Andres Felipe Bolivar Rojas

Universidad De Los Andes

Proyecto 2 – Sistemas transaccionales

1. Para esta entrega se tuvo que hacer modificaciones en la tabla disponibilidad para poder hacer los ajustes necesarios en del RF8-9 (Adjunto imagen de los cambios realizados).

```
93  CREATE TABLE DISPONIBILIDAD (
94      id_disponibilidad NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
95      cedula_conductor NUMBER(10) NOT NULL,
96      placa_vehiculo VARCHAR2(10) NOT NULL,
97      dia DATE NOT NULL,
98      franja_horaria VARCHAR2(50) NOT NULL,
99      tipo_transporte VARCHAR2(50) NOT NULL,
100     estado VARCHAR2(20) DEFAULT 'DISPONIBLE' NOT NULL,
101     CONSTRAINT fk_disponibilidad_conductor FOREIGN KEY (cedula_conductor) REFERENCES USUARIO(cedula),
102     CONSTRAINT chk_disponibilidad_estado CHECK (estado IN ('DISPONIBLE', 'ENSERVICIO')),
103     CONSTRAINT fk_disp_vehiculo FOREIGN KEY (placa_vehiculo) REFERENCES VEHICULO(placa)
104 );
```

También se tuvo que cambiar algo de lógica en DisponibilidadRepository , ServicioRepository, UsuarioRepository, ServicioService y UsuarioService. Todos los cambios asociados a usuario fueron necesarios para implementar el cambio de RF2 (que no sea necesario registrarse con una tarjeta) y los cambios en disponibilidad y servicio fueron necesarios para poder implementar de manera exitosa los RF8,9.

En la entrega pasada ningún RF fallo, por lo que en esta entrega solo se hicieron cambios de los RF (2; 8; 9) mencionados en el documento.

En la entrega pasada ningún RFC fallo, por lo que en esta entrega solo se hicieron cambios del RFC1 mencionado en el documento.

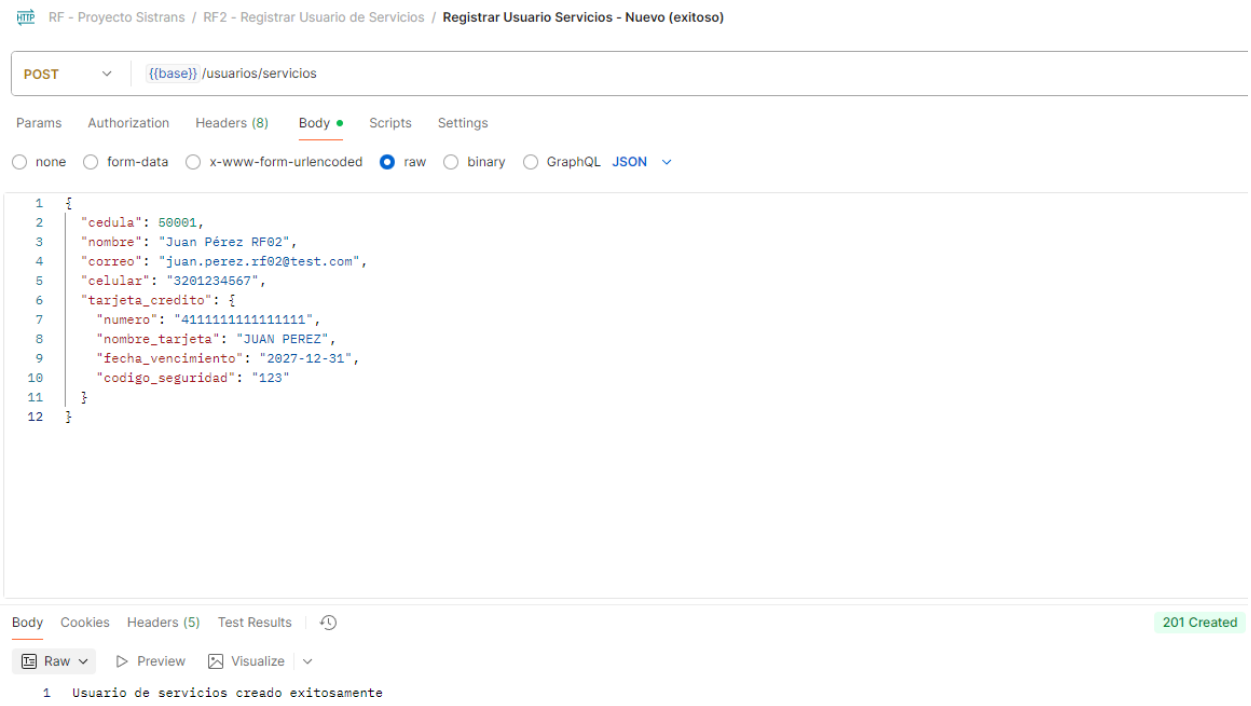
De los cambios mas destacados es que ahora cuando se crean un conductor, este va a tener una disponibilidad asociada, la cual por defecto siempre va a ser **“DISPONIBLE”,** y cuando se pide un servicio el estado del conductor será **“ENSERVICIO”** , lo que indicara que el conductor se encuentra prestando un servicio en la plataforma y no está disponible

Otro cambio que también fue muy relevante es que ahora se quitó la verificación que existía en Usuario la cual prohibía crear un usuario sin datos de una tarjeta.

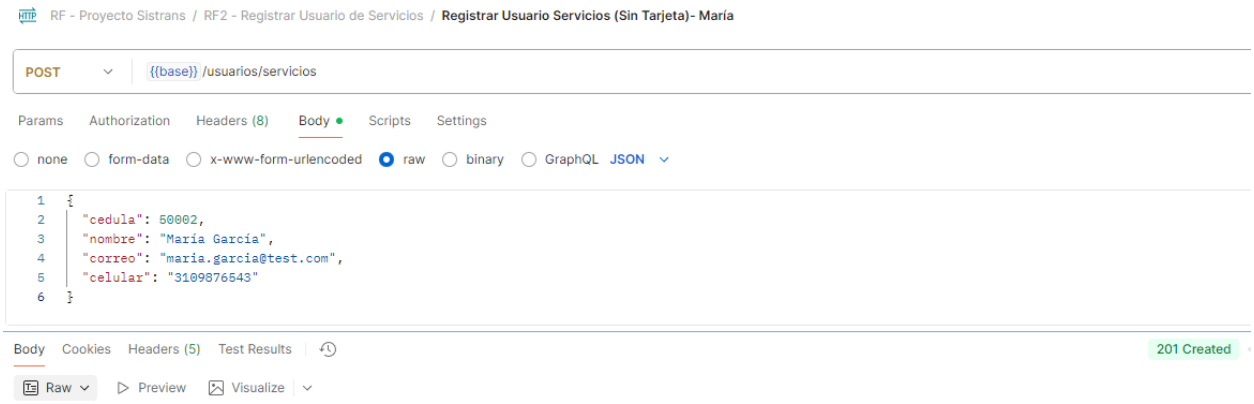
2.

PRUEBA RF2(Crear Usuario):

Una vez tenemos el RF2 implementado, ejecutamos en el postman la creación del servicio y vemos que ahora podemos crear usuarios con tarjeta de crédito:



Y de la misma forma podemos crear usuarios sin tarjeta de crédito:



Prueba RF8 (SOLICITAR SERVICIO):

Una vez tenemos el RF8 implementado, ejecutamos en el postman la creación del servicio

RF - Proyecto Sistrans / RF8 - Solicitar Servicio / **Solicitar Servicio Estándar (exitoso)**

POST {{base}}/servicios/solicitar

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
  "cedula_solicitante": 50001,
  "id_punto_partida": 1,
  "ids_puntos_destino": [2],
  "tipo_servicio": "TRANSPORTE_PASAJEROS",
  "nivel_transporte": "ESTANDAR",
  "numero_tarjeta": "4111111111111111"
}
```

Body Cookies Headers (5) Test Results

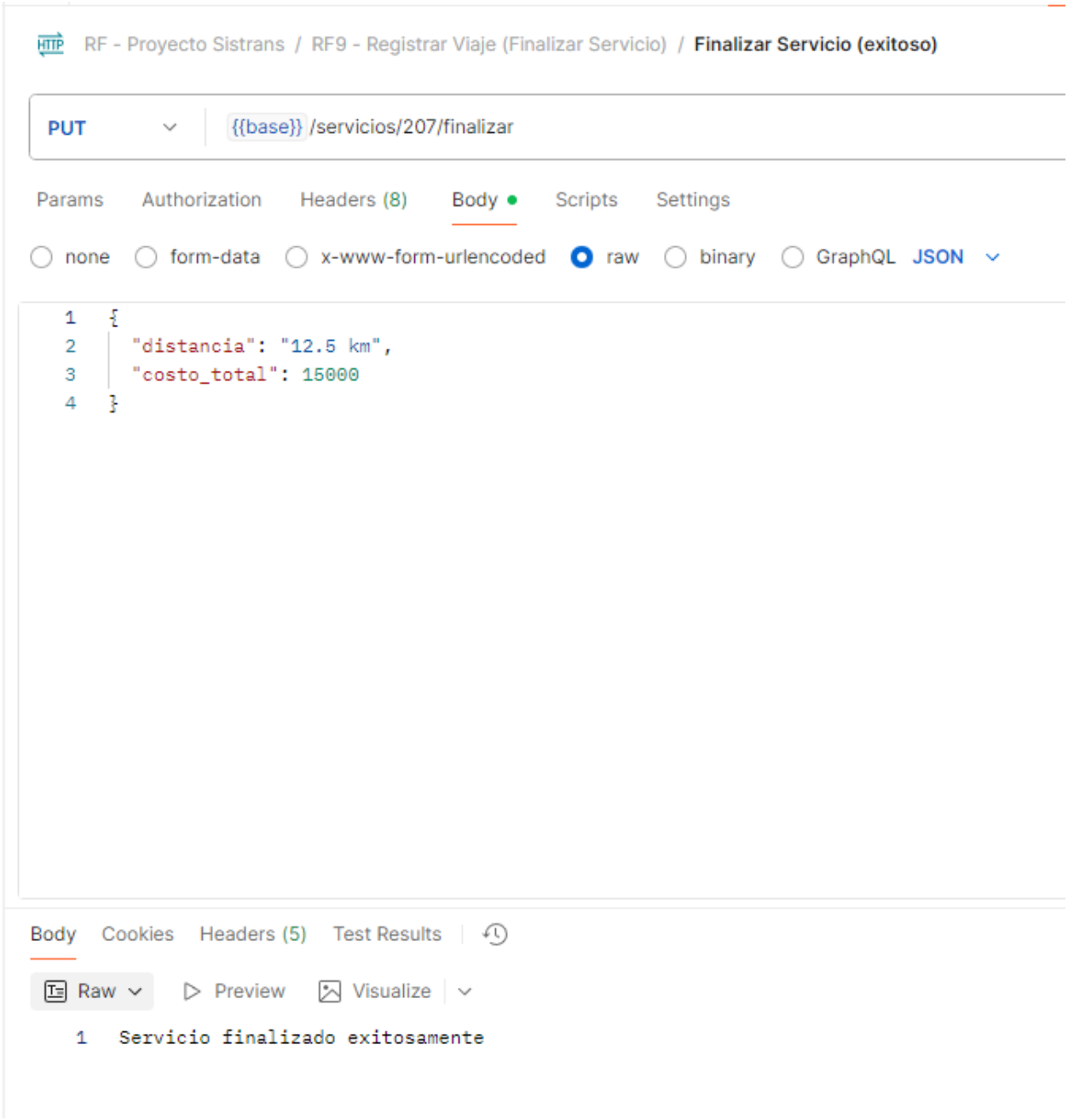
{ } JSON Preview Visualize


```
{
    "id": 207,
    "fecha_hora_inicio": "2025-11-02T15:32:02.1694356",
    "fecha_hora_fin": null,
    "distancia": "5km",
    "costo_total": 20000.0,
    "tipo": "TRANSPORTE_PASAJEROS",
    "cedula_solicitante": 50001,
    "cedula_conductor": 1004,
    "placa_vehiculo": "AUT004",
    "id_punto_partida": 1,
    "tarjeta_credito": "4111111111111111"
}
```

Y este se crea exitosamente, en hora inicio tenemos por defecto que se ponga la hora a la que se ejecuto esta prueba, y en distancia tenemos por defecto 5km, en costo total esta por defecto 20000. Con esto probamos que el servicio se creó de forma exitosa.

Prueba RF9 (FINALIZAR SERVICIO):

Ahora en el RF9 corremos esta prueba en postman



NOTE QUE usamos el mismo id de la prueba pasada para verificar que esta funcionando de manera correcta. Y le pasamos como distancia 12.5km y un costo total de 15000 , adicionalmente

se le debería agregar por defecto fecha_hora_fin , la hora en la que se ejecuto la prueba .Para verificar que esto se guardó de manera exitosa ejecutamos lo siguiente en Oracle:

```
SELECT * FROM SERVICIO
WHERE ID=207;
```

Y el resultado de esta consulta es la siguiente:

	ID	FECHA_HORA_INICIO	FECHA_HORA_FIN	DISTANCIA	COSTO_TOTAL	TIPO	CEDULA_SOLICITANTE	CEDULA_CONDUCTOR	PLACA_VEHICULO	ID_PUNTO_PARTIDA	TARJETA_CREDITO
1	207	02/11/25 03:42:12,536282000 PM	02/11/25 03:54:34,483141000 PM	12.5 km	15000	TRANSPORTE_PASAJEROS	50001	1004	AUT004	1	4111111111111111

Como se puede observar ahora la hora fin no es null , la distancia es 12.5km y el costo total es de 15000.

Para confirmar que al terminar el servicio esto queda bien hacemos lo siguiente

```
SELECT estado FROM DISPONIBILIDAD WHERE cedula_conductor = 1004
```

Y logramos ver el resultado que es correcto, el conductor están disponible luego de finalizar el servicio

	ESTADO
1	DISPONIBLE

3. IMPLEMENTACION RFC1

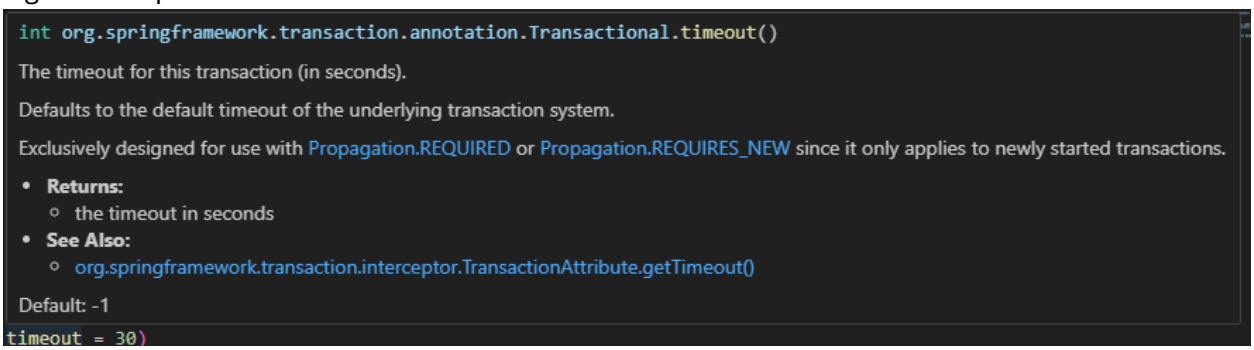
Para la implementación de esto se modificaron los archivos ReporteController , en los cuales se añadieron las 2 transacionalidades requeridas para esta entrega

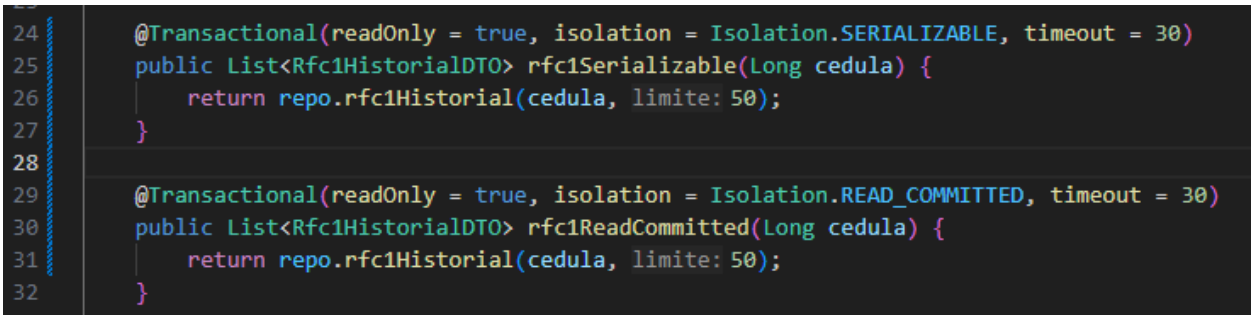
```
    // GET /rfc1/read-committed?cedula=50001
    @GetMapping("/read-committed")
    public List<Rfc1HistorialDTO> rfc1ReadCommitted(@RequestParam Long cedula) {
        return service.rfc1ReadCommitted(cedula);
    }

    // GET /rfc1/serializable?cedula=50001
    @GetMapping("/serializable")
    public List<Rfc1HistorialDTO> rfc1Serializable(@RequestParam Long cedula) {
        return service.rfc1Serializable(cedula);
    }
```

Tambien en RFCRepository cambio un poco la forma de como leemos los datos y en ReporteService se añadió el timeout=30 , esta notación esta en segundos como se aprecia en la

siguiente captura

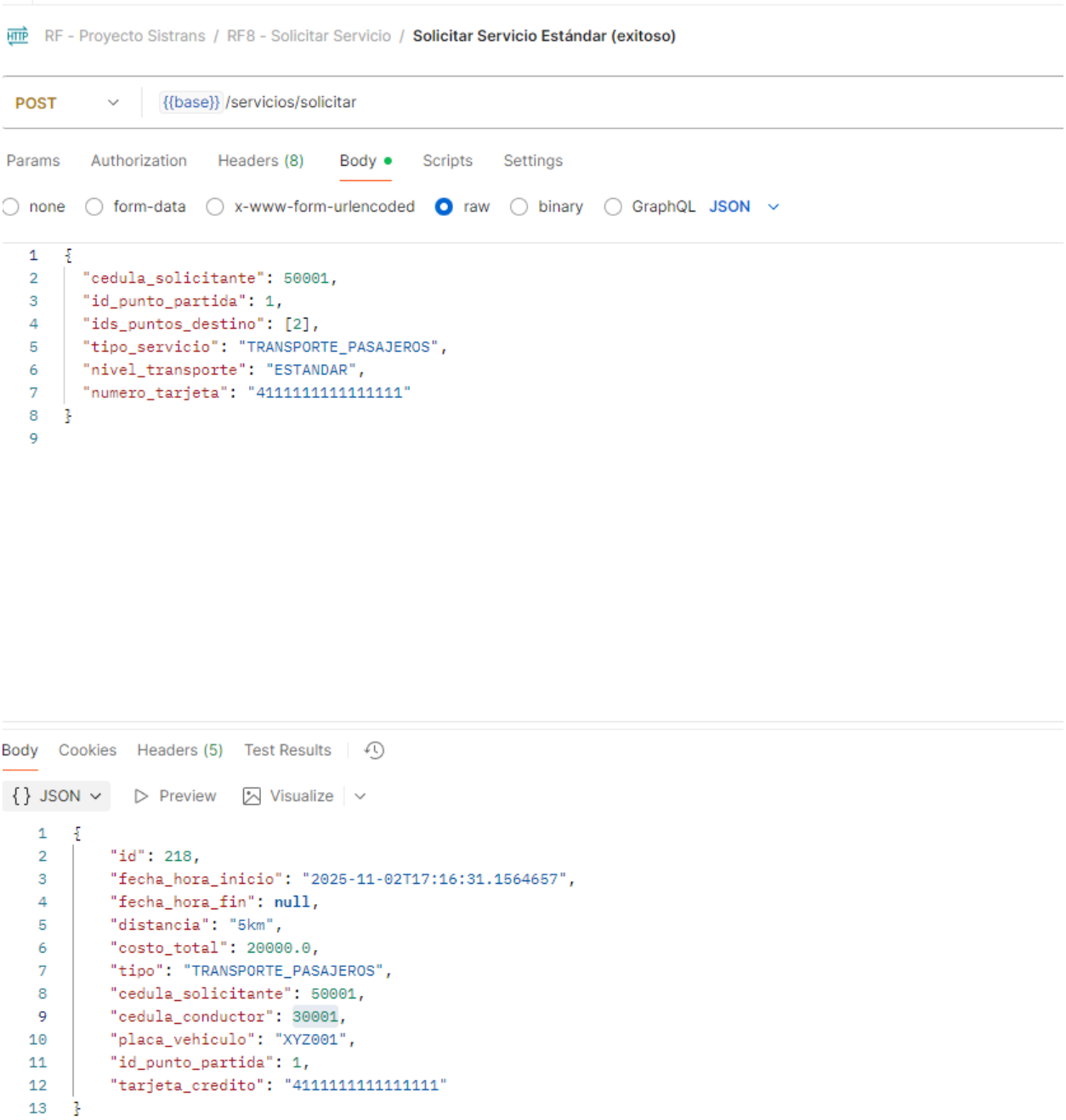




4. **PRUEBA RFC1 (Serializable)**

Forma de ejecutarlo:

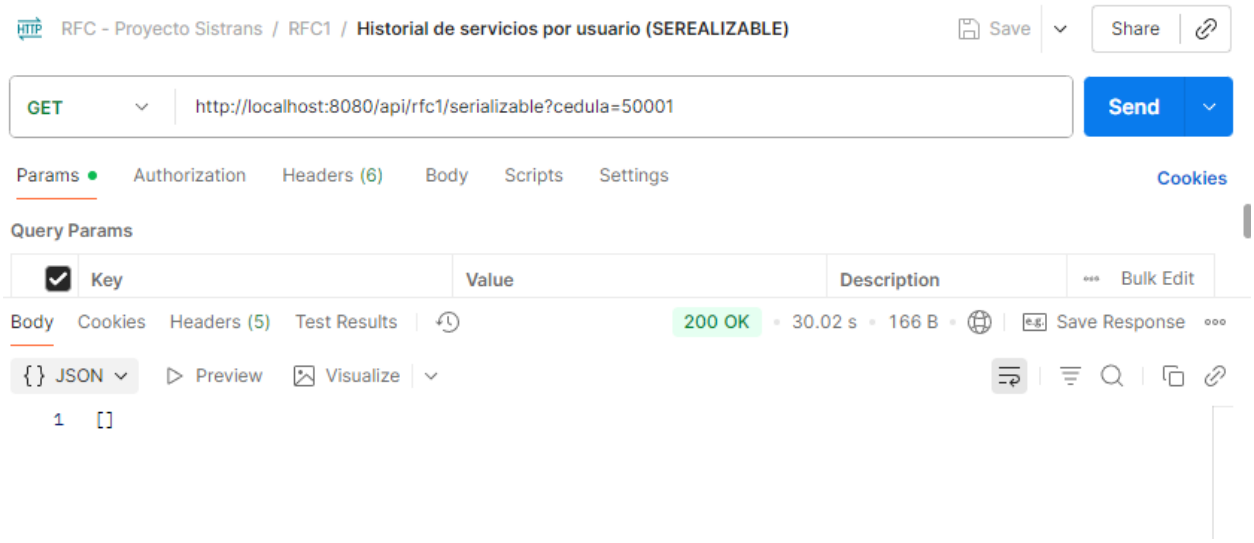
Lo primero que hacemos es para ejecutar este RFC1 SERIALIZABLE es: Ejecutar el codigó del proyecto , luego debes importar a postman el environment y los RFC/RF POSTMAN.JSON . Una vez ya tienes preparado el ambiente tienes que asegurarte que tus tablas están limpias usando el archivo “limpiar.sql” y después tenemos que poblar las tablas con el archivo “población.sql” y luego ejecutas los RF2 (registrar usuario usado en la prueba), RF3(registrar conductor usado en la prueba) , RF4(registrar vehículo usado en la prueba) Y RF5 (Registrar disponibilidad usada en la prueba).Ya con todo esto listo , lo que se tiene que hacer es ejecutar el RFC1(Serializable) , y antes de que pasen 30 segundos vas a ejecutar el RF8 .



Resultado:

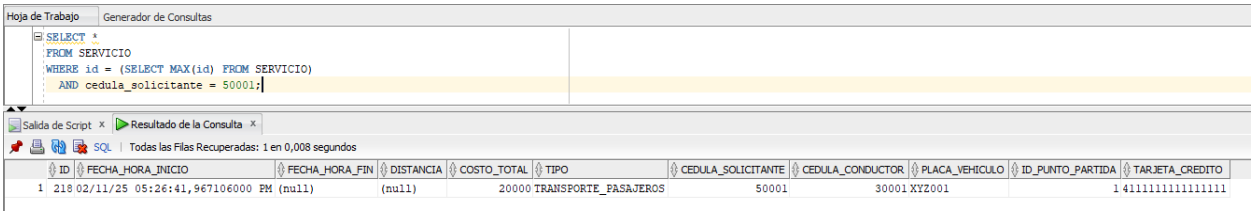
Al esperar los 30 segundos que le tenemos configurados al RFC1 , nos arroja una lista vacia (sin ningún elemento)

En este caso no aparece ningún servicio registrado para el usuario con cedula= 50001 , esto sucede ya que como estamos usando el modo de transacción “serializable”, solo se pueden ver las transacciones realizadas antes de la consulta , ya que las transacciones serializables son lineales, con esto confirmamos que no hay una lectura fantasma en el RFC.



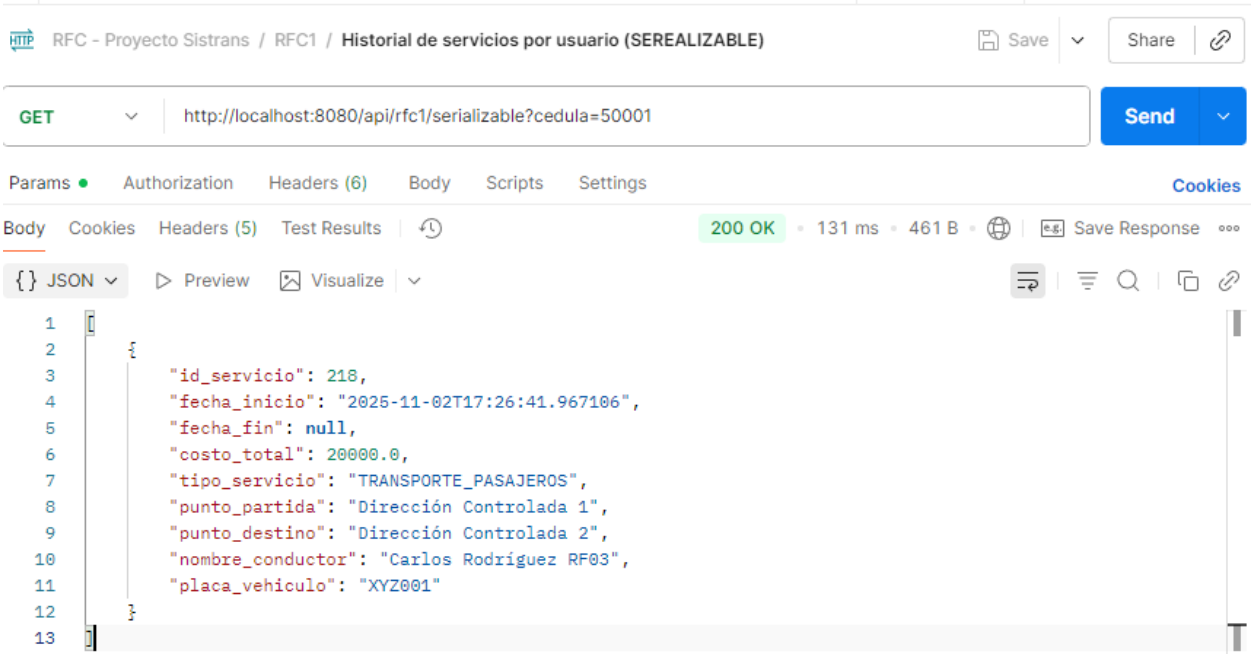
Verificacion en Oracle:

Al ejecutar esto , logramos verificar que si se guardo el RF8 , pero no lo logramos ver en la consulta por el modo de transacción que teníamos configurado



Como hacer que si salga el RF8 en esta consulta?

Para permitir que el resultado del RF8 salga en la consulta, primero tenemos que ejecutar el RF8 , y luego pasados los 30segundos la consulta nos mostrara esta información de manera correcta



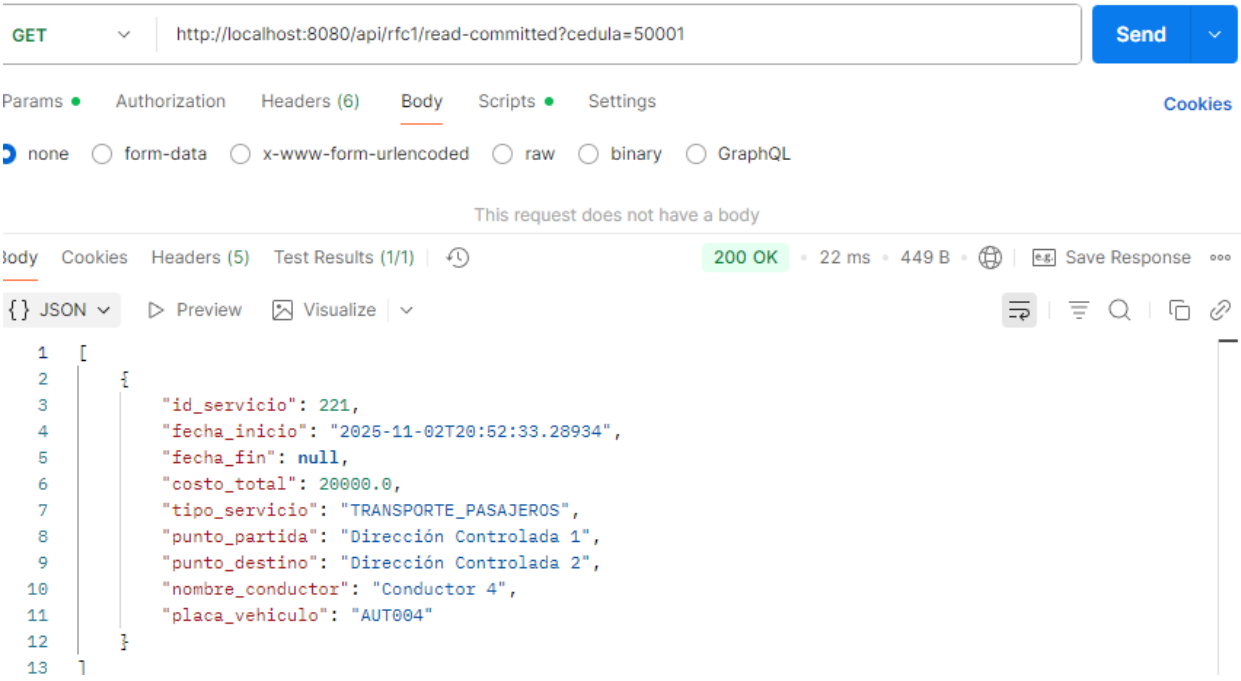
Y como podemos observar , ahora si obtenemos la información deseada.

5. **PRUEBA RFC1 (Read Commited)**

Forma de ejecutarlo:

Lo primero que hacemos para ejecutar este RFC1 Read Commited es: ejecutar el codigó del proyecto , luego debes importar a postman el environment y los RFC/RF POSTMAN.JSON . Una vez ya tienes preparado el ambiente tienes que asegurarte que tus tablas están limpias usando el archivo “limpiar.sql” y después tenemos que poblar las tablas con el archivo “población.sql” y luego ejecutas los RF2 (registrar usuario usado en la prueba), RF3(registrar conductor usado en la prueba) , RF4(registrar vehículo usado en la prueba) Y RF5 (Registrar disponibilidad usada en la prueba).Ya con todo esto listo , lo que se tiene que hacer es ejecutar el RFC1(Read Commited) , y antes de que pasen 30 segundos vas a ejecutar el RF8 .

Resultado:



Al esperar los 30 segundos configurados en RFC1, la consulta retorna la lista de servicios incluyendo el registro creado por RF8.

Esto ocurre porque RF8 realizó COMMIT antes de que RFC1 ejecutara el SELECT. RFC1 no bloquea ni espera a RF8; simplemente se queda en pausa durante los 30 segundos definidos. Cuando finalmente ejecuta la consulta, si el COMMIT de RF8 ya sucedió, RFC1 alcanza a ver la nueva orden registrada.

En este escenario se evidencia una lectura no repetible (lectura fantasma) propia del nivel de aislamiento READ COMMITTED, donde RFC1 nunca ve datos sin confirmar, pero sí puede ver nuevas filas que hayan sido comiteadas durante su tiempo de espera. Por eso, al hacer commit RF8 dentro de los 30 segundos, la orden recién ingresada aparece en el resultado final de RFC1.

Verificacion en Oracle:

```
SELECT * FROM SERVICIO
WHERE id= 221
```

	ID	FECHA_HORA_INICIO	FECHA_HORA_FIN	DISTANCIA	COSTO_TOTAL	TIPO	CEDULA_SOLICITANTE	CEDULA_CONDUCTOR	PLACA_VEHICULO	ID_PUNTO_PARTIDA	TARJETA_CREDITO
1	221	02/11/25 08:52:33,289340000 PM	(null)	(null)	20000	TRANSPORTE_PASAJEROS	50001	1004	AUT004	1	4111111111111111