Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148

# Proyecto AlpesCab

Entrega 3

*Sistemas Transaccionales*

### Visión general del dominio

ALPESCAB es una plataforma transaccional para coordinar servicios de movilidad y logística (pasajeros, comida, mercancías) entre usuarios de servicios (pasajeros/clientes) y usuarios conductores. El sistema gestiona ciudades, puntos geográficos, disponibilidad de conductores y vehículos, solicitudes de servicio, viajes, pagos multimétodo y reseñas bilaterales, siguiendo reglas de negocio que garantizan consistencia, trazabilidad y no solapamiento operativo.

## 1. Análisis y corrección de los RF de AlpesCab

### 1.1. Implementación transaccional y cobertura de requerimientos

Arquitectura y lineamientos de implementación

- **Capas:** Controller → Service (transaccional) → Repository (SQL nativo).
- **Control transaccional:** todos los métodos de la capa Service que modifican estado están anotados con @Transactional; cualquier excepción de negocio (IllegalArgumentException/IllegalStateException/RuntimeException) provoca rollback automático.
- **Repositorios SQL nativo:** se usan @Query(nativeQuery=true) y @Modifying para INSERT/UPDATE/DELETE; SELECTs de proyección simples para consultas.
- **Normalización y validación:** catálogos en mayúscula, formato horario HH:mm:ss, validaciones de rangos, FK existentes y unicidades.
- **Ids y secuencias Oracle**: se emplean secuencias SOLICITUD_SEQ, VIAJE_SEQ y PAGO_SEQ; cuando aplica, se consulta NEXTVAL en una única transacción.
- **Bloqueos y concurrencia:** selección de disponibilidad con “FOR UPDATE SKIP LOCKED” para evitar doble asignación (RF8).

- **Manejo de errores:**
  - 400: parámetros inválidos (validaciones).

  - 404: entidad no encontrada (FKs, búsquedas).
  - 409: conflictos de negocio (duplicados, solapes).
  - 500: errores internos o violaciones inesperadas de integridad.

- **Endpoints y pruebas:** endpoints REST consistentes; pruebas con Postman/cURL; para alta de datos se admite x-www-form-urlencoded y JSON según el caso.
- **Seguridad de datos:** no se persiste CVV; número de tarjeta validado con Luhn; se promueve enmascaramiento/mascarado en respuestas.

## 1.2. Cambios y correcciones frente a la Entrega 1 y 2

Nosotros teníamos en la anterior entrega todos los requerimientos funcionales excepto el RFC1, pero hicimos varios cambios para mejorar la optimización de la aplicación, por lo que realizamos lo siguiente:

- validación por conductor/día y por vehículo/día, tanto en registro como en modificación (RF5/RF6).
- Puntos geográficos: unicidad por (UPPER(nombre), UPPER(dirección), ciudad) y por (ciudad, latitud, longitud). Limpieza de duplicados previa a crear índices.
- **Tarjetas de crédito:** validación Luhn, vigencia, y pertenencia al usuario.
- **RF8 orquestado:** flujo atómico Solicitud → Viaje → Pago con bloqueo de disponibilidad; rollback total si falla cualquiera.
- **Finalización de viaje:** actualización de hora_fin y liberación lógica del conductor (no viajes abiertos simultáneos).
- **Reseñas bilaterales:** roles consistentes (USR y COND) y solo sobre viajes finalizados.
- **Consultas:** RFC1 con versiones READ COMMITTED y SERIALIZABLE con temporizador de 30s y doble lectura; RFC2–RFC4 con agregados e índices de apoyo.

Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148

# 2. Implementación del RF8

## 2.1. Objetivo del RF8

Registrar, en una sola transacción, la solicitud de servicio, la asignación del conductor/vehículo disponible, el inicio del viaje y el pago asociado, garantizando consistencia y rollback total ante cualquier error.

## 2.2. Contenido

1. Descripción general:

Este requerimiento implementa el flujo completo para que un usuario de servicios cree una solicitud y el sistema asigne un conductor con disponibilidad activa, calcule el costo, inicie el viaje y registre el pago, todo dentro de una única transacción. Si alguna etapa falla (por ejemplo, el pago), se revierte todo: ni solicitud, ni viaje, ni pago quedan persistidos.

2. Reglas de negocio aplicadas

- Catálogos: TIPO ∈ {PASAJEROS, COMIDA, MERCANCIAS}; para PASAJEROS, NIVEL ∈ {ESTANDAR, CONFORT, LARGE}.
- Método de pago: TARJETA, EFECTIVO, WALLET o PSE.
- TARJETA: debe pertenecer al usuario, pasar Luhn y estar vigente (mes/año). Si no se envía idTarjeta, el sistema escoge una vigente del usuario.
- Disponibilidad: se selecciona por ciudad de origen, día actual y tipo de servicio, con ventana horaria activa; se excluyen conductores con viajes abiertos.
- Concurrencia: selección con bloqueo por “FOR UPDATE SKIP LOCKED” para evitar doble asignación del mismo conductor/vehículo.
- Integridad: un pago por viaje; todas las FKs deben existir (usuario, puntos, solicitud, viaje, tarjeta).

3. Validación de entrada:

normalización de TIPO y, si aplica, NIVEL; verificación de ids de usuario y puntos.

Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148

4. Validación de usuario/medio de pago:

si el método es TARJETA, se valida propiedad, vigencia y Luhn; si no se envía idTarjeta, se elige una vigente.

5. Puntos y ciudad:

se consultan partida y llegada; se toma la ciudad de origen desde el punto de partida.

6. Selección de disponibilidad:

se elige una disponibilidad compatible (ciudad, día, tipo y ventana horaria), bloqueándola con “FOR UPDATE SKIP LOCKED” y excluyendo conductores con viaje abierto.

7. Cálculo operativo:

se calcula distancia por haversine y costo con una tarifa simple (base + porKm × km × multiplicador por nivel en PASAJEROS).

8. Inserciones atómicas:
    a. Solicitud: se inserta la SOLICITUD_SERVICIO (estado=CREADA; fecha y fecha_solicitud=SYSDATE).
    b. Viaje: se inserta el VIAJE referenciando la solicitud (fecha_asignacion y hora_inicio=SYSDATE; hora_fin=NULL).
    c. Pago: se inserta el PAGO referenciado al viaje (método y, si aplica, idTarjeta; estado inicial=APROBADO para pruebas).

9. Respuesta: se retornan idViaje, idPago, método, idTarjeta (si aplica), idConductor, idVehiculo, distancia y costo.
10. Consideraciones técnicas de implementación

- Capa Service: método anotado con @Transactional que orquesta todo el flujo.
- Repositorios SQL nativo:
    - Disponibilidad: consulta con “FOR UPDATE SKIP LOCKED” por ciudad, día y tipo, filtrando conductores con viaje abierto.
    - Viaje y Solicitud: INSERT nativos alineados al DDL real (columnas y FKs vigentes).
    - Pago: método `insertarPagoConViaje` con FK a viaje y validaciones propias de método.

- Secuencias Oracle: generación de ids con NEXTVAL para SOLICITUD, VIAJE y PAGO.
- Manejo de errores:
    - Parámetros inválidos ⇒ 400.
    - Entidades inexistentes ⇒ 404.
    - Conflictos/duplicados ⇒ 409.
    - Violaciones/ref BD u otros ⇒ 500 (con rollback).

## 2.3. Escenarios de prueba:

Existen SECs Oracle (SOLICITUD_SEQ, VIAJE_SEQ, PAGO_SEQ), ciudad operativa, usuario de servicios con tarjeta vigente, conductor con vehículo en la misma ciudad y al menos una disponibilidad activa para el día y hora actual, dos puntos geográficos válidos (partida y llegada).

1. Caso 200 (método TARJETA con tarjeta vigente)

    a. Entrada (POST /solicitudes/solicitar, JSON):

    {"idUsuarioServicio": 11012,
    "idPuntoPartida": 40004,
    "idPuntoLlegada": 40005,
    "tipo": "PASAJEROS",
    "nivel": "CONFORT",
    "metodoPago": "TARJETA",
    "idTarjeta": 18005 }

    b. Resultado esperado: Código 200 OK.

- Cuerpo incluye: idViaje, idPago, metodoPago, idTarjeta, idConductor, idVehiculo, distanciaKm (> 0), costoTotal (> 0), tipo, nivel.

- Verificación en BD:
    - Existe fila en SOLICITUD_SERVICIO con estado = ‘CREADA’ y FK a idUsuarioServicio y puntos.
    - Existe fila en VIAJE con HORA_INICIO no nula, HORA_FIN nula, y FK a la solicitud.

- Existe fila en PAGO con FK a VIAJE, MONTO = costoTotal, FECHA = SYSDATE, ESTADO = ‘APROBADO’, METODO = ‘TARJETA’ y ID_TARJETA = 18005.

2. Caso “sin conductor disponible”

   No debe existir disponibilidad activa compatible en la ciudad y hora actual para el tipo solicitado (por ejemplo, borrar o mover horarios).

   a. Entrada:
   {"idUsuarioServicio": 11012,
   "idPuntoPartida": 40004,
   "idPuntoLlegada": 40005,
   "tipo": "COMIDA",
   "metodoPago": "EFECTIVO"}

   b. Resultado esperado: Respuesta de error (409 o 500 controlado según tu controlador) con mensaje “No hay conductor disponible en este momento”

- Verificación en BD: no se creó solicitud, viaje ni pago.

3. Caso “tarjeta no vigente o no pertenece al usuario”

   Usar idTarjeta vencida o perteneciente a otro usuario.

   a. Entrada:
   { "idUsuarioServicio": 11012,
   "idPuntoPartida": 40004,
   "idPuntoLlegada": 40005,
   "tipo": "PASAJEROS",
   "nivel": "LARGE",
   "metodoPago": "TARJETA",
   "idTarjeta": 19999
   }
   b. Resultado esperado: Error de negocio indicando “Tarjeta no vigente o no pertenece al usuario”.

Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148

- Verificación en BD: Sin inserciones en SOLICITUD_SERVICIO, VIAJE y PAGO.

4. Caso “falla de pago” (duplicado o restricción única)
   Forzar una violación de integridad al insertar PAGO, por ejemplo, intentando repetir el mismo idPago de manera manual en dos solicitudes concurrentes o activando una UQ que choque.
   a. Entrada:
      { "idUsuarioServicio": 11012,
      "idPuntoPartida": 40004,
      "idPuntoLlegada": 40005,
      "tipo": "PASAJEROS",
      "nivel": "CONFORT",
      "metodoPago": "TARJETA",
      "idTarjeta": 18005
      }

   b. Resultado esperado: Mensaje “No fue posible registrar el pago (duplicado o restricción BD)”.
- Verificación en BD: no debe quedar viaje ni solicitud “huérfanos”. La transacción se revierte por completo.

5. Caso “método de pago alterno” (EFECTIVO/WALLET/PSE)
   a. Entrada:
      {"idUsuarioServicio": 11012,
      "idPuntoPartida": 40004,
      "idPuntoLlegada": 40005,
      "tipo": "MERCANCIAS",
      "metodoPago": "EFECTIVO"
      }

   b. Resultado esperado: 200 OK con idViaje e idPago; metodoPago = “EFECTIVO”; idTarjeta = null.
- PAGO insertado con METODO = “EFECTIVO” y ESTADO = “CONFIRMADO” y solicitud y viajes creados de forma consistente.

Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148

## 2.4. Evidencia con Postman

- curl -X POST "http://localhost:8080/solicitudes/solicitar" ^
  -H "Content-Type: application/json" ^
  -d "{ "idUsuarioServicio":11012, "idPuntoPartida":40004, "idPuntoLlegada":40005, "tipo":"PASAJEROS", "nivel":"CONFORT", "metodoPago":"TARJETA", "idTarjeta":18005 }"

- curl -X POST "http://localhost:8080/solicitudes/solicitar" ^
  -H "Content-Type: application/json" ^
  -d "{ "idUsuarioServicio":11012, "idPuntoPartida":40004, "idPuntoLlegada":40005, "tipo":"PASAJEROS", "nivel":"ESTANDAR", "metodoPago":"TARJETA" }"

- curl -X POST "http://localhost:8080/solicitudes/solicitar" ^
  -H "Content-Type: application/json" ^
  -d "{ "idUsuarioServicio":11012, "idPuntoPartida":40004, "idPuntoLlegada":40005, "tipo":"COMIDA", "metodoPago":"EFECTIVO" }"

- curl -X POST "http://localhost:8080/solicitudes/solicitar" ^
  -H "Content-Type: application/json" ^
  -d "{ "idUsuarioServicio":11012, "idPuntoPartida":40004, "idPuntoLlegada":40005, "tipo":"PASAJEROS", "nivel":"LARGE", "metodoPago":"TARJETA", "idTarjeta":19999}"

# 3. Implementación del RFC1 en diferentes niveles de aislamiento

Implementar y evidenciar el RFC1 como transacción en dos niveles de aislamiento (READ_COMMITTED y SERIALIZABLE), incluyendo un temporizador de 30 segundos y dos lecturas de la misma consulta (antes y después del temporizador) para observar efectos de concurrencia frente al RF8 ejecutado en paralelo.

Aplica para el histórico de viajes de un usuario de servicios (consolidando información de solicitud, viaje, conductor, vehículo, puntos y pago). Se presenta versión READ_COMMITTED y versión SERIALIZABLE, cada una ejecutando la consulta dos veces dentro de la misma transacción con un “sleep” de 30 segundos.

Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148

1. Diseño general del RFC1 transaccional

El servicio expone dos endpoints equivalentes en funcionalidad, pero distintos en nivel de aislamiento: uno opera en READ_COMMITTED y otro en SERIALIZABLE. Ambos realizan exactamente dos lecturas del histórico de viajes de un usuario dentro de la misma transacción: una lectura inicial, una espera de 30 segundos y una segunda lectura. El objetivo es comparar si, durante esa ventana, operaciones concurrentes del RF8 (creación de solicitud, viaje y pago) afectan la visibilidad de los datos ya confirmados en la base de datos.

2. Lógica de la consulta de histórico

La consulta obtiene, para un idUsuarioServicio, la lista de viajes realizados, incluyendo al menos: identificador del viaje, fechas de asignación e inicio/fin, costo total, distancia, tipo y nivel, vehículo y conductor asignado, puntos de partida y llegada, y el estado del pago. Esta consulta se implementa en el repositorio de reportes mediante SQL nativo optimizado con las uniones necesarias a VIAJE, SOLICITUD_SERVICIO, USUARIO_CONDUCTOR, VEHICULO, PUNTO_GEOGRAFICO y PAGO. El resultado se devuelve como lista de filas mapeadas a una estructura simple (mapas o DTO liviano) para presentar en el controlador.

3. Endpoints propuestos para pruebas

- GET /reportes/usuario/{id}/servicios/read-committed

Transacción con nivel READ_COMMITTED. Dentro, hace la primera lectura, espera 30 segundos y hace la segunda lectura. La respuesta devuelve un objeto con “primeraLectura” y “segundaLectura”.

- GET /reportes/usuario/{id}/servicios/serializable

Transacción con nivel SERIALIZABLE. Dentro, hace la primera lectura, espera 30 segundos y hace la segunda lectura. La respuesta devuelve un objeto con “primeraLectura” y “segundaLectura”.

4. Comportamiento esperado por nivel de aislamiento
    a. READ_COMMITTED: La segunda lectura puede reflejar cambios confirmados por transacciones concurrentes ejecutadas durante la espera de 30 segundos (por ejemplo, un nuevo viaje creado por RF8 que haya hecho

commit en ese lapso). Por lo tanto, es posible que la segunda lectura incluya más filas que la primera.

b. SERIALIZABLE: La transacción mantiene una visión consistente como si todas las operaciones fueran serializadas. Dependiendo de la configuración del motor, la segunda lectura normalmente no mostrará cambios externos ocurridos después de la primera lectura mientras la transacción siga abierta, y puede generar bloqueos o reintentos si existen conflictos. En la práctica de Oracle, el nivel "SERIALIZABLE" emula una instantánea consistente de tiempo; se espera que la segunda lectura sea igual a la primera en cuanto a visibilidad de nuevas filas creadas por RF8 durante el "sleep".

5. Escenario de prueba: SERIALIZABLE vs RF8 concurrente

   Línea de tiempo:

   I. Iniciar RFC1 en /serializable para un usuario de servicios con histórico existente (o vacío).
   II. RFC1 hace la primera lectura y entra en temporizador de 30 segundos.
   III. Dentro de esos 30 segundos, otro cliente ejecuta RF8 (POST /solicitudes/solicitar) para el mismo usuario de servicios.
   IV. RF8 crea solicitud, viaje y pago y hace commit.
   V. Vuelve RFC1 y realiza la segunda lectura en SERIALIZABLE.

   Resultado esperado: La segunda lectura no debe incluir el nuevo viaje creado por RF8 (visión consistente), a pesar de que RF8 ya haya confirmado. Además, se puede describir si hubo espera o bloqueo percibido dependiendo de la competencia por recursos e índices involucrados en la consulta.

6. Escenario de prueba: READ_COMMITTED vs RF8 concurrente

   Línea de tiempo:

   I. Iniciar RFC1 en /read-committed para un usuario de servicios.
   II. RFC1 hace la primera lectura y entra en temporizador de 30 segundos.
   III. Durante esos 30 segundos, ejecutar RF8 (POST /solicitudes/solicitar) para el mismo usuario.
   IV. RF8 crea solicitud, viaje y pago y hace commit.

Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148

V. Vuelve RFC1 y realiza la segunda lectura en READ_COMMITTED.

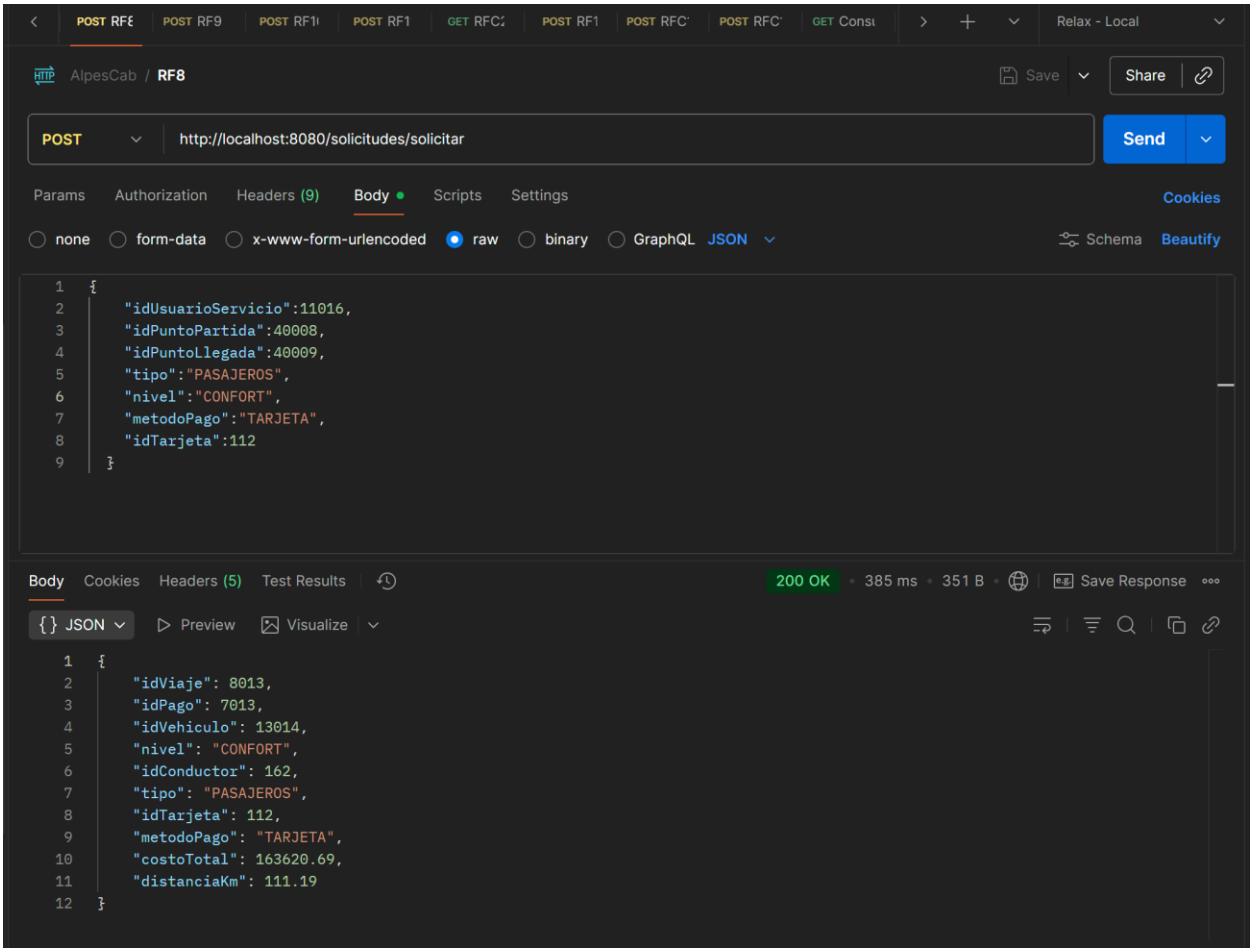
Resultado esperado: La segunda lectura debería incluir el viaje creado por RF8, pues fue confirmado durante la ventana de espera. La primera lectura no lo incluye; la segunda sí.

7. Resultados y evidencias

a) Primera lectura: listado de viajes, con número total de filas y ejemplo de contenido (id_viaje, fechas, costo, etc.).

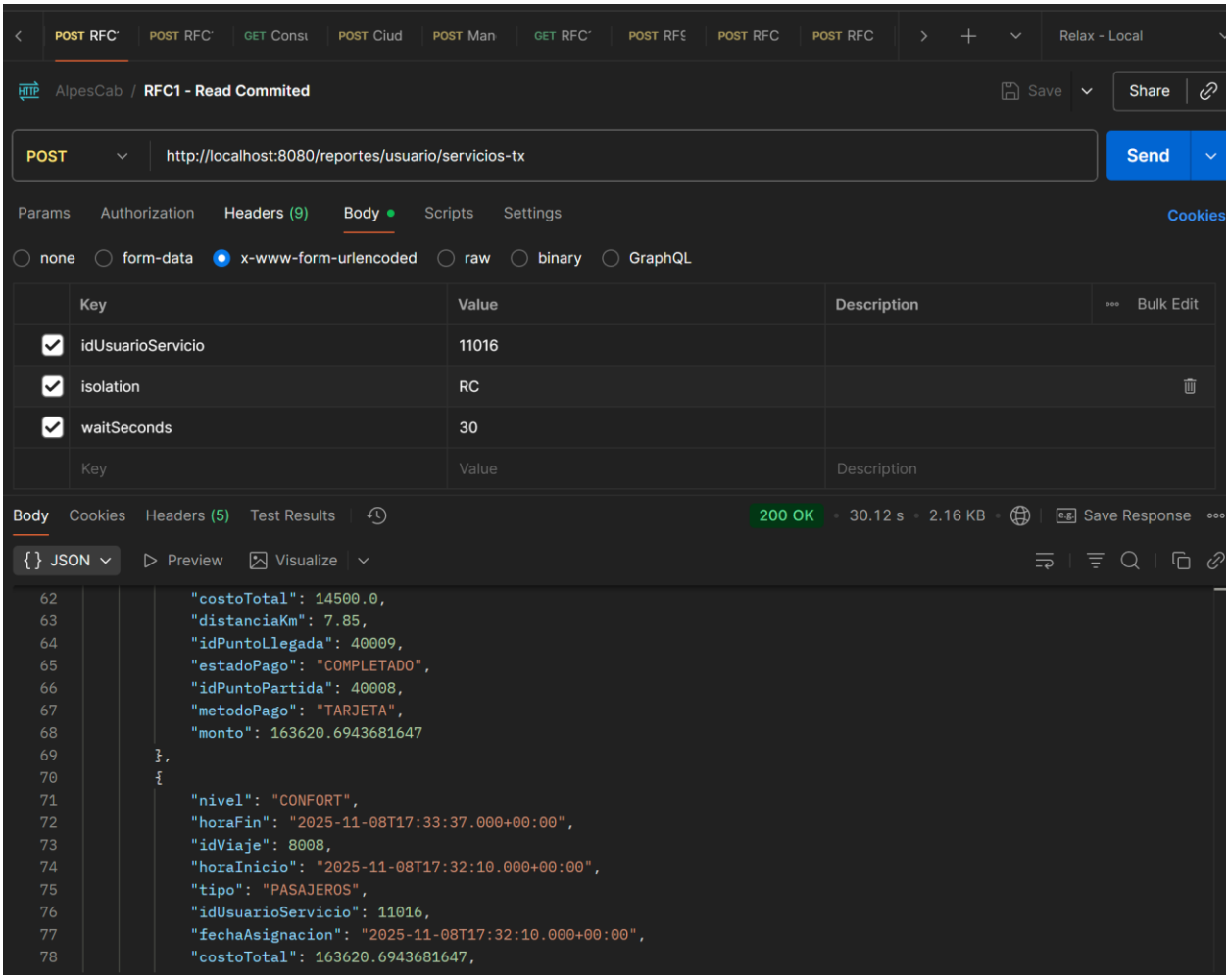
b) Marca temporal de inicio y fin de la espera de 30 segundos.
c) Segunda lectura: listado de viajes con número total de filas y diferencias visibles frente a la primera lectura.
d) Ejecución concurrente de RF8: request y response del POST /solicitudes/solicitar, con ids generados.

Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148

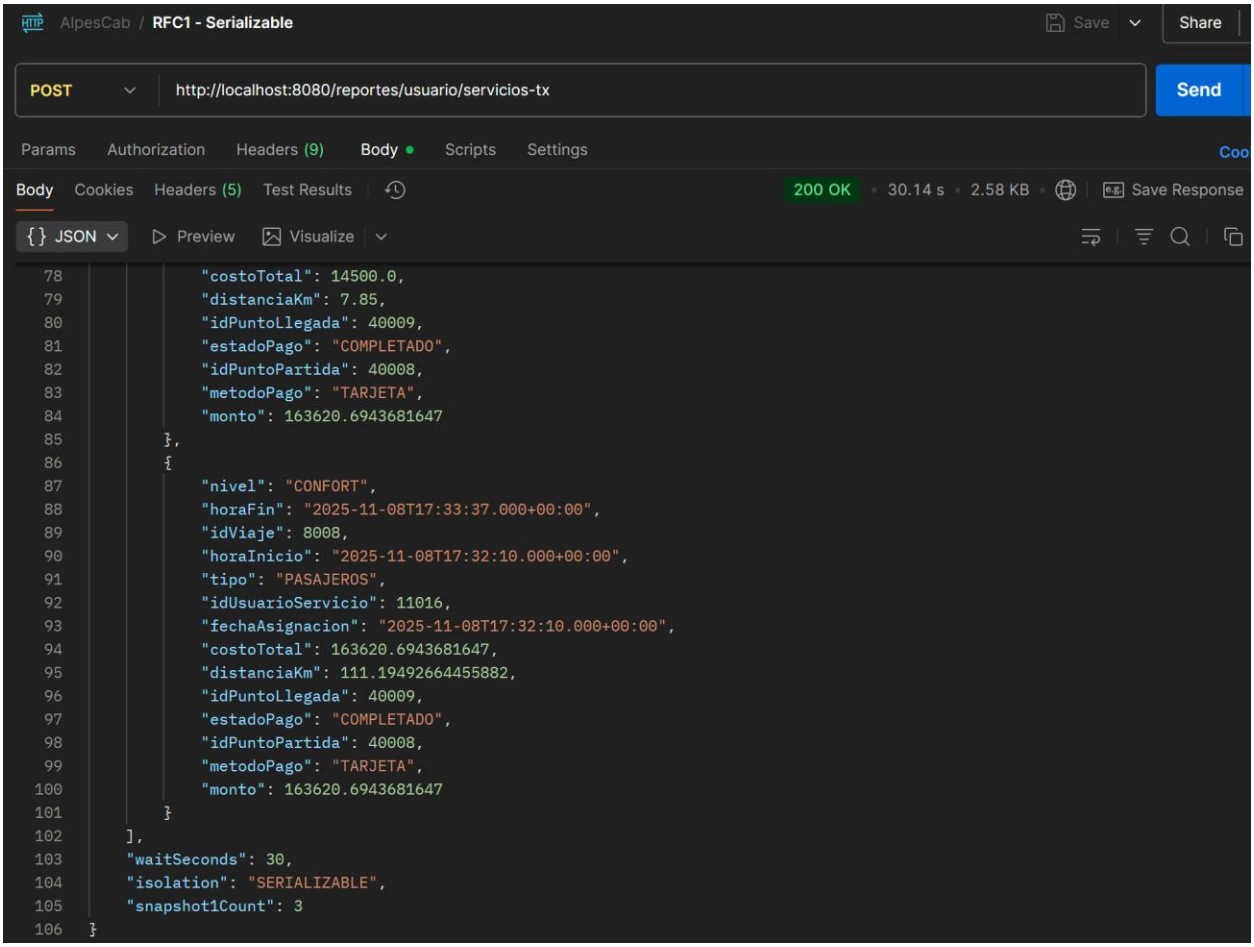



e) Conclusión por nivel de aislamiento: si apareció o no el nuevo viaje en la segunda lectura, tiempos y cualquier contención observada.

Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148



Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148




8. Observaciones técnicas

- La consulta se ejecuta dos veces en la misma transacción para el mismo usuario.
- La espera de 30 segundos se realiza dentro del método transaccional del servicio con un sleep controlado.

# 4.1. Escenarios de prueba (cobertura por requerimiento)

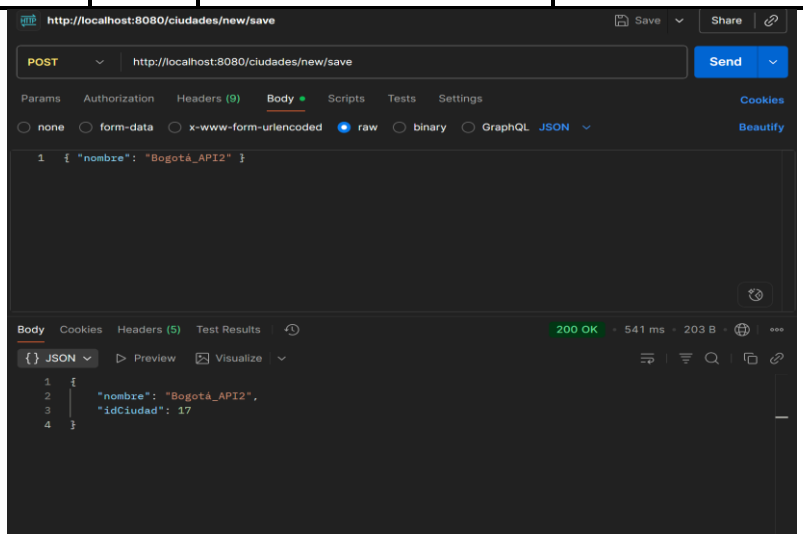
Cada RF con un caso exitoso y un caso de falla.

## RF1 Registrar ciudad

| Caso | Objetivo | Rol | Pasos | Resultado esperado |
|---|---|---|---|---|
| RF1 OK | Crear ciudad válida | ADM | Ingresar nombre “Barranquilla” y guardar | Ciudad creada, nombre único visible en catálogo |

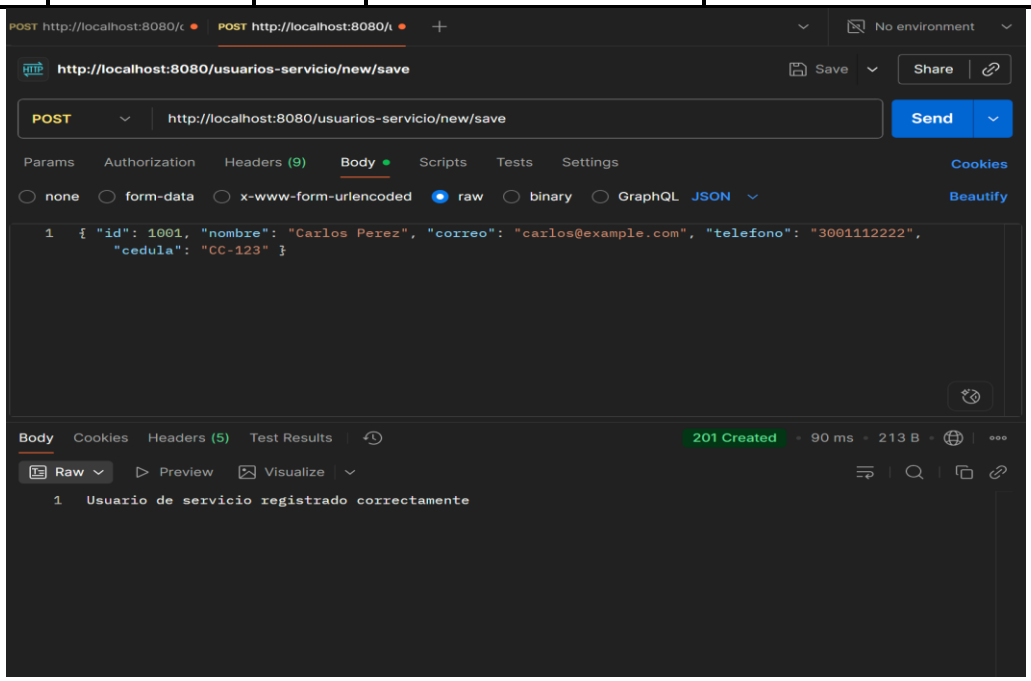
Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148

| | | | | |
|---|---|---|---|---|
| RF1 FAIL | Duplicado de nombre | ADM | Intentar crear “Bogotá” nuevamente | Rechazo por ND o mensaje de duplicado |




# RF2 Registrar UsuarioServicio

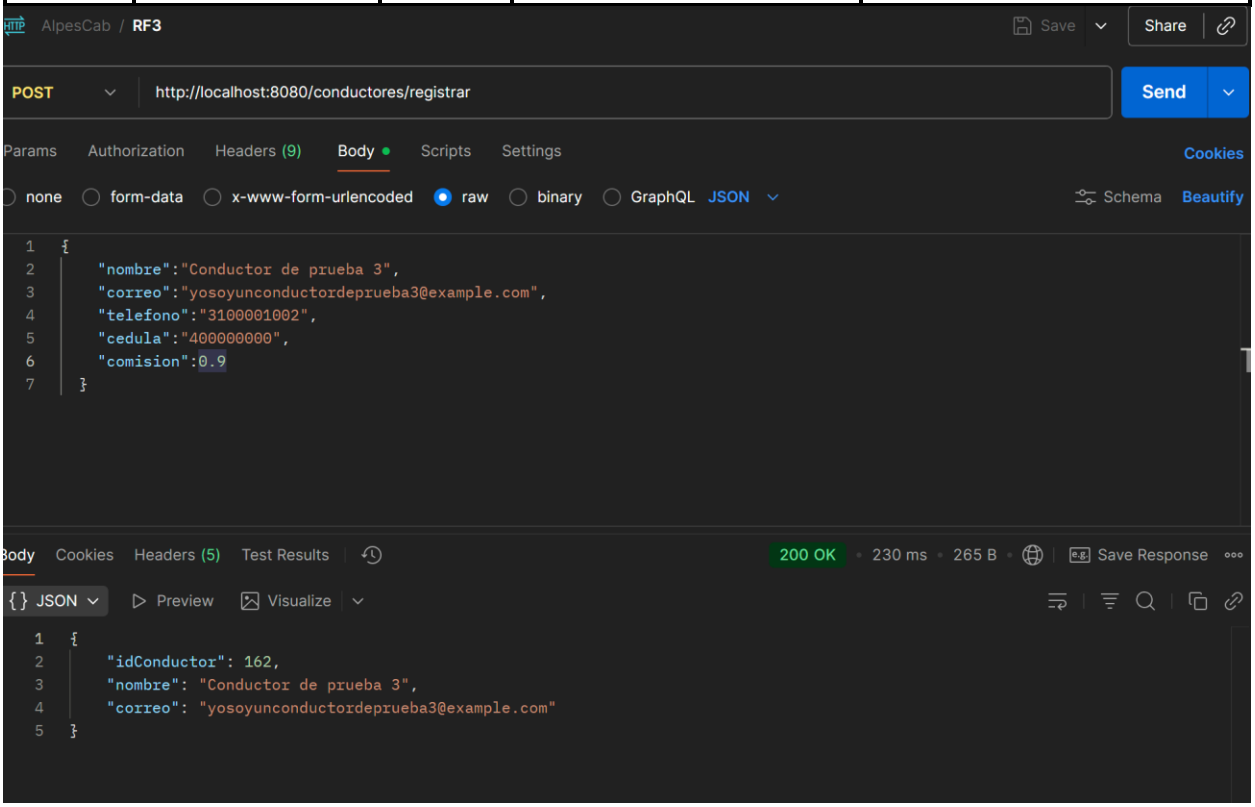
| Caso | Objetivo | Rol | Pasos | Resultado esperado |
|---|---|---|---|---|
| RF2 OK | Registrar cliente | USR | Completar nombre, correo, teléfono, cédula | UsuarioServicio creado con correo y cédula únicos |
| RF2 FAIL | Correo duplicado | USR | Registrar con correo de 1001 | Rechazo por ND correo |



Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148

# RF3 Registrar UsuarioConductor

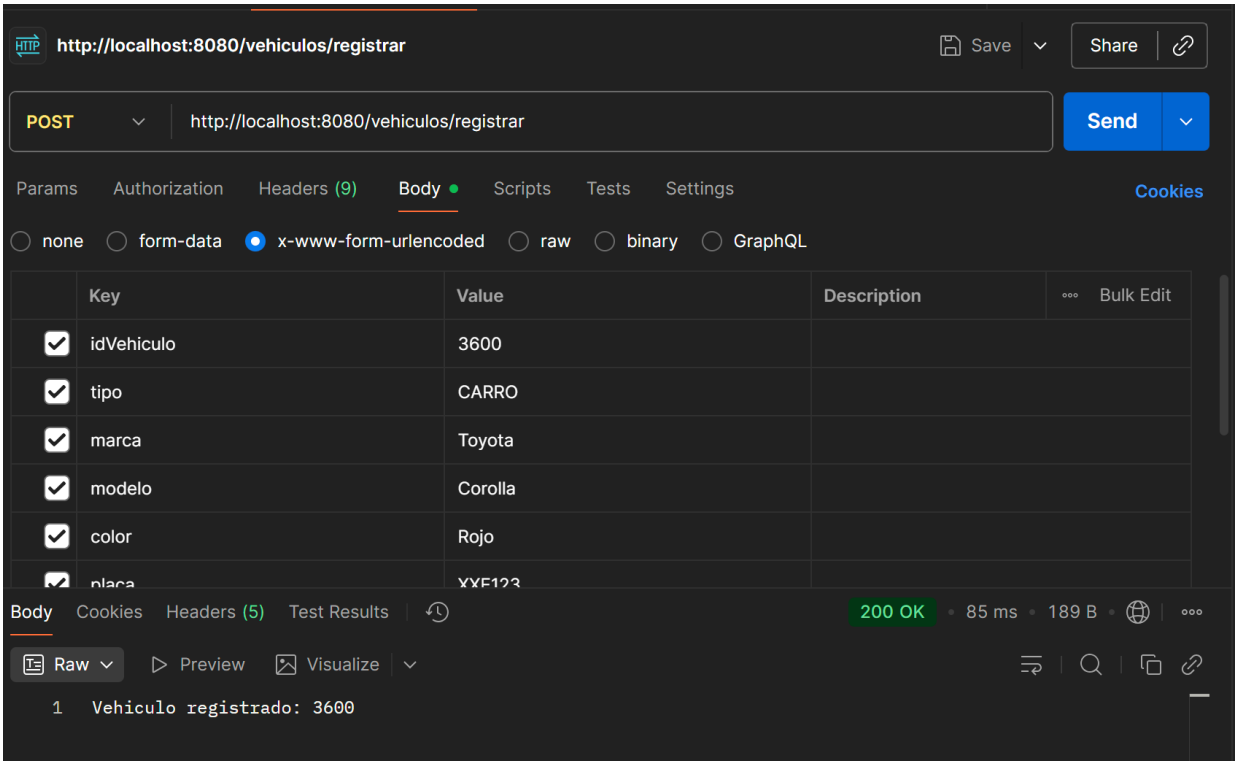
| Caso | Objetivo | Rol | Pasos | Resultado esperado |
|---|---|---|---|---|
| RF3 OK | Registrar conductor | COND | Completar datos y comision 0.6 | Conductor creado |
| RF3 FAIL | Comisión inválida | COND | comision 1.5 | Rechazo por check de comisión |




# RF4 Registrar Vehiculo

| Caso | Objetivo | Rol | Pasos | Resultado esperado |
|---|---|---|---|---|
| RF4 OK | Vehículo del conductor | COND | Seleccionar conductor 2001 y ciudad 1. Placa ABC999 | Vehículo creado. Placa única |
| RF4 FAIL | Ciudad inexistente | COND | Usar Ciudad 99 | Rechazo por FK a Ciudad |

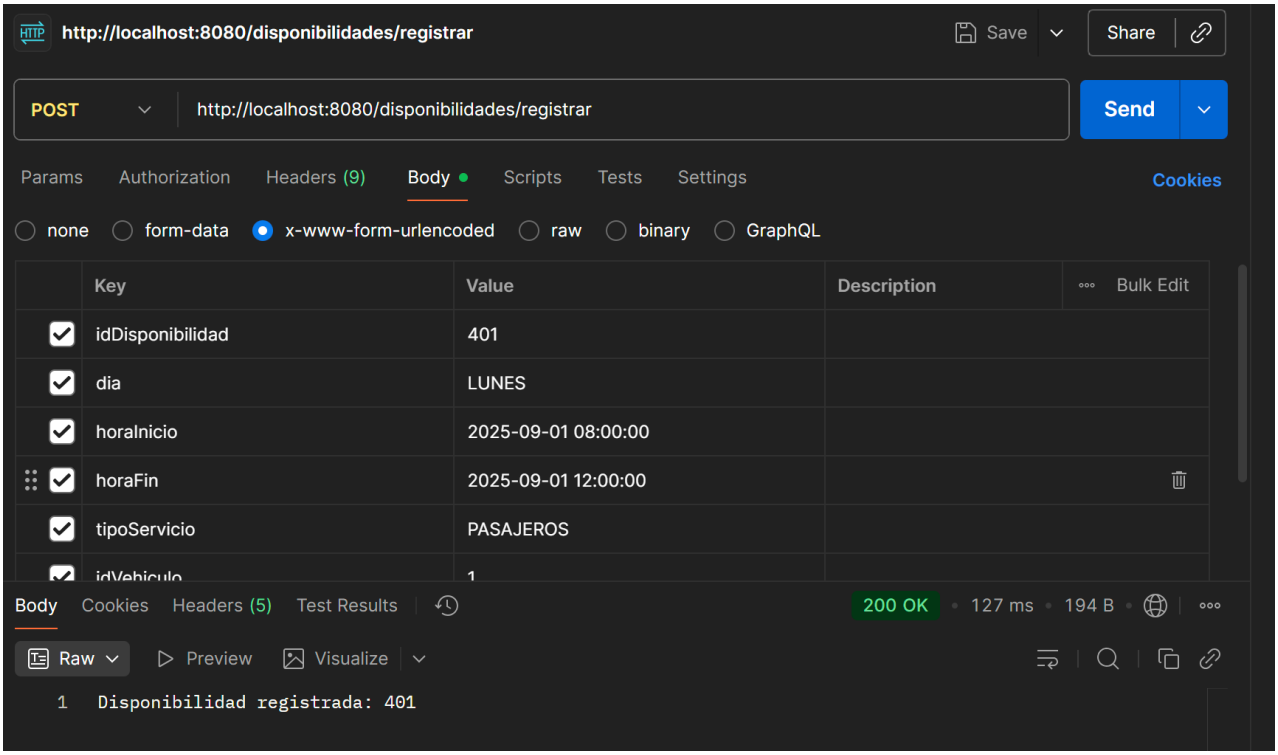
Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148




# RF5 Registrar disponibilidad de conductor

| Caso | Objetivo | Rol | Pasos | Resultado esperado |
|---|---|---|---|---|
| RF5 OK | Crear franja sin solape | COND | Lunes 08:00 a 12:00 para 3001 | Disponibilidad creada |
| RF5 FAIL | Solape | COND | Crear Lunes 10:00 a 11:00 para 3001 | Rechazo por trigger de solape |

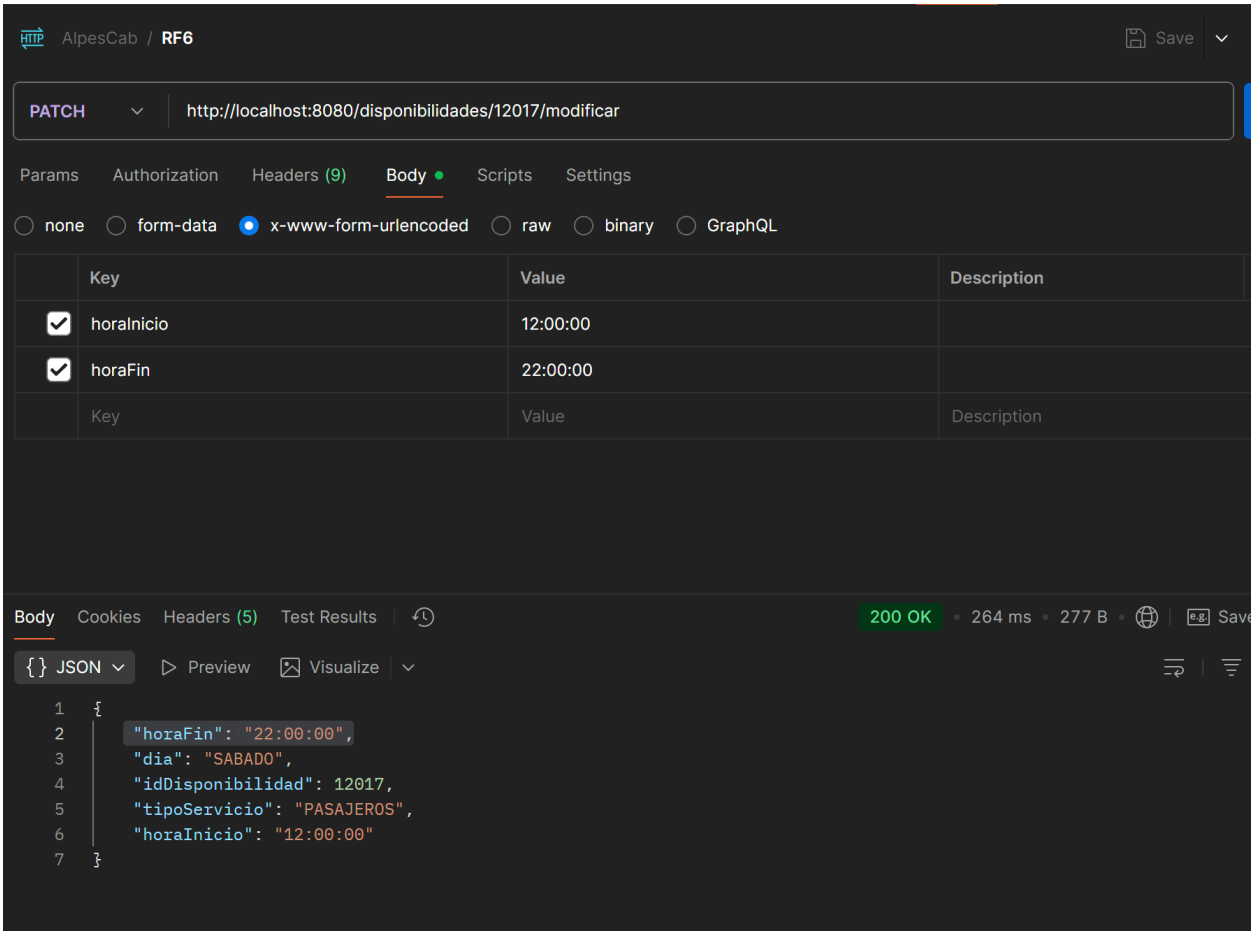
Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148




# RF6 Registrar que un conductor está disponible para un tipo de servicio

| Caso | Objetivo | Rol | Pasos | Resultado esperado |
|---|---|---|---|---|
| RF6 OK | Tipo PASAJEROS | COND | Marcar tipo_servicio PASAJEROS en disponibilidad | Disponibilidad guardada |
| RF6 FAIL | Tipo inválido | COND | tipo_servicio “AEREO” | Rechazo por check de dominio |

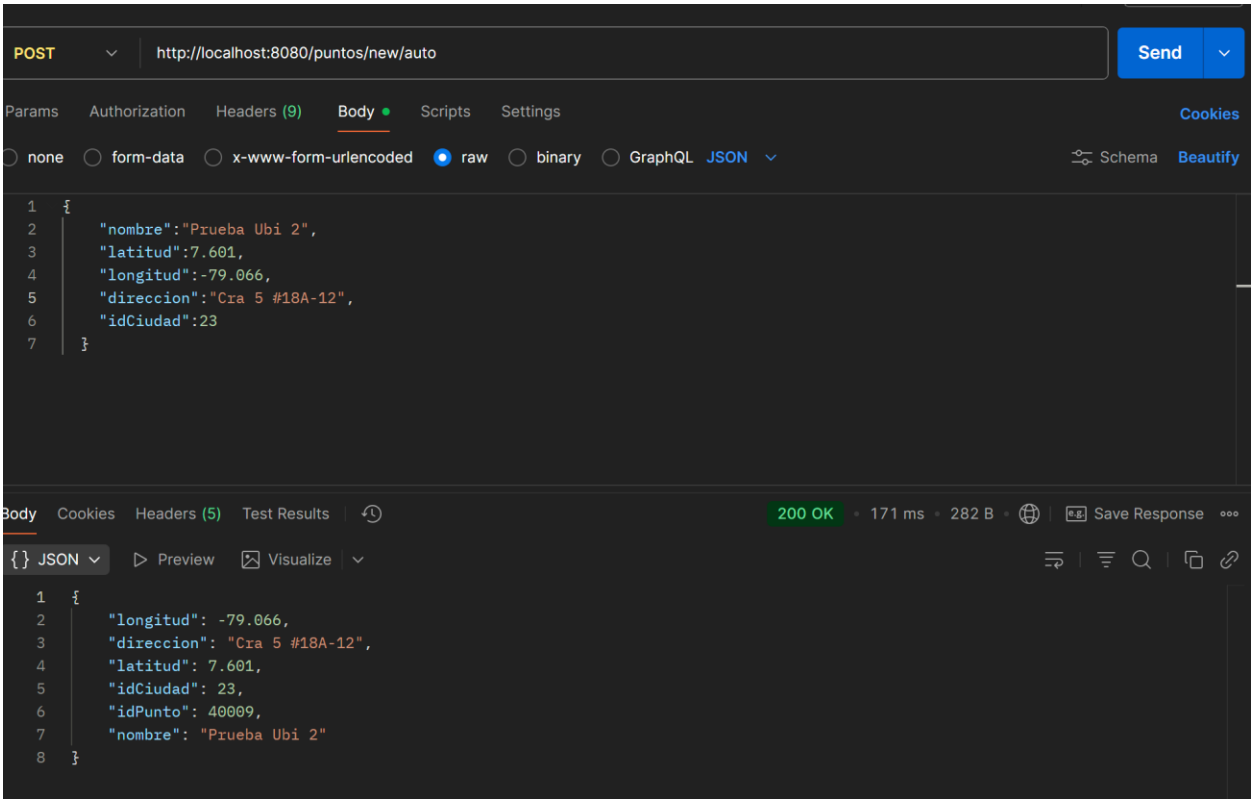
Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148




# RF7 Registrar puntos geográficos

| Caso | Objetivo | Rol | Pasos | Resultado esperado |
|---|---|---|---|---|
| RF7 OK | Crear punto en ciudad válida | USR | Latitud y longitud válidas. Ciudad 1 | Punto creado |
| RF7 FAIL | Ciudad inexistente | USR | Ciudad 99 | Rechazo por FK |

Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148

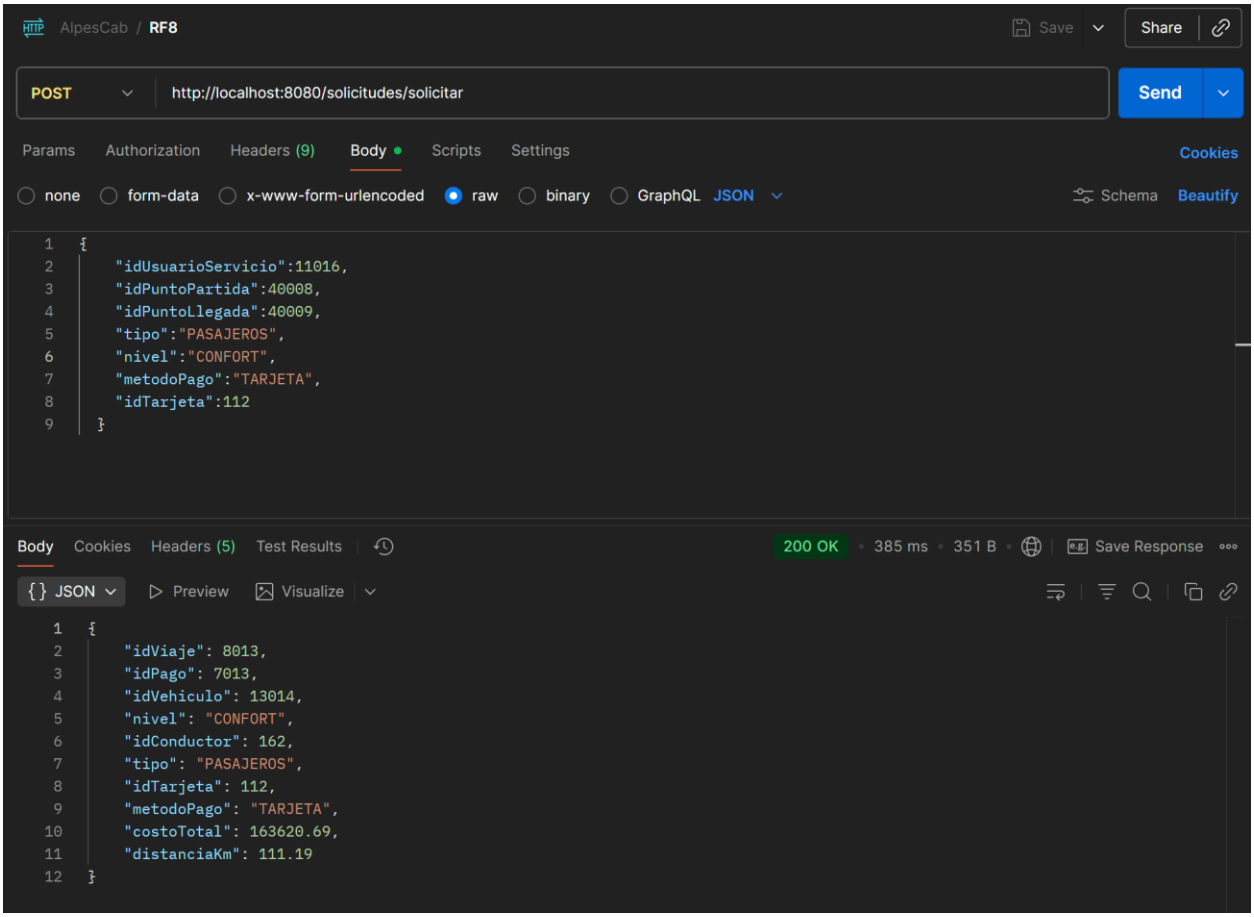



# RF8 Solicitar servicio y asignación

| Caso | Objetivo | Rol | Pasos | Resultado esperado |
|---|---|---|---|---|
| RF8 OK | Crear solicitud válida y asignar | USR | Tipo PASAJEROS nivel ESTANDAR, origen 4001. App asigna conductor cercano disponible 2001 con 3001 | Solicitud en estado ASIGNADA y Viaje creado con fecha_asignacion |
| RF8 FAIL | Sin disponibilidad | USR | Mismo origen y hora sin ninguna disponibilidad para tipo | Solicitud queda CREADA o RECHAZADA según lógica. Sin Viaje asignado |

1. Crear Solicitud de Servicio

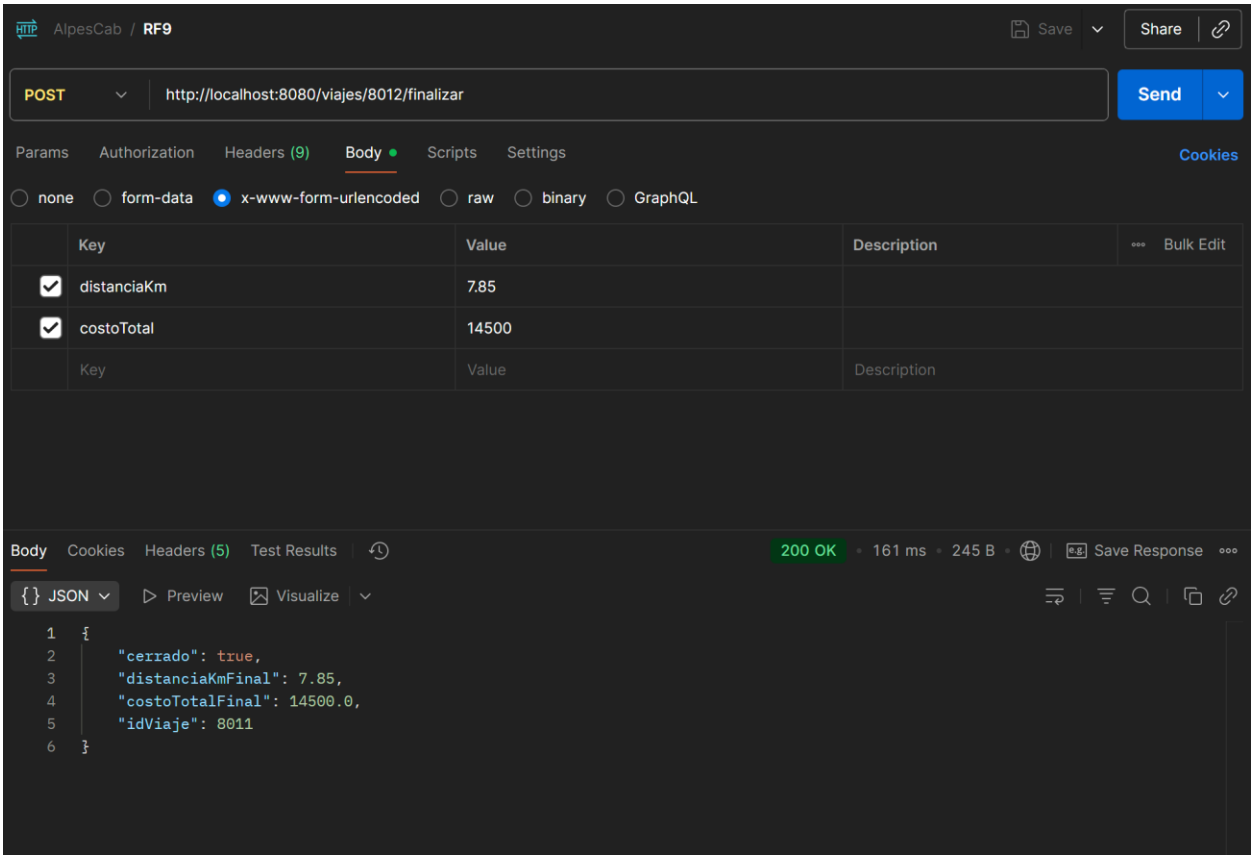
Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148




# RF9 Registrar viaje terminado

| Caso | Objetivo | Rol | Pasos | Resultado esperado |
|---|---|---|---|---|
| RF9 OK | Cerrar viaje con métricas | COND | Iniciar 14:05 fin 14:45 distancia 12.8 costo calculado | Viaje con hora_fin y costo_total persistidos |
| RF9 FAIL | Tiempos incoherentes | COND | Fin menor que inicio | Rechazo por check tiempos |

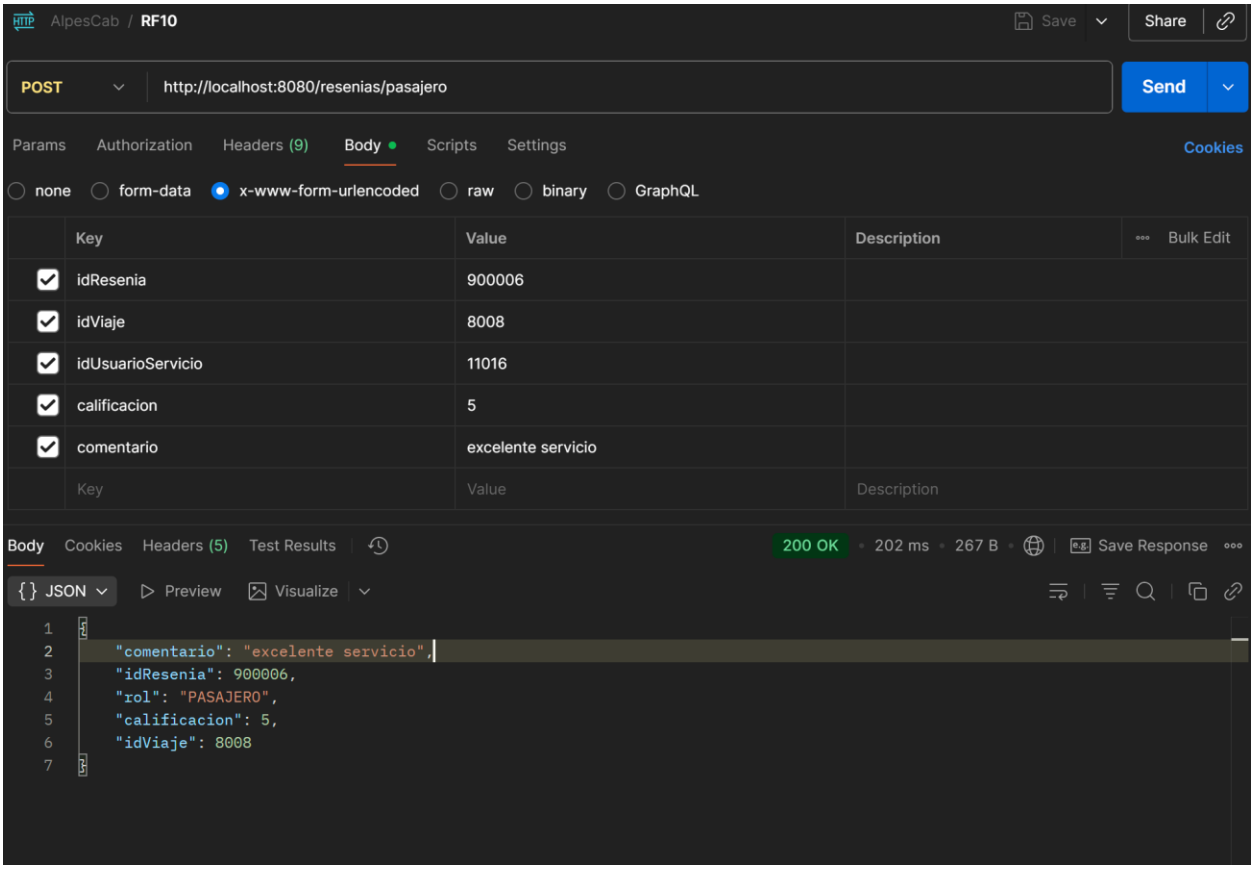
Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148




# RF10 Reseña del cliente al conductor

| Caso | Objetivo | Rol | Pasos | Resultado esperado |
|---|---|---|---|---|
| RF10 OK | Calificar viaje | USR | calificacion 5 comentario | Reseña creada ligada a Viaje |
| RF10 FAIL | Calificación inválida | USR | calificacion 6 | Rechazo por check calificación |

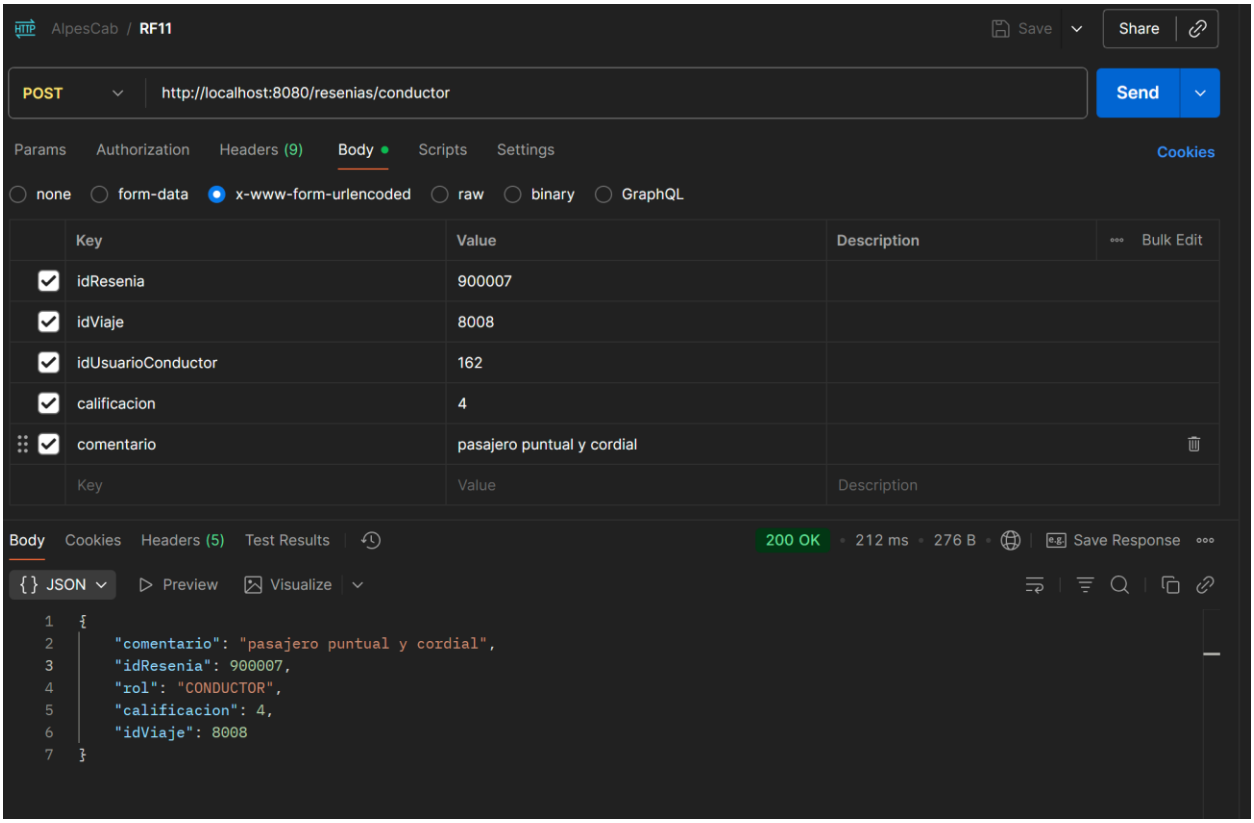
Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148




# RF11 Reseña del conductor al cliente

| Caso | Objetivo | Rol | Pasos | Resultado esperado |
|---|---|---|---|---|
| RF11 OK | Calificar viaje | COND | calificacion 4 comentario | Reseña creada ligada a Viaje |
| RF11 FAIL | Duplicado por política | COND | Intentar dos reseñas para el mismo viaje y autor | Rechazo por política de negocio si se aplica única por autor y viaje |

Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148




## 4.1.2 Pruebas de consultas RFC1

- ❖ Descripción de lo sucedido

  En ambos niveles de aislamiento, RFC1 no necesitó esperar a que RF8 terminara para continuar su ejecución.

- En SERIALIZABLE: RFC1 mantuvo una “foto” consistente desde que abrió la transacción; no bloqueó a RF8 ni esperó su commit.
- En READ_COMMITTED: RFC1 hizo su primera lectura, esperó 30 s y continuó normalmente; tampoco bloqueó ni esperó a RF8.

- ❖ Resultado presentado por RFC1

- Modo SERIALIZABLE: la segunda lectura de RFC1 mostró exactamente el mismo histórico que la primera (mismo número de filas). La orden creada por RF8 durante la espera no apareció en RFC1 porque la transacción conserva la vista consistente.

Mariana Cediel – 202321548
Alejandro Cruz – 201912149
Nicolás Hernández - 202322148

- Modo READ_COMMITTED: la segunda lectura de RFC1 sí incluyó la nueva orden generada por RF8 (la cantidad de filas aumentó y apareció el viaje/pago recién confirmados), siempre que RF8 hiciera commit antes de la segunda lectura.