1	(null)
2	FECHAINGRESO	DATE	No	SYSDATE	2	(null)
3	BODEGA	NUMBER	No	(null)	3	(null)
4	ORDENCOMPRA	NUMBER	No	(null)	4	(null)

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	
1	FK_BODEGAING	Foreign_Key	(null)	I
2	FK_ORDENING	Foreign_Key	(null)	I
3	PK_IDINGRESO	Primary_Key	(null)	(
4	SYS_C001336895	Check	"FECHAINGRESO" IS NOT NULL	(
5	SYS_C001336896	Check	"BODEGA" IS NOT NULL	(
6	SYS_C001336897	Check	"ORDENCOMPRA" IS NOT NULL	(

El objetivo de esta tabla es poder registrar qué orden fue entregada y enviada a cuál bodega en qué fecha, cumpliendo las restricciones pedidas por el cliente para la realización del documento de ingreso.

- Modificación del proyecto de Java Spring:

Entidad de ingreso:

```

14
15 @Entity
16 @Table(name = "ingresoproducto")
17 public class Ingreso {
18     @Id
19     @GeneratedValue(strategy = GenerationType.AUTO)
20     @Column(name = "idingreso")
21     private int idIngreso;
22     @Column(name = "fechaingreso")
23     private LocalDate fechaIngreso;
24     @ManyToOne
25     @JoinColumn(name = "bodega", referencedColumnName = "ID")
26     private Bodega idBodega;
27     @OneToOne
28     @JoinColumn(name="ordencompra", referencedColumnName = "ID")
29     private Orden ordencompra;
30 }
31

```

Repositorio de ingreso:

```

public interface IngresoRepository extends JpaRepository<Ingreso,Integer>{
    @Modifying
    @Transactional
    @Query(value = "INSERT INTO INGRESOPRODUCTO (IDINGRESO, FECHAINGRESO, BODEGA, ORDENCOMPRA) " +
        "VALUES (ingreso_sequence.nextVal, SYSDATE, :bodega, :orden)", nativeQuery = true)
    void insertarIngreso(@Param("bodega") long bodega, @Param("orden") long orden);
}

```

Servicio de ingreso (en esta clase es donde se realiza la transacción del requerimiento funcional):

```

15 public class IngresoService {
22
23
24 //RFC10
25 @Transactional(isolation = Isolation.SERIALIZABLE, rollbackFor = Exception.class)
26 public void ingresarIngreso(long idBodega, long idOrden ) throws Exception {
27     try{
28         //Operación 1: Registrar el ingreso en la tabla INGRESOPRODUCTO
29         ingresoRepository.insertarIngreso(idBodega, idOrden);
30         List<Object[]> productosAfectados = ordenRepository.obtenerInfoRFC10Ingresos(idOrden);
31         if (productosAfectados != null && !productosAfectados.isEmpty()) {
32
33             for (Object[] row : productosAfectados) {
34                 long codbarras = ((BigDecimal) row[0]).longValue();
35                 long cantidad = ((BigDecimal) row[1]).longValue();
36                 long costo = ((BigDecimal) row[2]).longValue();
37                 //Operación 2: Actualizar la información de los inventarios con los productos ingresados
38                 inventarioRepository.ActualizarInventarios(codbarras, idBodega, costo, cantidad);
39             }
40         }
41     }
42     else{
43         System.out.println(x:"hay algo re mal");
44     }
45     //Operación 3: Actualizar el estado de la Orden por "ENTREGADO"
46     ordenRepository.entregarOrden(idOrden);
47 }
48 catch (Exception e) {
49     throw e; // Manejo de excepciones.
50 }
51 }
52 }
53
54 }

```

Controlador del ingreso:

```

@RestController
@RequestMapping("/ingresos")
@CrossOrigin(origins = "http://localhost:3000")
public class IngresoController {

    @Autowired
    private IngresoService ingresoService;

    @PostMapping("/nuevo")
    public ResponseEntity<String> guardarProducto
        (@RequestParam(required = false) Long idBodega,
        @RequestParam(required = false) Long idOrden) {

        try {
            ingresoService.ingresarIngreso(idBodega, idOrden);
        } catch (Exception e) {
            System.out.println(e.getMessage());
            return new ResponseEntity<>(body:"Error al realizar la transacción. Haciendo rollback...", HttpStatus.
        )
        }
        return new ResponseEntity<>(body:"Transacción completada exitosamente", HttpStatus.CREATED);
    }
}

```

- Normas ACID del requerimiento: Para cumplir con las normas ACID del manejo de transacciones, se realizaron los siguientes ajustes:
 - Atomicidad: La transacción en el Servicio de Ingresos está configurada para que, en caso de que cualquier operación de la transacción falle, se haga rollback. Esto se hace con la anotación

```
@Transactional(isolation = Isolation.SERIALIZABLE, rollbackFor = Exception.class)
```

- Consistencia: Esta propiedad es manejada por el funcionamiento interno de Oracle SQLDeveloper, no es necesario realizar ningún procedimiento para asegurar la consistencia de la base de datos.

- Aislamiento: Para asegurar un nivel de aislamiento alto, se usó la misma anotación de Atomicidad, donde se especifica que el nivel de aislamiento es Serializable.
- Durabilidad: Al igual que la consistencia, esta propiedad es asegurada por el funcionamiento de la base de datos de Oracle.

- Prueba Atomicidad:

Si se realiza la solicitud correctamente, es decir, ingresando un id de Orden correcto y un id de bodega correspondiente a esa orden (la orden solo puede ser entregada a una de las bodegas que sean pertenecientes a la sucursal que realizó la orden), la transacción se completará y se presentará un mensaje de éxito:

The screenshot shows a REST client interface with the following details:

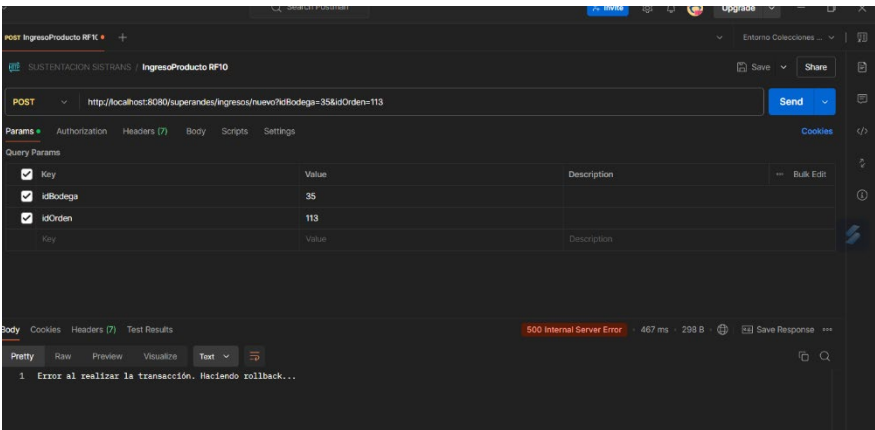
- Method:** POST
- URL:** `http://localhost:8080/superandes/ingresos/nuevo?idBodega=116&idOrden=109`
- Query Params:**

Key	Value
idBodega	116
idOrden	109
- Body:** Transacción completada exitosamente

Si se modifica algo de la transacción que incumpla alguna restricción de negocio, se hará rollback de la transacción entera, incluso si es en la última operación. Por ejemplo, se cambiará la operación de marcar la orden como entregada a que diga algún otro mensaje, lo cual incumplirá la restricción del estado de la orden, ya que solo puede estar entre 'vigente', 'entregada' y 'anulada':

```
@Modifying
@Transactional
@Query(value = "UPDATE ORDENES SET ESTADO = 'e' WHERE ID = :idOrden", nativeQuery = true)
void entregarOrden(@Param("idOrden") long idOrden);
```

Si se hace esto y se vuelve a intentar realizar un ingreso de productos, se devolverá este mensaje:



Incluso si el fallo se realizó en la última operación, se puede revisar en la base de datos que no se hizo commit de ninguna de las operaciones anteriores, tal y como se puede ver que no existe un ingreso con el IdOrden de 113:

A screenshot of a SQL query result in a database management tool. The query is `superandes.sql`. The result is a table with columns: IDINGRESO, FECHAINGRESO, BODEGA, and ORDENCOMPRA. The table contains 15 rows of data. A filter is applied: `Filter: OR...`.

IDINGRESO	FECHAINGRESO	BODEGA	ORDENCOMPRA
1	28 30-OCT-24	24	
2	10 22-OCT-24	35	
3	29 30-OCT-24	24	
4	30 30-OCT-24	24	102
5	31 30-OCT-24	24	103
6	32 30-OCT-24	24	105
7	33 30-OCT-24	24	107
8	41 02-NOV-24	24	109
9	42 02-NOV-24	116	27
10	90 03-NOV-24	116	53
11	91 03-NOV-24	24	54
12	94 03-NOV-24	24	55
13	95 03-NOV-24	24	56
14	96 03-NOV-24	116	61
15	61 02-NOV-24	24	

Documentación RFC 6

Para el requerimiento funcional de consulta 6 se realizaron diferentes cambios sobre el proyecto de Spring.

```
//RFC 6
@Query(value = "SELECT SUCURSALES.nombre as sucursal, BODEGAS.nombre as bodega, INGRESOPRODUCTO.idingreso as idingreso, INGRESOPRODUCTO.fechaingreso as fecha, PROVEEDORES.nombre as nombreProveedor \n\n" + //
"FROM BODEGAS \n\n" + //
"INNER JOIN SUCURSALES ON BODEGAS.IDSUCURSAL = SUCURSALES.IDSUCURSAL \n\n" + //
"INNER JOIN INGRESOPRODUCTO ON BODEGAS.ID = INGRESOPRODUCTO.BODEGA \n\n" + //
"INNER JOIN ORDENES ON INGRESOPRODUCTO.ORDENCOMPRA = ORDENES.ID \n\n" + //
"INNER JOIN PROVEEDORES ON ORDENES.NIT_PROVEEDOR = PROVEEDORES.NIT \n\n" + //
"WHERE INGRESOPRODUCTO.fechaingreso >= SYSDATE - 30 AND BODEGAS.ID = :id AND BODEGAS.IDSUCURSAL = :idsucursal", nativeQuery = true)
Collection<respuestaDocumento> consultarInfoOrdenes(@Param("id") Long id, @Param("idsucursal") Long idsucursal);
```

Se creó la Query necesaria para la consulta en el repositorio de Bodega. A partir de los parámetros de id y idSucursal se entrega el nombre de la Sucursal, nombre de la Bodega, id de Ingreso, fecha de ingreso y nombre del proveedor para cada uno de los ingresos que cumplan las condiciones del requerimiento. Para calcular la fecha mínima para que el ingreso sea tomado en cuenta (30 días antes de la fecha actual) se utiliza SYSDATE de SQL.

```
@Transactional(isolation = Isolation.SERIALIZABLE, rollbackFor = Exception.class)
public Collection<respuestaDocumento> getDocumentoIngresos(long id, long idsucursal) throws Exception {
    try {
        Collection<respuestaDocumento> respuesta= bodegaRepository.consultarInfoBodega(id,idsucursal);
        Thread.sleep(millis:30000);
        return respuesta;
    } catch (InterruptedException e) {
        throw new Exception(message:"Error en la espera", e);
    }
    catch (Exception ex) {
        throw new Exception(message:"Error en la consulta. Se hizo Rollback", ex);
    }
}
```

Se creó el servicio de bodega. La función getDocumentoIngresosS aplica la serialización y la espera de 30 segundos a la consulta realizada con la Query del repositorio. Se utilizó la función sleep de la clase Thread para generar la espera. Esta función realiza en un primero la espera y posteriormente la consulta, de esta forma se garantiza que la serialización sea aplicada correctamente.

```
public class BodegaController {
    @PostMapping("/bodegas/RFC6")
    public ResponseEntity<> consultaBodegaRFC6(@RequestBody List<String> params) {
        if (params.size() != 2) {
            return new ResponseEntity<>(body:"Número incorrecto de parámetros para la consulta del documento", HttpStatus.BAD_REQUEST);
        }
        try {
            Long id = Long.valueOf(params.get(index:0));
            Long idsucursal = Long.valueOf(params.get(index:1));

            Collection<respuestaDocumento> informacion = bodegaService.getDocumentoIngresos(id, idsucursal);

            // Verificar cuántos objetos están llegando en la respuesta
            int cantidadObjetos = informacion.size();
            System.out.println("Cantidad de objetos en la respuesta: " + cantidadObjetos);

            // Construir una lista de respuestas
            List<Map<String, Object>> responseList = new ArrayList<>();
            for (respuestaDocumento info : informacion) {
                Map<String, Object> response = new HashMap<>();
                response.put(key:"sucursal", info.getSucursal());
                response.put(key:"bodega", info.getBodega());
                response.put(key:"idIngreso", info.getIdIngreso());
                response.put(key:"fecha", info.getFecha());
                response.put(key:"nombreProveedor", info.getNombreProveedor());
                responseList.add(response);
            }

            return ResponseEntity.ok(responseList);
        } catch (Exception e) {
            return new ResponseEntity<>(body:"Error al consultar la información del documento", HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
}
```

Se implementó la función consultarBodegaRFC6 en el controlador de bodega. La función recibe los parámetros dados por el usuario en una lista y los extrae para implementar la función creada en el servicio. La colección que se obtiene como respuesta desde el servicio es organizada para el mejor entendimiento del usuario,

todos los ingresos son agregados a una lista response que finalmente será retornada como respuesta de la consulta.

Documentación RFC 7

Se realizaron los siguientes cambios al proyecto en Spring para la implementación del RFC7. Por otro lado, el requerimiento 7 utiliza el mismo Query que el RFC6, explicado en el punto anterior.

```
@Transactional(isolation = Isolation.READ_COMMITTED, rollbackFor = Exception.class)
public Collection<respuestaDocumento> getDocumentoIngresoRC(long id, long idsucursal) throws Exception {
    try {
        Thread.sleep(millis:30000);
        return bodegaRepository.consultarInfoBodega(id,idsucursal);
    } catch (InterruptedException e) {
        throw new Exception(message:"Error en la espera", e);
    }
    catch (Exception ex) {
        throw new Exception(message:"Error en la consulta. Se hizo Rollback", ex);
    }
}
```

Dentro del Servicio de bodega se creó la función getDocumentosIngresoRC. A diferencia de la función en servicio para el RFC6, esta función realiza la espera antes de llamar a la consulta desde el repositorio, para que así, el nivel de aislamiento pueda funcionar correctamente. La demás implementación en este método es igual que para la función getDocumentoIngresosS detallada en el punto anterior.

```
public class BodegaController {
    @PostMapping("/bodegas/RFC7")
    @CrossOrigin(origins = "http://localhost:3000")
    public ResponseEntity<> consultaBodegaRF7(@RequestBody List<String> params) {
        if (params.size() != 2) {
            return new ResponseEntity<>(body:"Número incorrecto de parámetros para la consulta del documento", HttpStatus.BAD_REQUEST);
        }
        try {
            Long id = Long.valueOf(params.get(index:0));
            Long idsucursal = Long.valueOf(params.get(index:1));

            Collection<respuestaDocumento> informacion = bodegaService.getDocumentoIngresoRC(id, idsucursal);

            // Verificar cuántos objetos están llegando en la respuesta
            int cantidadObjetos = informacion.size();
            System.out.println("Cantidad de objetos en la respuesta: " + cantidadObjetos);

            // Construir una lista de respuestas
            List<Map<String, Object>> responseList = new ArrayList<>();
            for (respuestaDocumento info : informacion) {
                Map<String, Object> response = new HashMap<>();
                response.put(key:"sucursal", info.getSucursal());
                response.put(key:"bodega", info.getBodega());
                response.put(key:"idIngreso", info.getIdIngreso());
                response.put(key:"fecha", info.getFecha());
                response.put(key:"nombreProvedor", info.getNombreProvedor());
                responseList.add(response);
            }

            return ResponseEntity.ok(responseList);
        } catch (Exception e) {
            return new ResponseEntity<>(body:"Error al consultar la información del documento", HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
}
```

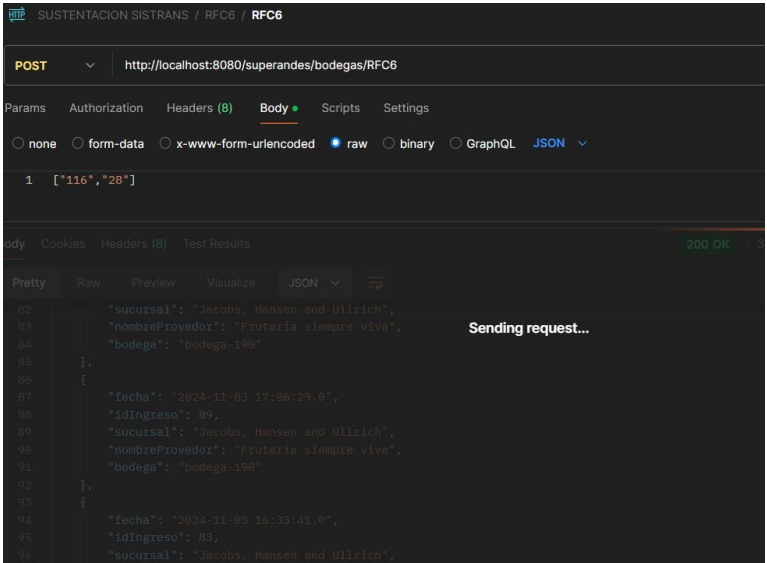
Se implementó la función consultarBodegaRF7 en el controlador de bodega. Funciona exactamente igual que la función creada para el requerimiento de consulta 6; la única diferencia se encuentra en el mapping de cada una y el método que llaman desde el servicio, siendo en este caso getDocumentosIngresoRC.

Escenario concurrencia 1

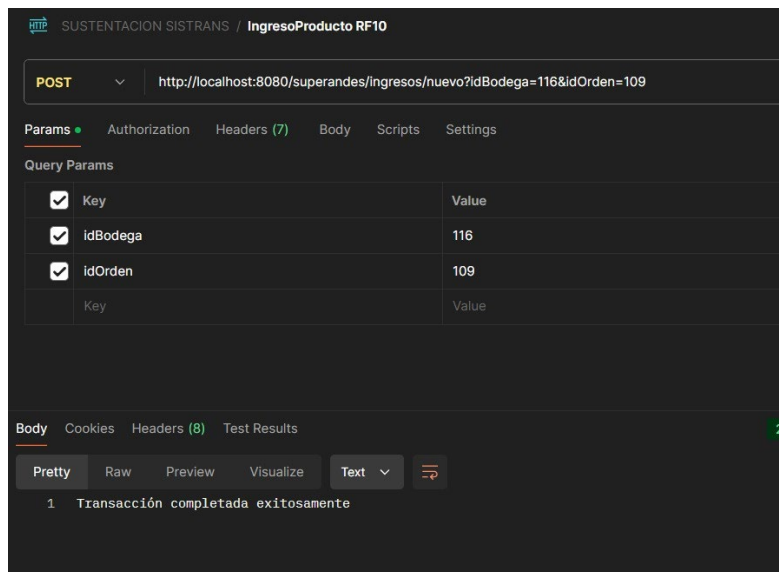
Antes de comenzar esta es la tabla de ingresos para la bodega 116:

	IDINGRESO	FECHAINGRESO	BODEGA	ORDENCOMPRA
1	41	02/11/24	116	53
2	42	02/11/24	116	27
3	81	03/11/24	116	27
4	100	02/02/24	116	26
5	101	02/05/24	116	26
6	102	02/08/24	116	26
7	43	02/11/24	116	27
8	44	02/11/24	116	27

Se ejecuta el RFC6, el cual espera 30 segundos como indica el enunciado:

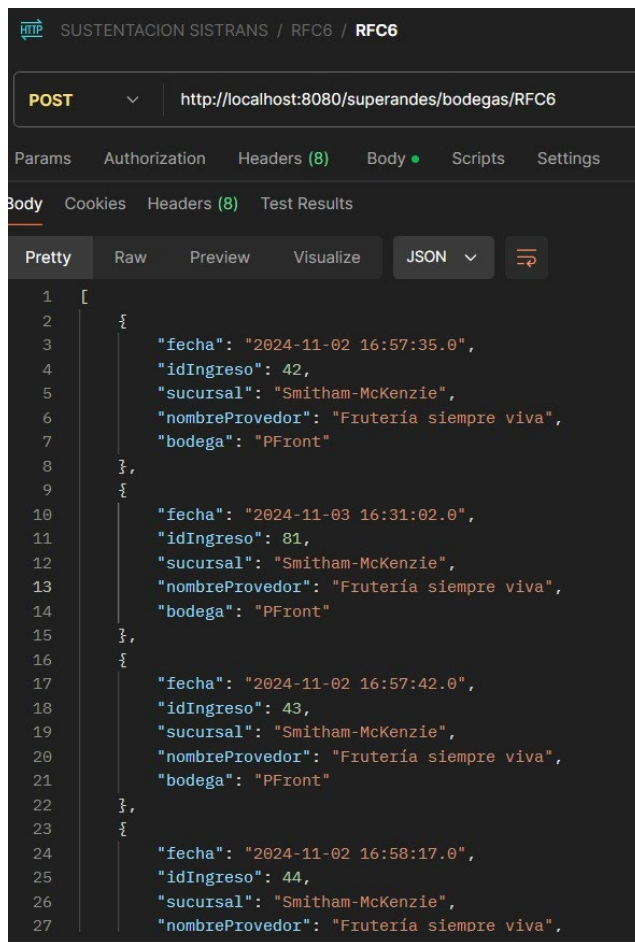


Se ejecuta el RF10, el cual crea un nuevo ingreso en la bodega 116 para la orden 109. También vemos que en la base de datos se creó el ingreso (tiene id 95).



	IDINGRESO	FECHAINGRESO	BODEGA	ORDENCOMPRA
1	41	02/11/24	116	53
2	42	02/11/24	116	27
3	95	03/11/24	116	109
4	81	03/11/24	116	27
5	100	02/02/24	116	26
6	101	02/05/24	116	26
7	102	02/08/24	116	26
8	43	02/11/24	116	27
9	44	02/11/24	116	27

No obstante, vemos en la respuesta del RFC6 que NO aparece dicho ingreso. Notar que, de todos los ingresos, solo 5 cumplen el criterio de la fecha, es decir, haber ingresado en los últimos 30 días.



```
HTTP SUSTENTACION SISTRANS / RFC6 / RFC6

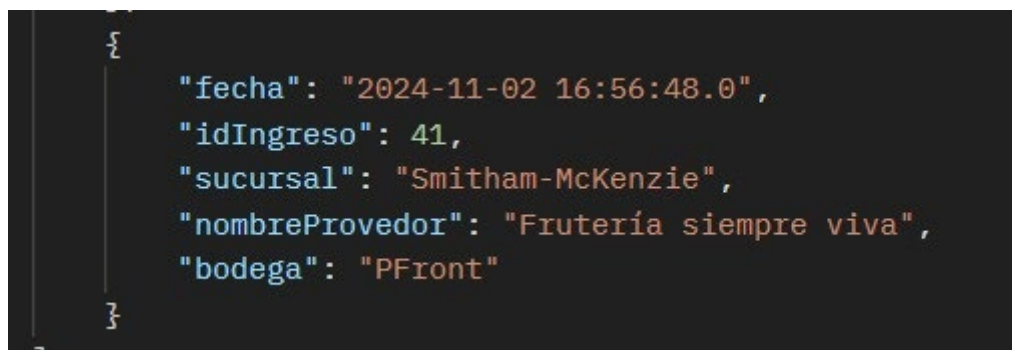
POST http://localhost:8080/superandes/bodegas/RFC6

Params Authorization Headers (8) Body Scripts Settings

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

1 [
2   {
3     "fecha": "2024-11-02 16:57:35.0",
4     "idIngreso": 42,
5     "sucursal": "Smitham-McKenzie",
6     "nombreProvedor": "Frutería siempre viva",
7     "bodega": "PFront"
8   },
9   {
10    "fecha": "2024-11-03 16:31:02.0",
11    "idIngreso": 81,
12    "sucursal": "Smitham-McKenzie",
13    "nombreProvedor": "Frutería siempre viva",
14    "bodega": "PFront"
15  },
16  {
17    "fecha": "2024-11-02 16:57:42.0",
18    "idIngreso": 43,
19    "sucursal": "Smitham-McKenzie",
20    "nombreProvedor": "Frutería siempre viva",
21    "bodega": "PFront"
22  },
23  {
24    "fecha": "2024-11-02 16:58:17.0",
25    "idIngreso": 44,
26    "sucursal": "Smitham-McKenzie",
27    "nombreProvedor": "Frutería siempre viva",
```



```
{
  "fecha": "2024-11-02 16:56:48.0",
  "idIngreso": 41,
  "sucursal": "Smitham-McKenzie",
  "nombreProvedor": "Frutería siempre viva",
  "bodega": "PFront"
}
```

Los pasos de ejecución siguen el orden de las fotos: ejecutar el RFC6, luego insertar un ingreso mediante el RF10, esperar al resultado de RFC10, y ver que no aparece la inserción, indicando que esta se realizó después de la consulta.

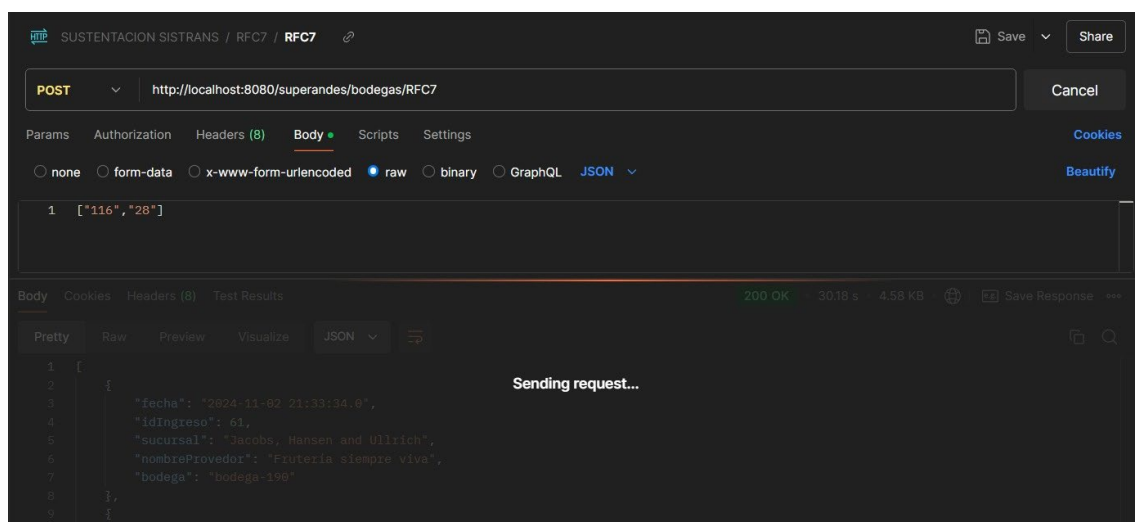
Se evidencia que el RF10 espera a que termine el RFC6 para insertar un nuevo dato. El RFC6, al tener un aislamiento de tipo “serializable”, pone un candado sobre los datos que consulta, de esta forma el RF10 no puede insertar efectivamente los datos hasta después que la lectura del RFC6 termina. Por esto, el RFC6 no muestra el dato nuevo, ya que fue agregado después de la consulta, y tendríamos que volver a leer los dato para ver la inserción.

Escenario concurrencia 2

Antes de comenzar esta es la tabla de ingresos para la bodega 116:

	IDINGRESO	FECHAINGRESO	BODEGA	ORDENCOMPRA
1	41	02/11/24	116	53
2	42	02/11/24	116	27
3	95	03/11/24	116	109
4	81	03/11/24	116	27
5	100	02/02/24	116	26
6	101	02/05/24	116	26
7	102	02/08/24	116	26
8	43	02/11/24	116	27
9	44	02/11/24	116	27

Se ejecuta el RFC7, el cual espera 30 segundos como indica el enunciado:



Se ejecuta el RF10, el cual crea un nuevo ingreso en la bodega 116 para la orden 109. También vemos que en la base de datos se creó el ingreso (tiene id 96).

HTTP SUSTENTACION SISTTRANS / IngresoProducto RF10

POST <http://localhost:8080/superandes/ingresos/nuevo?idBodega=116&idOrden=109>

Params • Authorization Headers (7) Body Scripts Settings

Query Params

<input checked="" type="checkbox"/>	Key	Value
<input checked="" type="checkbox"/>	idBodega	116
<input checked="" type="checkbox"/>	idOrden	109
<input type="checkbox"/>	Key	Value

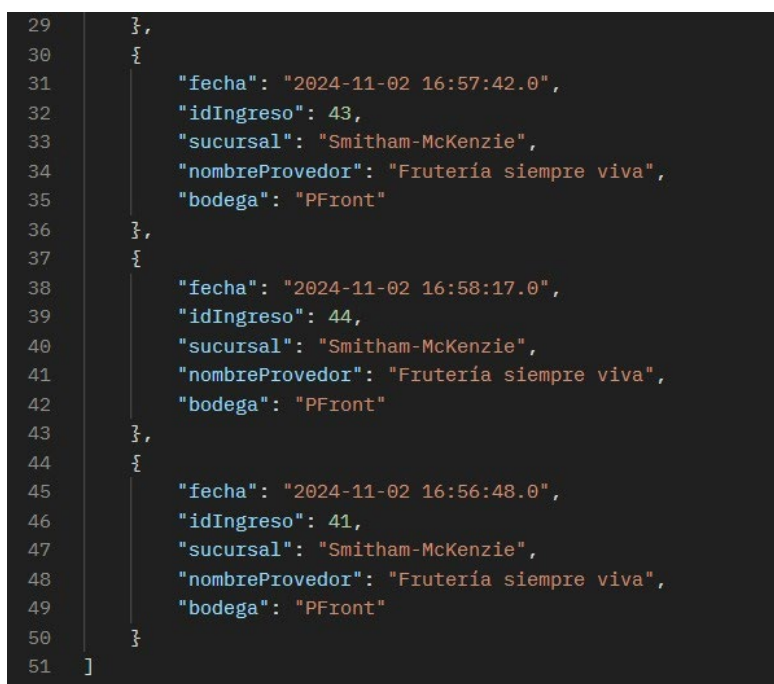
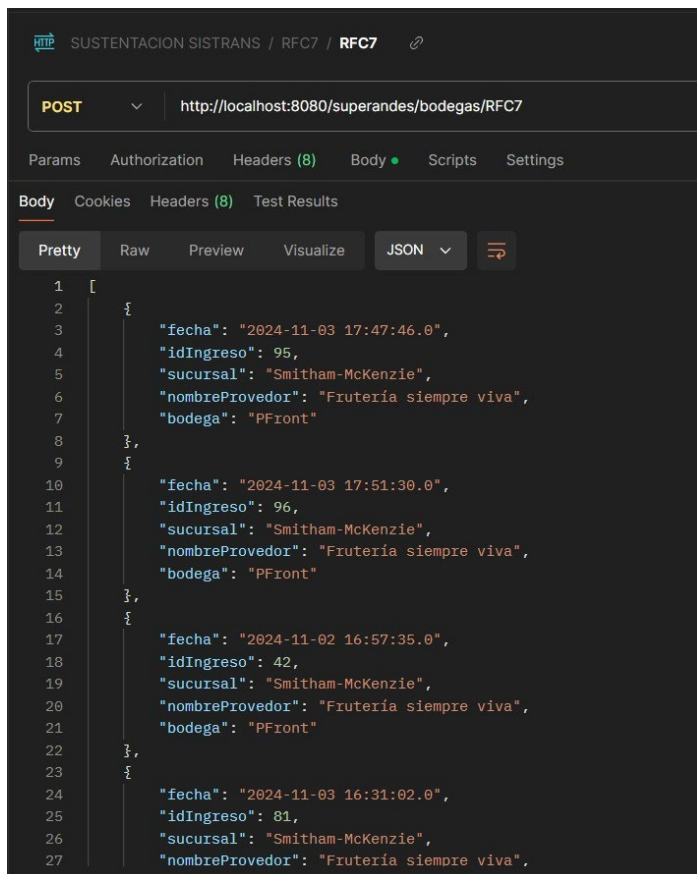
Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize Text

```
1 Transacción completada exitosamente
```

	IDINGRESO	FECHAINGRESO	BODEGA	ORDENCOMPRA
1	41	02/11/24	116	53
2	42	02/11/24	116	27
3	95	03/11/24	116	109
4	96	03/11/24	116	109
5	81	03/11/24	116	27
6	100	02/02/24	116	26
7	101	02/05/24	116	26
8	102	02/08/24	116	26
9	43	02/11/24	116	27
10	44	02/11/24	116	27

Entonces, vemos en la respuesta del RFC7 que SI aparece dicho ingreso, además de los demás que estaban.



Los pasos de ejecución siguen el orden de las fotos: ejecutar el RFC7, luego insertar un ingreso mediante el RF10, esperar al resultado de RFC10, y ver que si aparece la inserción, indicando que esta se realizó durante (o antes) la consulta.

Se evidencia que el RF10 NO espera a que termine el RFC7 para insertar un nuevo dato. El RFC7, al tener un aislamiento de tipo “read committed”, pone un candado sobre escrituras no confirmadas, por lo cual RF10 puede insertar efectivamente los datos antes de la consulta del RFC7. Por esto, el RFC7 muestra el dato nuevo, ya que fue agregado antes de la consulta.