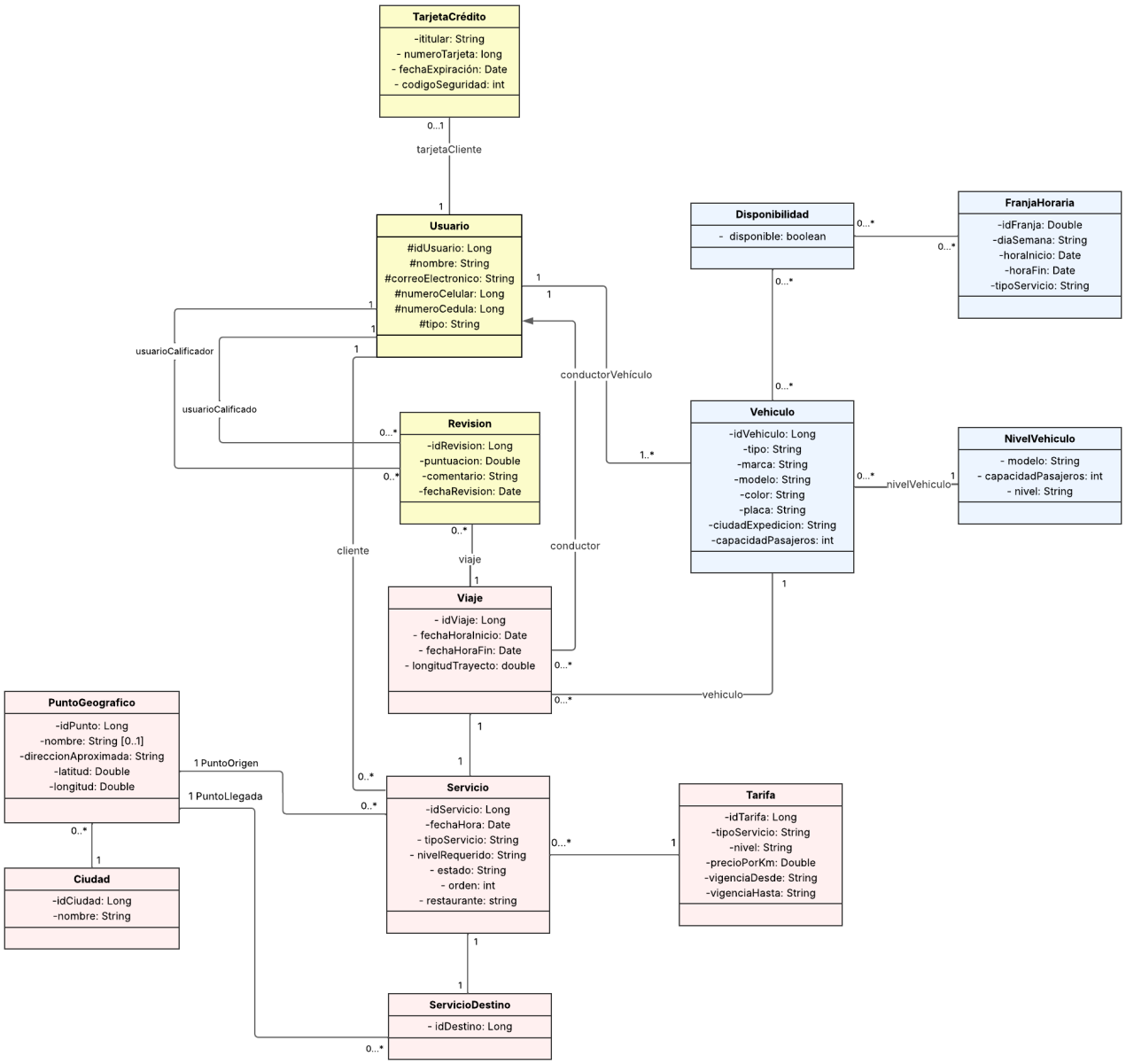
Entrega 2 AlpesCab

Sergio Arias | 202412370
Sara González | 202210908
Laura Martinez-Galindo | 202125643

---

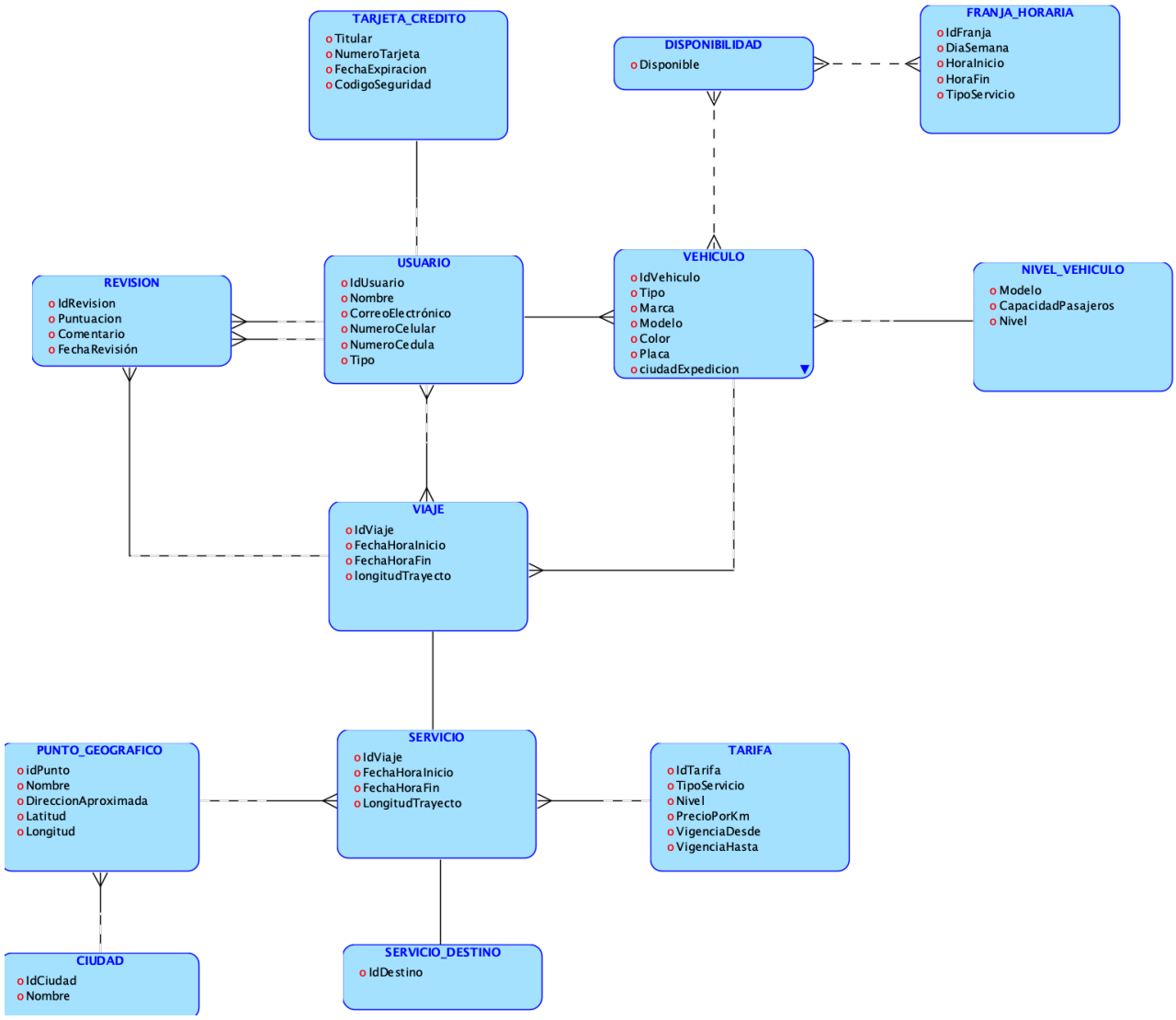
1. Análisis y revisión del modelo de datos

a. Modelo Conceptual UML



[Modelo en LucidChart para mejor visualización]

b. Modelo Conceptual E/R



c. Diseño de la base de datos y análisis de normalización

Después de haber analizado el diseño propuesto para la base de datos para la aplicación de AlpesCab, se lograron observar que según los niveles de normalización vistos en clase (hasta BCNF) se logró llegar a resultados positivos.

Se afirma que el modelo propuesto cumple con la Forma Normal de Boyce-Codd (BCNF), ya que se garantiza la eliminación de redundancias y anomalías de actualización. Cada entidad dentro del diagrama tiene una llave primaria bien definida, bien sea por el id o por una tupla de atributos, que determina de manera completa y única el resto de los atributos. Por ejemplo, en entidades como Vehículo, Usuario o FranjaHoraria, se observan los ID que se establecen por el sistema para facilitar la unicidad y el acceso a la información relevante; mientras que, en NivelVehiculo, por ejemplo, la llave primaria está compuesta por el modelo y la cantidad de pasajeros del automóvil y no hay otra llave candidata que comparta atributos con esta.

Asimismo, no se presentan dependencias parciales ni transitivas, y las relaciones de muchos a muchos se encuentran correctamente descompuestas en tablas intermedias, como en los casos de *Revisión*, *ServicioDestino* o *Disponibilidad*. Adicional a ello, los atributos que podrían generar jerarquías o multivariación, como los niveles de vehículo o las ciudades, se modelan como entidades independientes, evitando dependencias entre atributos no clave dentro de una misma tabla. Gracias a estas decisiones de diseño, existe integridad, consistencia y flexibilidad para futuras extensiones. También se realizaron simplificaciones y separaciones de tablas con el fin de que poder acceder y cumplir con los requerimientos funcionales junto con los de consulta, ejemplos de esto fueron las desapariciones de las tablas de usuarioServicio y usuarioConductor, quedando una única de Usuario.

Así luce el modelo relacional final, cuyas relaciones están en Forma Normal de Boyce-Codd (BCNF) como ya se justificó:

Usuario

idUsuario	nombre	numeroCelular	numeroCedula	correoElectronico	tipoUsuario
PK, SA	NN	NN	UA, NN, ND	UA, NN, ND	CK("Cliente", "Conductor")
1	Jorge	3004567890	1012456789	jaconductor@gamil.com	cliente
2	Tatiana	3109876543	1023456781	nierfan@gmail.com	conductor
3	Carlos	3206549871	1034567892	3sbstchr@gmail.com	cliente
4	Luka	3156781234	1045678912	LukaAnchor@gmail.com	cliente
5	Stinson	3155755237	1056789123	barney999@gmail.com	conductor
6	Robin	3112345678	1105912091	nightwing771@gmail.com	cliente

Review

idRevision	idUsuarioCalificador	idUsuarioCalificado	puntuacion	comentario	fecha	idViaje
PK,NN,SA,ND	FK(Usuario.idUsuario),NN,CK	FK(Usuario.idUsuario),NN,CK	NN, CK	SA, CK	NN	FK(Viaje.idViaje)
1	1	3	3,5	Va muy lento	8/31/2025	1
2	2	2	5	Tiene muy buena conversación	8/31/2025	2
3	3	2	4,8	Gran cliente	8/31/2025	3
4	2	1	3	Se salta los semaforos	8/31/2025	4
5	1	1	1	Iba viendo tiktok manejando	8/31/2025	5

ConductorVehiculo

idConductor	idVehiculo
PK, FK(Usuario.IdUsuario),SA	PK, FK(Vehiculo.IdVehiculo), NN, ND
4	1
4	2
5	3

TarjetaCredito

idTarjetaCredito	titularDeLaTarjeta	numeroTarjeta	fechaExpiracion	codigoSeguridad	clienteId
PK,NN,SA,ND,CK	CK, ND,UA, NN	NN, ND	NN	NN,CK	NN, FK(Usuario.idUsuario), CK(Usuario.tipo="Cliente")
1	Barney Stinson	123456789	27/11	245	1
2	MADRE DE TATIANA	987654321	27/01	111	2
3	Dick Grayson	123789456	32/02	222	3

Vehiculo

idVehiculo	tipo	marca	modelo	color	placa	ciudadExpedicion	capacidadPasajeros
NN,PK,SA,ND	NN	NN	NN	NN	NN,ND	NN	NN
1	Carro	Toyota	Colorado	Negro	ABC123	Bogotá	4
2	Motocicleta	Yamaha	Fast4X30	Rojo	DEF123	Cali	4
3	Camioneta	Nissan	Quest	Café	ABC456	Popayán	6
4	Carro	Dodge	Journey	Blanco	HIJ123	Chía	2
5	Motocicleta	Yamaha	Fast4X30	Negro	DEF456	Envigado	3

NivelVehiculo

modelo	capacidadPasajeros	nivel
PK, NN	PK, NN, CK(>0)	CK(in "Estandar", "Confort", "Large")
Colorado	4	Estandar
Fast4X30	4	Confort
Quest	6	Large
Journey	2	Confort
Fast4X30	3	Estandar

FranjaHoraria

idFranja	diaSemana	horalnicio	horaFin	tipoServicio
PK,SA,ND	NN	NN	NN	NN, CK(in "Transporte Pasajeros", "Domicilio Comida", "Transporte Paquete")
1	Lunes	9:30	11:30	Transporte Pasajeros
2	Lunes	14:00	19:00	Domicilio Comida
3	Jueves	6:00	13:00	Transporte Paquete
4	Viernes	20:00	24:00:00	Transporte Pasajeros
5	Domingo	7:00	14:00	Domicilio Comida

Disponibilidad

idVehiculo	idFranja	disponible
PK,FK(Vehiculo.idVehiculo)	PK, FK(FranjaHoraria.idFranja)	NN
1	1	true
1	2	true
1	3	false
4	1	true
5	5	true

PuntoGeografico

idPunto	nombre	latitud	longitud	direccionAproximada	idCiudad
PK,SA	NULL	NN	NN	NN	NN, FK(Ciudad.idCiudad)
1	Universidad de los Andes	4,61104	-74,070288	Carrera 7 con Calle 24	1
2	NULL	4,591053	-74,072167	ntre Carreras Séptima y Octava con Calles Décima	1
3	Parque Arví	4,65	-74,1	En la localidad de Teusaquillo, atravesado por las carrera	1
4	NULL	4,53604	-74,1388	Carrera 5 #12-41	1
5	NULL	0	0	Centro	5

Ciudad

idCiudad	nombre
PK	NN, ND
1	Bogotá
2	Cali
3	Medellín
4	Barranquilla
5	Santa Marta

Servicio

idServicio	idCliente	fechaHora	tipoServicio	nivelRequerido	estado	orden	restaurante	idPuntoPartida
PK, SA	FK(UsuarioServicios.idUsuario), NN	NN	NN, CK(in "Transporte Pasajeros", "Domicilio Comida", "Transporte Paquete")	CK(in "Estandar", "Confort", "Large"), NULL	NN, CK(in "Pendiente", "Asignado", "Cancelado")			FK(PuntoGeografico.idPunto), NN
1	2	29/09/2025 9:05	Transporte Pasajeros	Estandar	Pendiente	NULL	NULL	1
2	5	29/09/2025 9:05	Domicilio Comida	Confort	Asignado	34	El Corral	2
3	6	29/09/2025 9:05	Transporte Paquete	Large	Cancelado	NULL	NULL	3
4	2	29/09/2025 9:05	Transporte Pasajeros	Confort	Pendiente	NULL	NULL	4
5	5	29/09/2025 9:05	Domicilio Comida	Estandar	Pendiente	12	Crepes & Waffles	5

ServicioDestino

idDestino	idServicio	idPuntoLlegada
PK, SA	FK(Servicio.idServicio), NN	FK(PuntoGeografico.idPunto), NN
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5

Tarifa

idTarifa	tipoServicio	nivel	precioPorKm	vigenciaDesde	vigenciaHasta
PK, NN, SA, ND	NN, CK(in "Transporte Pasajeros", "Domicilio Comida", "Transporte Paquete")	CK(in "Estandar", "Confort", "Large")	NN, CK(precioPorKm > 0)	NN, CK(vigenciaDesde<vigenciaHasta)	NN, CK(vigenciaHasta>vigenciaDesde)
1	Transporte Pasajeros	Estandar	2000	1/01/2025	12/31/2025
2	Domicilio Comida	Confort	2200	1/01/2025	12/31/2025
3	Transporte Paquete	Large	2500	1/01/2025	12/31/2025
4	Transporte Pasajeros	Confort	2800	1/01/2025	12/31/2025
5	Domicilio Comida	Estandar	1800	1/01/2025	12/31/2025

Viaje (este es el historico)

idViaje	fechaHoralnicio	fechaHoraFin	longitudTrayecto	idServicio	idConductor	idVehiculo
PK, NN, SA, ND	NN	NN	NN	FK(Servicio.IdServicio), NN	FK(UsuarioConductor.idUsuario), NN	FK(Vehiculo.IdVehiculo), NN
1	8:15:00	8:45:00	12.5	1	1	1
2	9:00:00	9:35:00	8.2	2	3	2
3	10:10:00	10:45:00	5.6	3	4	3
4	11:20:00	11:55:00	9.0	4	3	3
5	12:30:00	13:10:00	15.0	5	4	4

De igual forma se adjunta en los documentos del repositorio.

2. Implementación del esquema de la base de datos

En la ruta src\db\nuevasTablas.sql, se encuentra el script para generar las tablas del modelo. De igual forma, se generaron secuencias para los ids de algunas tablas, véase esto en src\db\crearSecuencias.sql.

3. Diseño de sentencias SQL

Los requerimientos de consulta se realizaron mediante scripts de sql encontrados en src\db\.

RFC1:

Tablas usadas: Usuarios, Servicio, Tarifa y viajes

Atributos usados: Usuarios: idUsuario

Servicios: idServicio, nivelRequerido

Tarifa: tipoServicio, nivel, vigenciaDesde, vigenciaHata

Vehiculo: fechaHoraInicio

JOIN: Se usó el INNER JOIN para en primera instancia unir el servicio junto con su usuario asignado, de viaje al servicio y tarifa a su tipo de servicio. Como los datos relevantes no contienen ningun valor nulo y solo usan datos en común usar este join era bastante apropiado.

```
---RFC 1
SELECT s.idServicio, v.fechaHoraInicio AS fecha, ROUND(v.longitudTrayecto, 2) AS longitudTrayecto,
ROUND((v.longitudTrayecto * t.precioPorKm), 2) AS costoTotal, s.tipoServicio, v.idVehiculo FROM usuarios du
INNER JOIN servicios s ON s.idCliente = du.idUsuario
INNER JOIN viajes v ON s.idServicio = v.idServicio
INNER JOIN tarifas t ON s.tipoServicio = t.tipoServicio AND t.nivel = s.nivelRequerido AND v.fechaHoraInicio
BETWEEN t.vigenciaDesde AND t.vigenciaHasta
WHERE du.idUsuario = 239;
```

RFC2:

Tablas usadas: Usuarios y Vehículos

Atributos: Usuario: idUsuario, nombre

Viajes: idConductor

JOIN: Se uso INNER JOIN para unir Usuarios con vehículo, como siempre hay elementos en común, es apropiado usar este tipo de JOIN.

```
---RFC 2
SELECT u.idUsuario AS idConductor, u.nombre, COUNT(v.idViaje) AS total_viajes FROM viajes v
INNER JOIN usuarios u ON v.idConductor = u.idUsuario
GROUP BY u.idUsuario, u.nombre
ORDER BY total_viajes DESC
FETCH FIRST 20 ROWS ONLY;
```

RFC3:

Tablas usadas: Vehículo, Servicio y Tarifas

Atributos: Viajes: idVehiculo, idConductor, fechaHoraInicio, longitudTrayecto

Tarifa: precioPorKM, tipoServicio

Servicios: tipoServicio, idServicio, tipoServicio

JOIN: Siguiendo la misma lógica, al solo necesitar unir elementos en común, se usó INNER JOIN pues lo que se une como lo son los servicios con vehículos y servicios tarifa y servicios, no existe la posibilidad de que se aparezcan nulos.

```
---RFC 3
SELECT v.idVehiculo, s.tipoServicio, v.idConductor AS idUsuario, ROUND(SUM(v.longitudTrayecto * t.precioPorKm)) AS costo_total,
ROUND(SUM(v.longitudTrayecto * t.precioPorKm) * (1 - 0.2)) AS ganancia FROM viajes v
INNER JOIN servicios s ON v.idServicio = s.idServicio
INNER JOIN tarifas t ON s.tipoServicio = t.tipoServicio AND t.nivel = s.nivelRequerido
AND v.fechaHoraInicio BETWEEN t.vigenciaDesde AND t.vigenciaHasta
WHERE v.idConductor = 1
GROUP BY v.idVehiculo, s.tipoServicio, v.idConductor
ORDER BY v.idVehiculo, s.tipoServicio;
```

RFC4:

Tablas usadas: Ciudad, Servicio y PuntoUbicacion

Atributos: Ciudad: nombre, idCiudad

Servicios: tipoServicios, nivelRequerido, id puntoPartida

PuntoPartida: idPunto, idCiudad

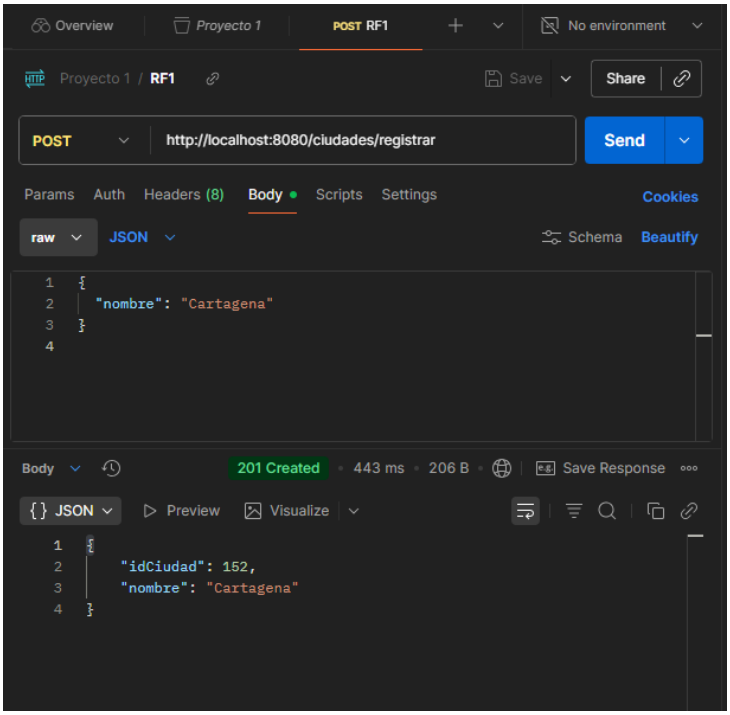
JOIN: Siguiendo la misma lógica, al solo necesitar unir elementos en común, se usó INNER JOIN.

```
---RFC 4
SELECT c.nombre AS ciudad, s.tipoServicio, s.nivelRequerido, COUNT(*) AS total_servicios,
 ROUND(100 * COUNT(*) / SUM(COUNT(*)) OVER (), 2) AS porcentaje FROM servicios s
INNER JOIN puntos_geograficos p ON s.idPuntoPartida = p.idPunto
INNER JOIN ciudades c ON p.idCiudad = c.idCiudad
WHERE c.idCiudad = 10 AND s.fechaHora BETWEEN TO_TIMESTAMP('2024-07-04 23:54:14.000','YYYY-MM-DD HH24:MI:SS.FF3')
AND TO_TIMESTAMP('2025-09-12 03:41:28.000','YYYY-MM-DD HH24:MI:SS.FF3')
GROUP BY c.nombre, s.tipoServicio, s.nivelRequerido
ORDER BY porcentaje DESC;
```

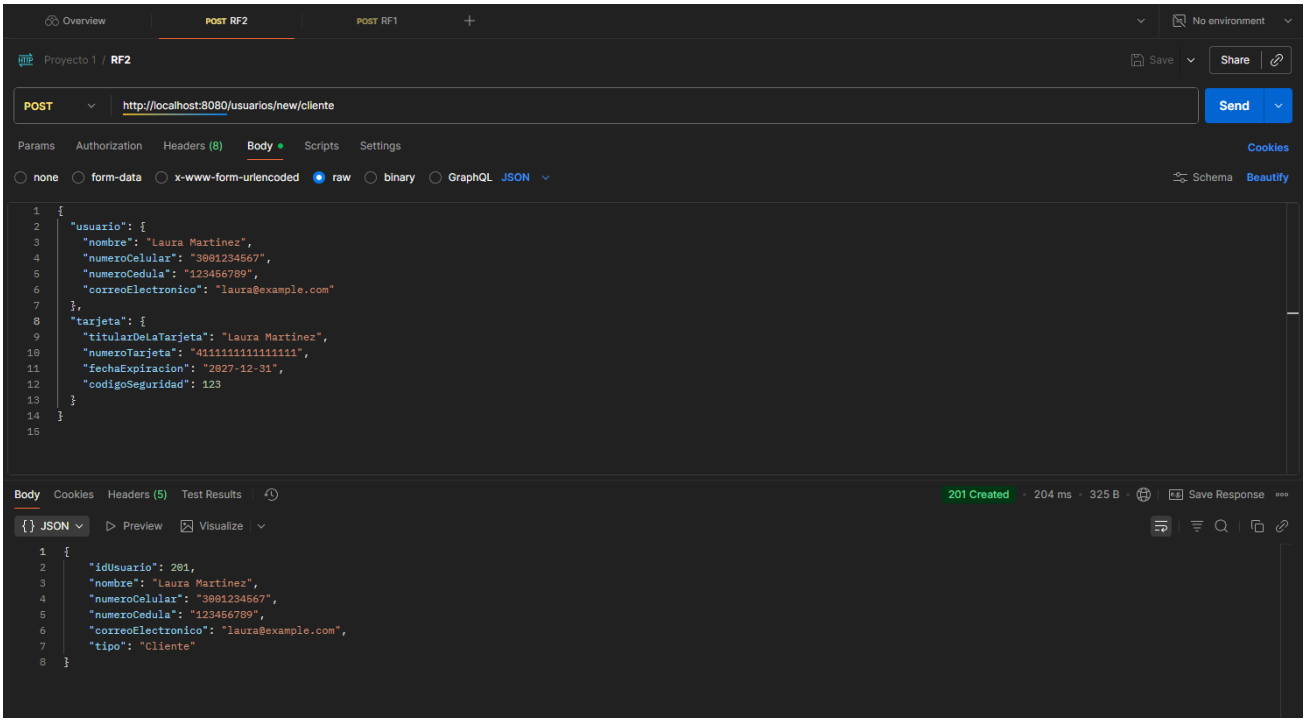
4. Desarrollo de la aplicación

Siguiendo el framework Spring, se realizaron las entidades, repositorios y controladores de cada una de las tablas o relaciones de nuestro modelo, todo se encuentra en el repositorio. Adicionalmente, se implementó la lógica para los requerimientos funcionales y se hicieron pruebas con postman para garantizar que todo estuviera bien. A continuación, se presentan estas solicitudes de postman:

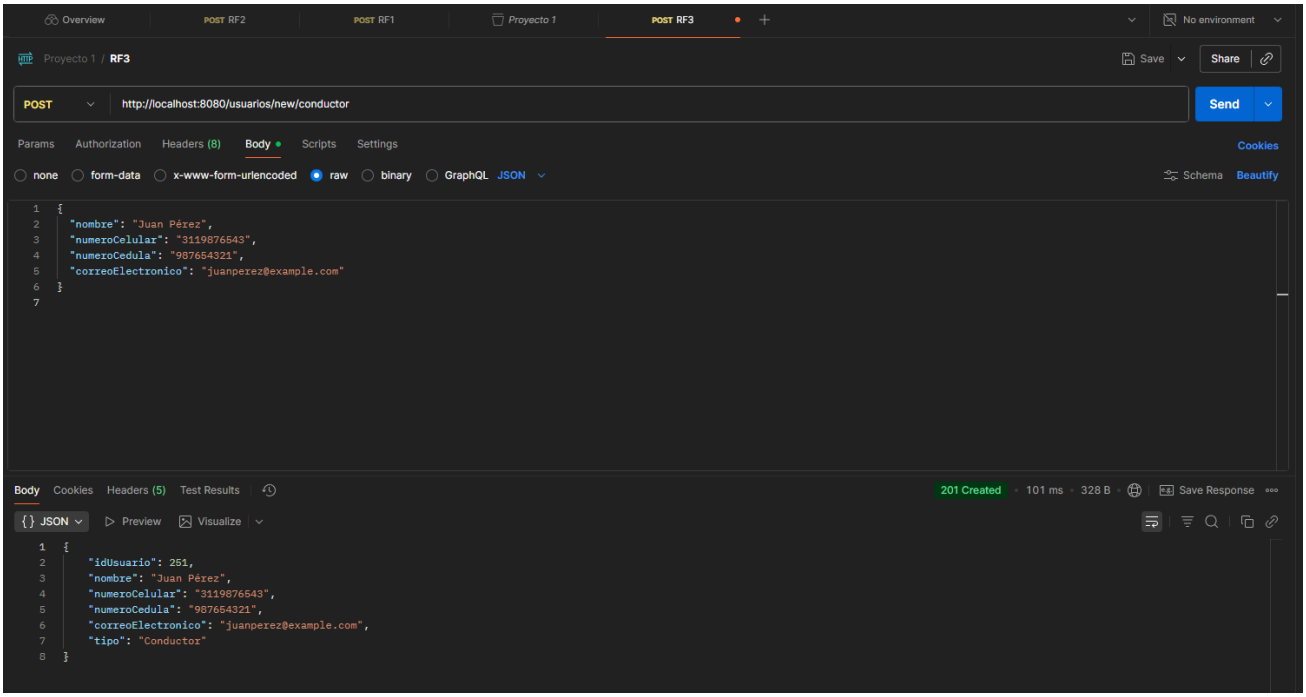
RF1: Registrar una ciudad



RF2: Registrar un usuario de servicios

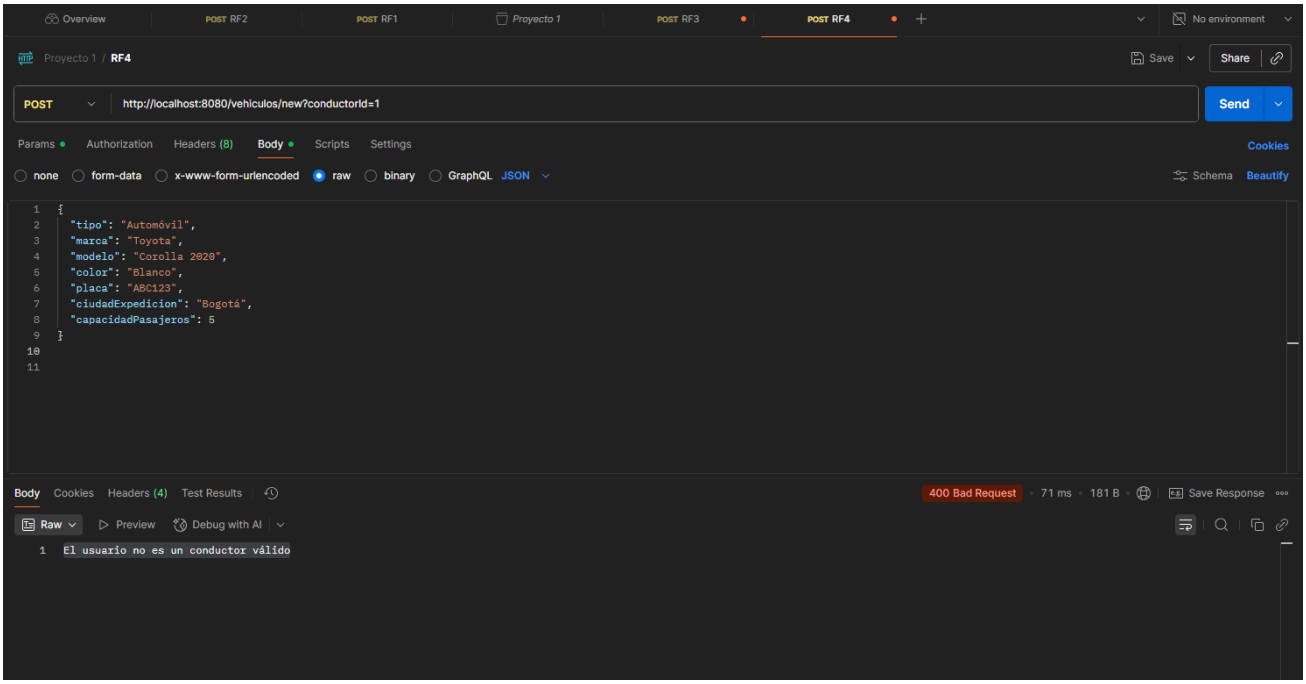


RF3: Registrar un usuario conductor

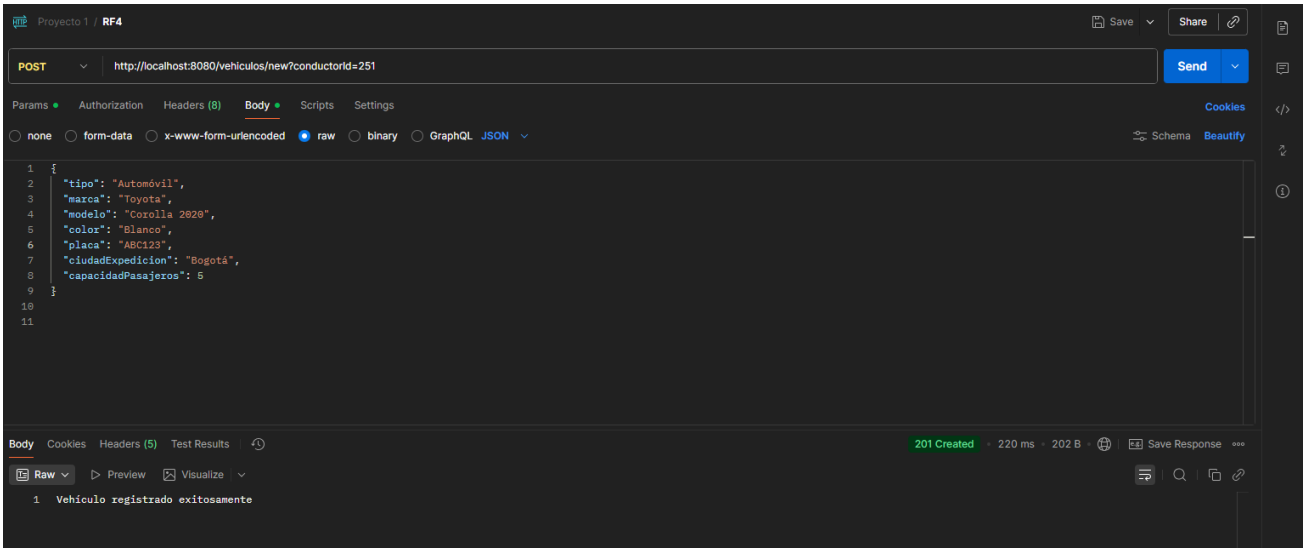


RF4: Registrar un vehículo para un usuario conductor

Cuando el usuario no es un conductor:

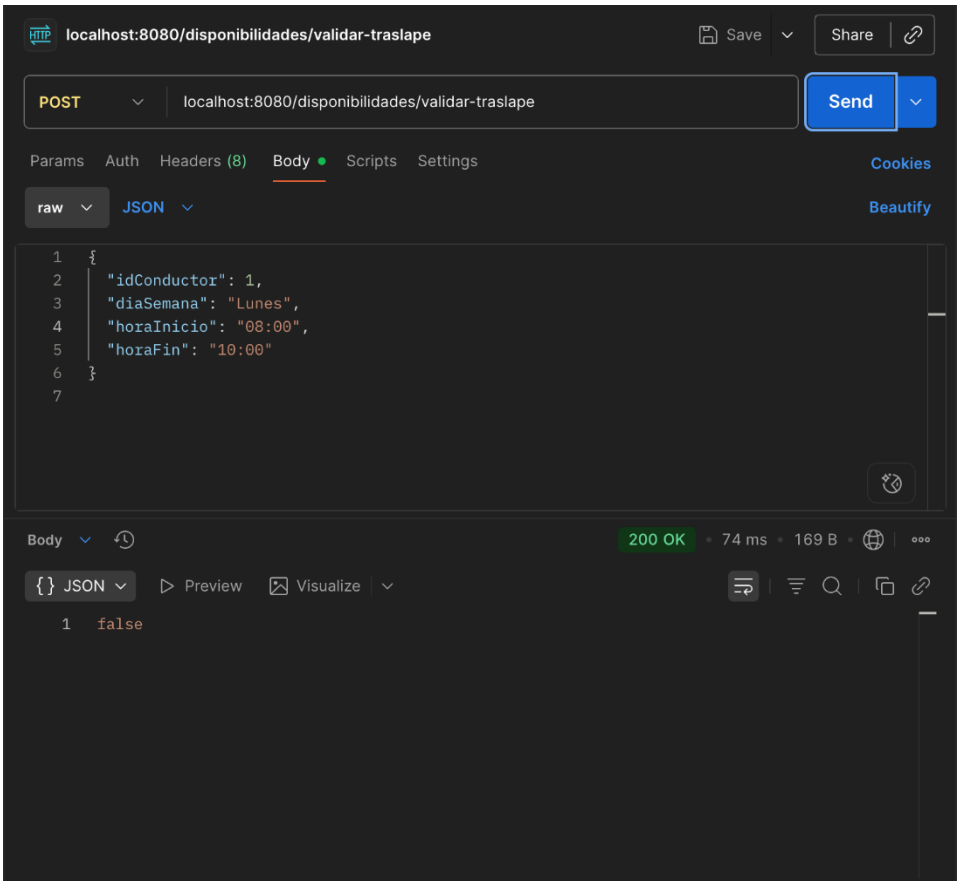


Cuando se registra un vehículo para un usuario que si es conductor:

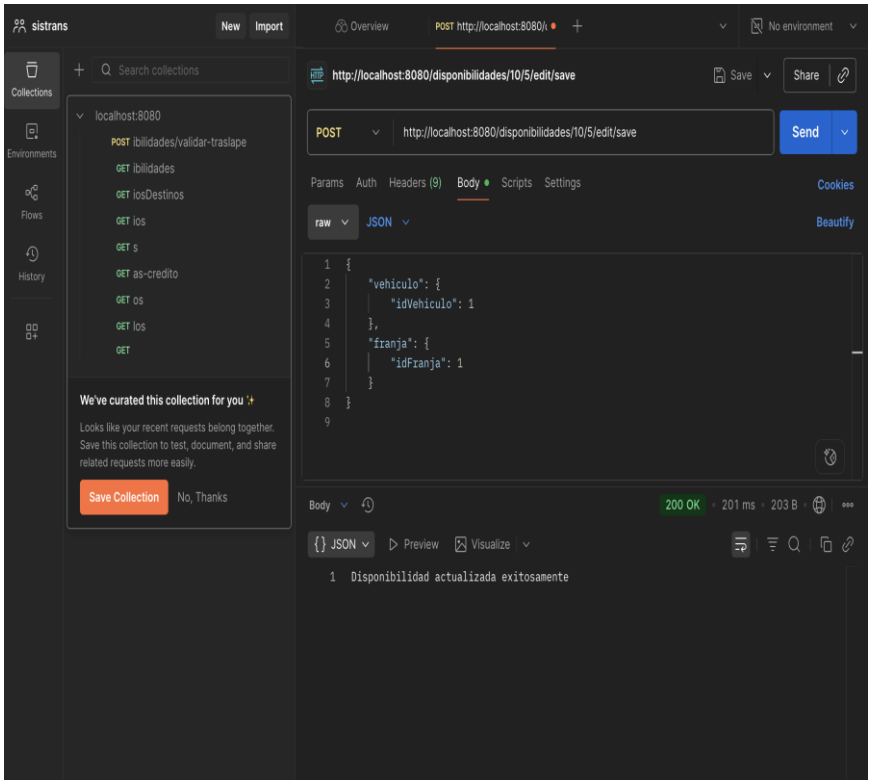


RF5: Registrar la disponibilidad de un vehículo para servicios

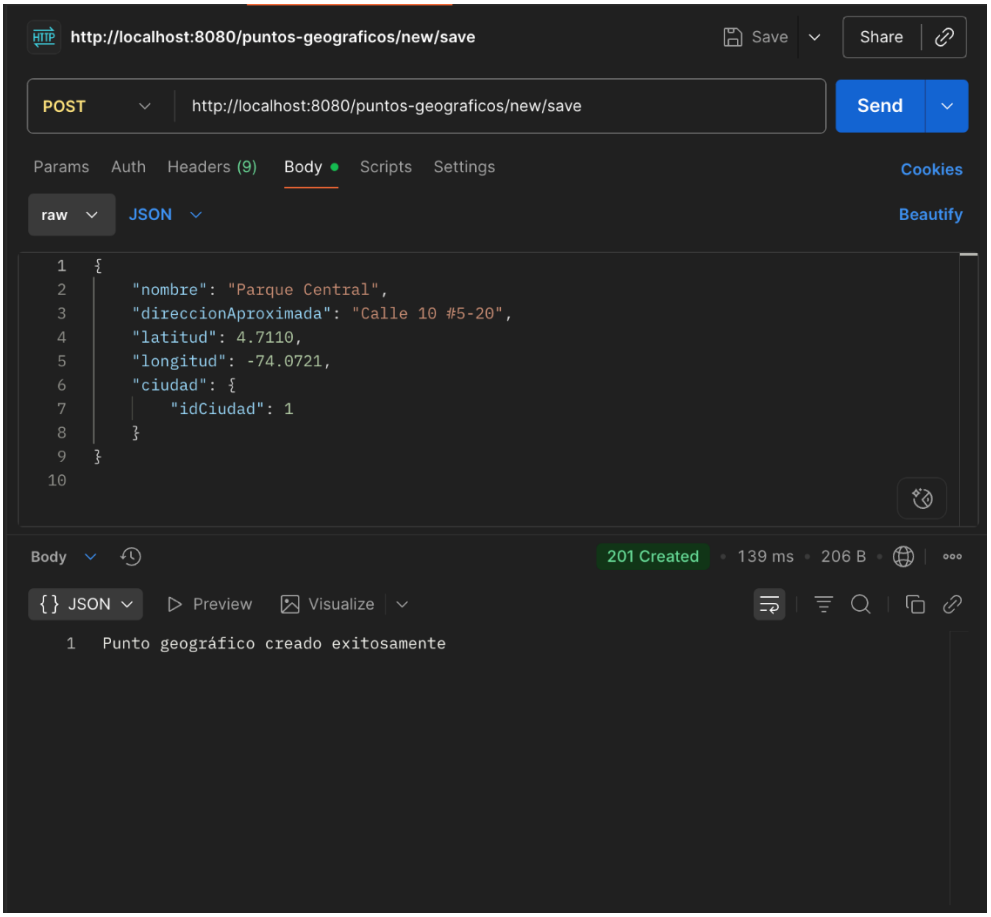
Validación función auxiliar



RF6- Modificar La Disponibilidad De Un Vehículo Para Servicios

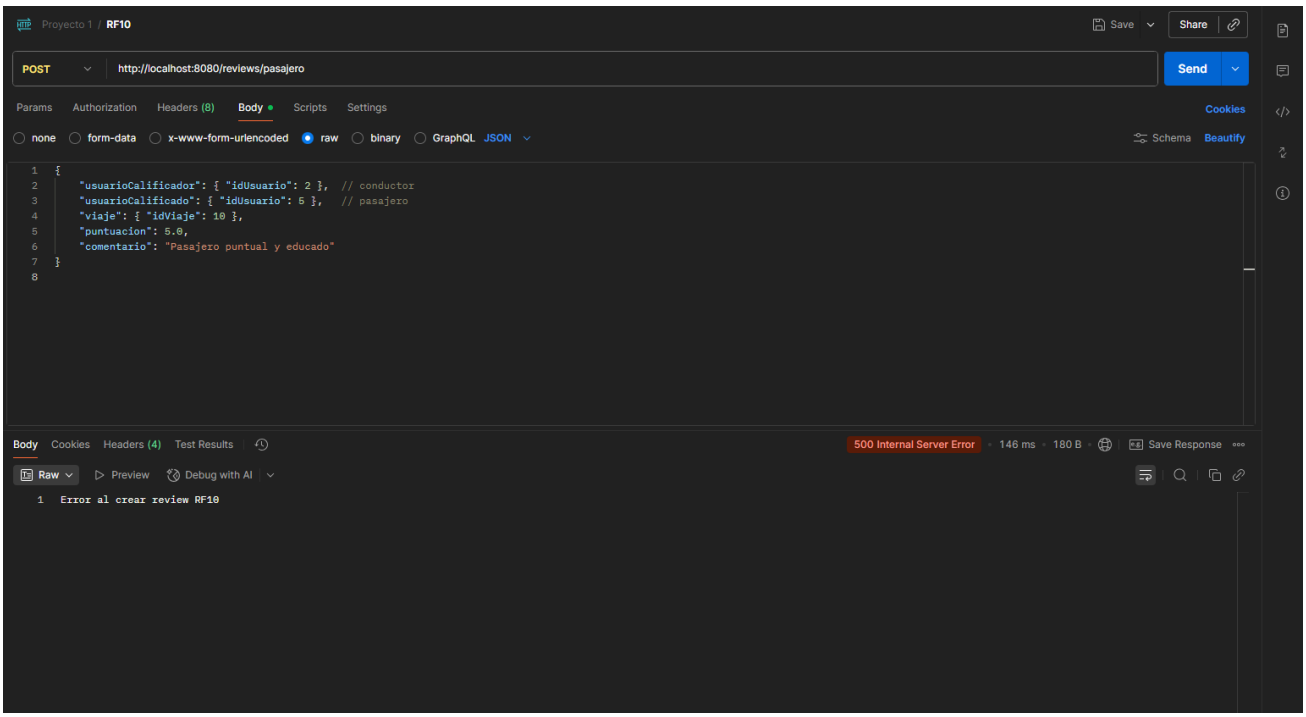


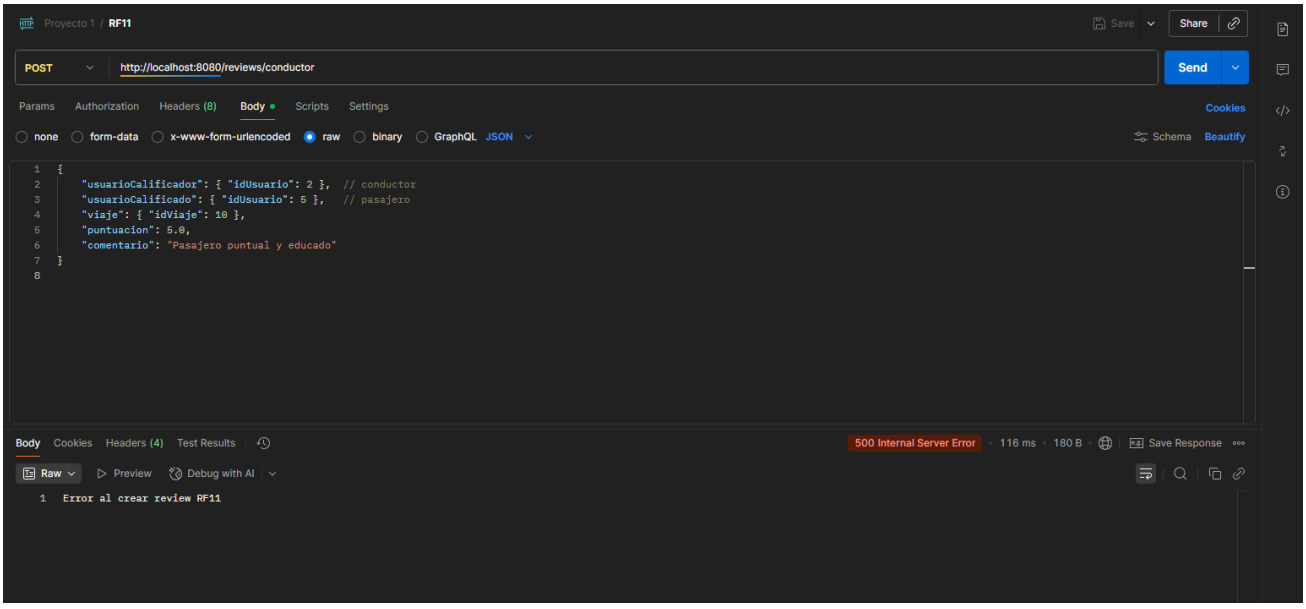
RF7: Registrar un punto geográfico



RF8 - SOLICITAR UN SERVICIO POR PARTE DE UN USUARIO DE SERVICIOS

RF10 y RF11 se añadieron en el controller de review, pero no se logró que funcionaran.





5. Población de tablas

Se poblaron las tablas mediante un script sql, el cual se encuentra en “src\db\poblarTablas.sql”. La base de datos con las relaciones, secuencias y los datos ya insertados se encuentra en el usuario ISIS2304A01202520 con contraseña kuuuULHdgiAZ.

6. Escenarios de prueba

No logramos implementarlos o verlos de manera practica con Postman, pero estos fueron los que definimos desde la entrega pasada:

6.1. Pruebas de unicidad de PK:

- Si intento crear un usuario de servicio con un id ya existente, por la restricción ND no va a ser posible crearlo.

Ej: Crear Usuario Servicio con ID U00003--> Error, ya existe un Usuario Conductor ese ID C00003.

- Si intento crear un usuario conductor con un id ya existente, por la restricción ND no va a ser posible crearlo.

Ej: Crear Usuario Conductor con ID C00001--> Error, ya existe un Usuario Conductor ese ID C00001.

- Si intento crear una review de servicio con un id ya existente, por la restricción ND no va a ser posible crearlo.

Ej: Crear review con ID 202408310001 --> Error, ya existe una review con ese ID 202408310001.

- Si intento crear una tarjeta de crédito con un id ya existente, por la restricción ND no va a ser posible crearlo.

Ej: Crear Tarjeta de crédito con ID T00001--> Error, ya existe un Usuario Conductor ese ID C0001.

- Si intento crear una franja horaria de actividad con un id ya existente, por la restricción ND no va a ser posible crearla.

Ej: Crear una franja horaria de actividad con ID F001 --> Error, ya existe una franja horaria de actividad de ID F001

- Si intento crear un vehículo con un id ya existente, por la restricción ND no va a ser posible crearla.

Ej: Crear un vehículo con ID V00001--> Error, ya existe un vehículo con el ID V00001

- Si intento crear una ciudad con un id ya existente, por la restricción ND no va a ser posible crearla.

Ej: Crear una ciudad con ID C001 --> Error, ya existe una ciudad de ID C001

- Al insertar, por ejemplo, Servicio(idServicio = S001, ..), me lo va a permitir. Pero si luego intento insertar otro servicio con el mismo id, es decir, Servicio(idServicio = S001, ..), va a fallar porque la PK ya existe (eso prueba la unicidad de tuplas en la tabla de servicios)

- Al insertar, por ejemplo, Tarifa(idTarifa = T001, tipoServicio = ‘Pasajero’, nivel = ‘Estándar’, precioPorKm = 2500, ...), funcionará correctamente. Sin embargo, al intentar insertar otra tupla con idTarifa = T001 → Falla por PK duplicada (las tuplas en la tabla tarifa son únicas)
- Al insertar ServicioPunto(idServicio = S001, idPunto = P001), no habrá problema. Pero al intentar insertar otra tupla ServicioPunto(idServicio = S001, idPunto = P001) → Falla, PK compuesta duplicada (las tuplas en la tabla ServicioPunto son únicas)

6.2. Pruebas de integridad de FK

- Si intento ingresar en review una foreign key proveniente de alguna tabla diferente a usuario servicio o conductor, por la restricción CK que va a revisar que pertenezca a la tabla de usuarios (Bool que verifique que este en la tabla de DatosUsuario ya que ahí están todos los usuarios).

Ej: Crear Review de ID 202408310005 con un usuario de idUsuario T0002 --> Error, el usuario ingresado no es válido.

- Si intento ingresar en Tarjeta de crédito una foreign key proveniente de alguna tabla diferente a usuario servicio o conductor, por la restricción CK que va a revisar que pertenezca a la tabla de usuarios.

Ej: Crear una Tarjeta de crédito de ID T00004 con un usuario de IdUsario de 202408310001 --> Error el usuario ingresado no es válido.

- Un caso válido sería si por ejemplo idTarifa=T001 existe en Tarifa, entonces, no tendría problema en insertar un servicio con dicho id de tarifa en la tabla servicios. Pero si, por el contrario, idTarifa = T999 no existe en Tarifa, fallará la integridad referencial. Algo similar pasa con idVehiculo, idConductor, idCliente, idPuntoOrigen en la tabla servicios.

6.3. Pruebas de integridad de las restricciones de chequeo

- Crear una tarjeta de crédito con un formato de fecha de vencimiento invalido.

Ej: Crear Tarjeta de crédito de ID T00004 con fecha de vencimiento de

1111 --> Error, la fecha debe estar en el formato MM/AA

- Crear una tarjeta de crédito con un codigo de seguridad invalido.

Ej: Crear Tarjeta de crédito de ID T00004 con código de seguridad

"prueba" --> Error, el código de seguridad debe ser un entero positivo de 3 dígitos.

- Crear una tarjeta de crédito con un titular invalido.

Ej: Crear Tarjeta de crédito de ID T00004 con titutar de “” --> Error, el titular de la tarjeta debe ser una cadena de caracteres de tamaño mayor o igual que 2.

- Crear una review con una calificación invalida.

Ej: Crear Review de ID R202408310001 con calificación de -1 --> Error, la calificación debe ser un entero entre 1 y 5.

- Crear una review con un comentario invalido.

Ej: Crear Review de ID R202408310001 con un comentario que dice “” --> Error, el comentario debe ser una cadena de caracteres de tamaño mayor o igual que 5.

- Crear un usuario con un número de teléfono invalido.

Ej: Crear un usuario de ID U00004 con número de teléfono 31550 --> Error, el número de teléfono debe ser un entero de 10 dígitos.

- En el caso de tarifa, si intento insertar precioPorKm = -1000, fallará porque CK me garantiza que precioPorKm > 0. Además, si intento añadir una tarifa para la cual vigenciaDesde > vigenciaHasta, la inserción también fallará porque vigenciaDesde debe ser estrictamente menor a vigenciaHasta.

6.4. Reglas de negocio

- Validar que un conductor no esté asignado a dos servicios simultáneamente. Por lo que insertar un Servicio con conductor en estado “ocupado”, debe fallar.
- Validar que se registre horaInicio y horaFin en viajes completados (RF9).
- No pueden existir períodos sin tarifa definida. Por ende, si se inserta servicio en fecha sin tarifa, la inserción debería fallar.
- Una combinación de tipoServicio + nivel debe ser única en un periodo de vigencia. Porlo que intentar insertar tarifa duplicada para mismo rango debería fallar.

- Cada servicio debe tener al menos un punto de origen y un punto de destino. Si se intenta insertar servicio sin puntos de origen y destino, la inserción debería fallar.
- No se permiten puntos duplicados en el mismo servicio. Por ejemplo, si punto origen = punto destino, la inserción del servicio falla.

6.5. Otros escenarios de prueba para RF y RFC

- RF1: Insertar Ciudad con idCiudad = C001 y luego intentar duplicar, debería fallar si se cumple unicidad.
- RF2 y RF3: Registrar usuario cliente y conductor, validar que los campos sean no nulos y la unicidad de los datos únicos como cedula o correo.
- RF4: Registrar vehículo y asociar a conductor, probar FK hacia UsuarioConductor.
- RF5 / RF6: Insertar franja horaria que se solapa con otra franja horaria para un mismo conductor, debería fallar así el tipo de vehículo sea distinto.
- RF7: Insertar punto con coordenadas válidas, luego duplicar dirección exacta. Debería fallar si se definió restricción de unicidad.
- RF8: Solicitar servicio, verificar cálculo automático de tarifa usando tabla Tarifa. Validar tanto para caso exitoso como para uno inválido (sin tarifa vigente).
- RF9: Cerrar viaje con horaInicio y horaFin, lo cual debería ser válido. Pero si se intentar cerrar sin horaFin, debería fallar.
- RF10/RF11: Insertar revisión vinculada a un servicio ya finalizado, lo cual debería ser válido. Mientras que, si se inserta revisión sobre un servicio no finalizado, falla.
- RFC1: Consultar todos los servicios de un usuario, debería devolver una lista.
- RFC2: Query que ordena conductores por cantidad de servicios → probar con dataset de prueba.
- RFC3: Calcular total ganado por vehículo, probar sumatorias con y sin comisión, para ver que en efecto se esta repartiendo bien entre el conductor y AlpesCab.
- RFC4: Consultar servicios en un rango de fechas, probar con fechas sin servicios para ver si la lista sale vacía y también con varios para ver si en efecto se llena.