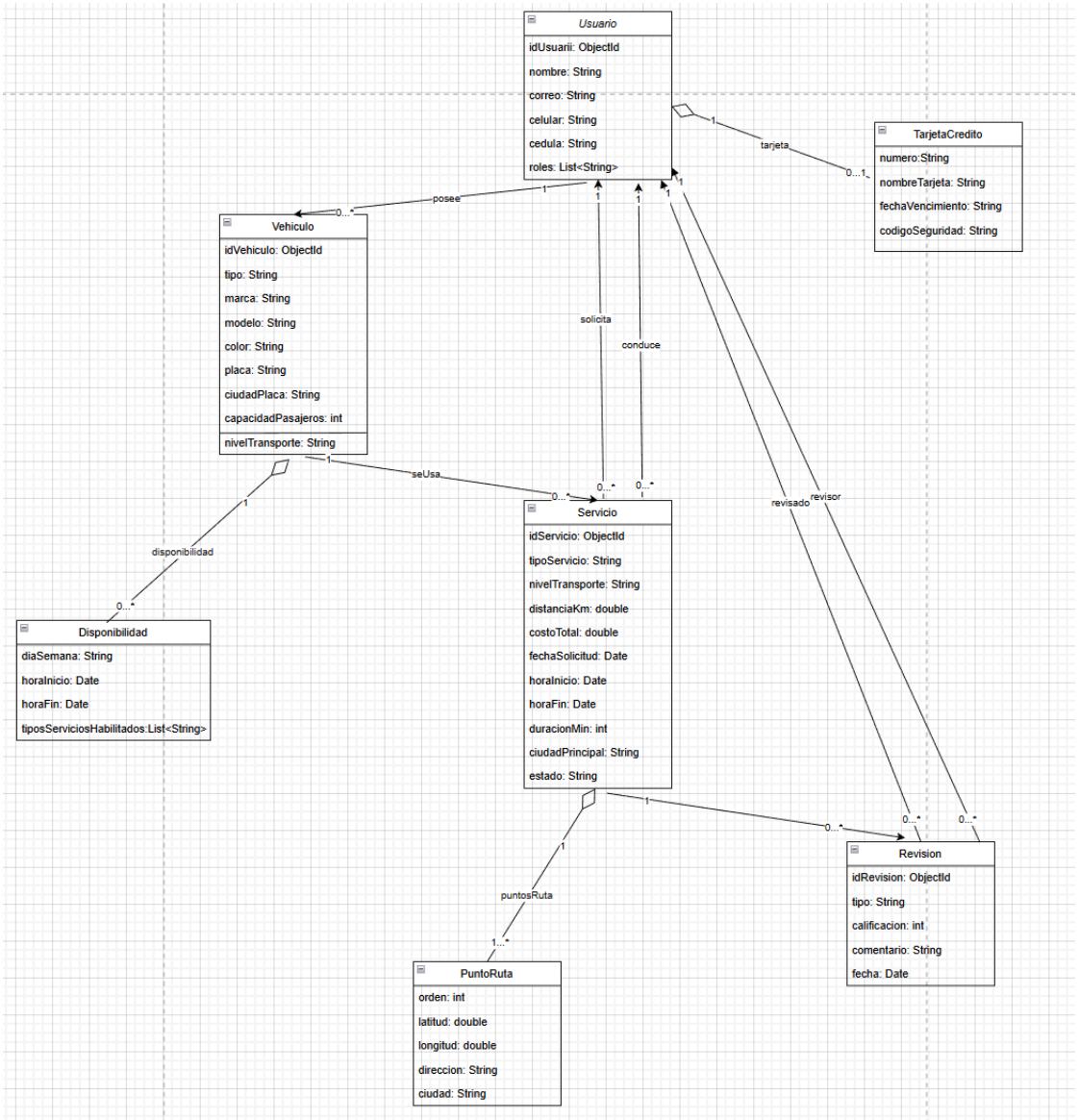
Juan Manuel Rojas-202321306
Mariana Castillo-202320169
Julian Restrepo-202320177

Entrega Final

Repositorio: https://github.com/jmrojasp1/ALPESCAB_MONGO.git

1. **(30%) Diseño de la base de datos para todos los requerimientos funcionales: Describan las colecciones de datos y las relaciones entre ellas**

a. **(5%) Proponga un modelo conceptual en UML o E/R que describa las entidades del modelo de datos para la aplicación que se quiere desarrollar.**



b. **(5%) La lista de entidades con la descripción de cada una de ellas, las relaciones entre entidades y su cardinalidad (uno a uno, uno a muchos, o muchos a muchos).**

1. Clase: Usuario

Atributos:

- idUsuario : ObjectId
- nombre : String
- correo : String
- celular : String
- cedula : String
- roles : List<String> // {SERVICIO, CONDUCTOR}

Asociaciones:

- vehiculos : List<Vehiculo> (0..*)
- revisionesRealizadas : List<Revision> (0..*)
- revisionesRecibidas : List<Revision> (0..*)
- tarjetaCredito : TarjetaCredito (0..1)

---

2. Clase: TarjetaCredito

Atributos:

- numero : String
- nombreTarjeta : String
- fechaVencimiento : String
- codigoSeguridad : String

Asociaciones:

- (Composición) pertenece a **Usuario**

---

3. Clase: Vehículo

Atributos:

- idVehiculo : ObjectId
- tipo : String // CARRO, CAMIONETA, MOTO
- marca : String
- modelo : String
- color : String
- placa : String
- ciudadPlaca : String
- capacidadPasajeros : int
- nivelTransporte : String // ESTANDAR, CONFORT, LARGE

Asociaciones:

- conductor : Usuario (1)
- disponibilidades : List<Disponibilidad> (0..*)
- servicios : List<Servicio> (0..*)

---

4. Clase: Disponibilidad

Atributos:

- diaSemana : String
- horaInicio : Date
- horaFin : Date

- tiposServicioHabilitados : List<String> // PASAJEROS, COMIDA, MERCANCIAS

Asociaciones:

- (Composición) pertenece a **Vehiculo**

---

5. Clase: Servicio

Atributos:

- idServicio : ObjectId
- tipoServicio : String // PASAJEROS, COMIDA, MERCANCIAS
- nivelTransporte : String // solo para pasajeros
- distanciaKm : double
- costoTotal : double
- fechaSolicitud : Date
- horaInicio : Date
- horaFin : Date
- duracionMin : int
- ciudadPrincipal : String
- estado : String // PENDIENTE, EN_CURSO, FINALIZADO

Asociaciones:

- usuarioServicio : Usuario (1)
- conductor : Usuario (1)
- vehiculo : Vehiculo (1)
- puntosRuta : List<PuntoRuta> (1..*)

---

6. Clase: PuntoRuta

Atributos:

- orden : int
- latitud : double
- longitud : double
- direccion : String
- ciudad : String

Asociaciones:

- (Composición) pertenece a **Servicio**

---

7. Clase: Revision

Atributos:

- idRevision : ObjectId
- tipo : String // CONDUCTOR→USUARIO o USUARIO→CONDUCTOR
- calificacion : int // 0–5
- comentario : String
- fecha : Date

Asociaciones:

- servicio : Servicio (1)
- revisor : Usuario (1)
- revisado : Usuario (1)

RELACIONES

- Usuario "1" -- "0..*" Vehiculo
- Vehiculo "1" o-- "0..*" Disponibilidad
- Servicio "1" o-- "1..*" PuntoRuta
- Usuario "1" -- "0..*" Servicio : solicita
- Usuario "1" -- "0..*" Servicio : conduce
- Vehiculo "1" -- "0..*" Servicio
- Servicio "1" -- "0..*" Revision
- Usuario "1" -- "0..*" Revision : revisor
- Usuario "1" -- "0..*" Revision : revisado
- Usuario "1" o-- "0..1" TarjetaCredito

a. **(10%) El análisis de selección de esquema de asociación (modelo normalizado o embebido) para cada relación entre entidades, justificado a partir de los requerimientos de la aplicación y un análisis de la carga de trabajo descritas antes**.

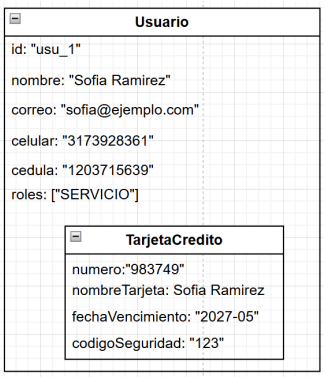
Relación	**Esquema**	**Donde queda**	**Justificación**
Usuario-TarjetaCredito (1-0…1)	Embebido	Usuario.tarjetaCredito	La tarjeta se muestra y se edita desde el usuario. Baja cardinalidad.
Usuario (conductor)-Vehículo (1-0...*)	Referenciado	Vehiculo.conductorId	Se filtra por conductor y por placa. Evita duplicar datos y usa indices
Vehiculo-Disponibilidad (1-0…*)	Embebido	Vehiculo.disponibilidades[]	Se consulta junto al vehículo para matching. Lista pequeña
Servicio-PuntoRuta (1-1…*)	Embebido	Servicio.puntosRuta[]	Puntos pertenecen al viaje. Siempre se leen con el servicio
Usuario (solicitante)-Servicio (1-0…*)	Referenciado	Servicio.usuarioId	Un usuario tiene muchos servicios. Evita copiar el perfil y el índice es por usuario

Usuario(conductor)- servicio (1-0…*)	Referenciado	Servicio.conductorId	Listados por conductor/fecha. No se duplica y el índice es por conductor
Vehiculo- servicio (1-0…*)	Referenciado	Servicio.vehiculoId	Muchos servicios por vehículo. Reportes por vehículo y el índice es por vehículo
Servicio- revision (1-0…*)	Referenciado	Revision.servicioId	Muchas revisiones, se listan por servicio usuario
Usuario (revisor)- revision (1-0…*)	Referenciado	Revision.revisorId	Listar lo que cada uno revisó, el índice es por revisor
Usuario (revidado)- revision (1-0…*)	Referenciado	Revision.revisadoId	Promedios por usuario. Indice por revisado

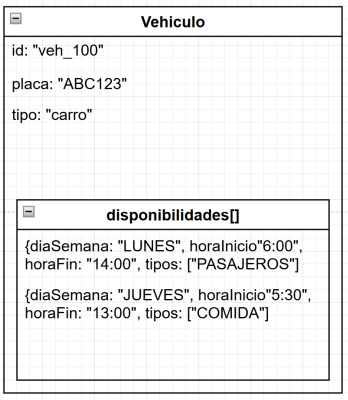
b. 10%) Una descripción gráfica usando Json de cada relación entre entidades en donde presente un ejemplo de datos junto con el esquema de asociación usado (referenciado o embebido). En los anexos se muestra un ejemplo de lo que se requiere.

JSON entre entidades

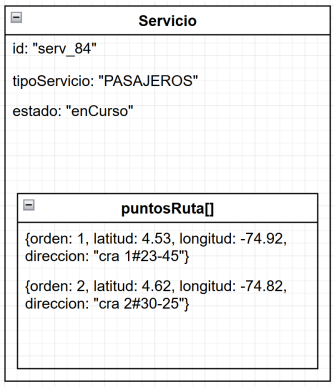
Usuario-TarjetaCredito Embebido



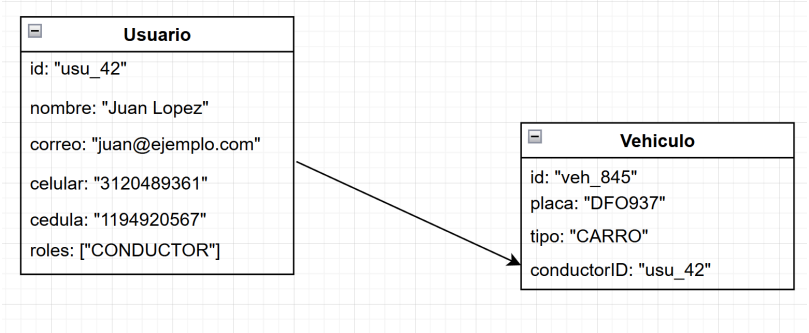
Vehiculo-Disponibilidad (array) Embebido



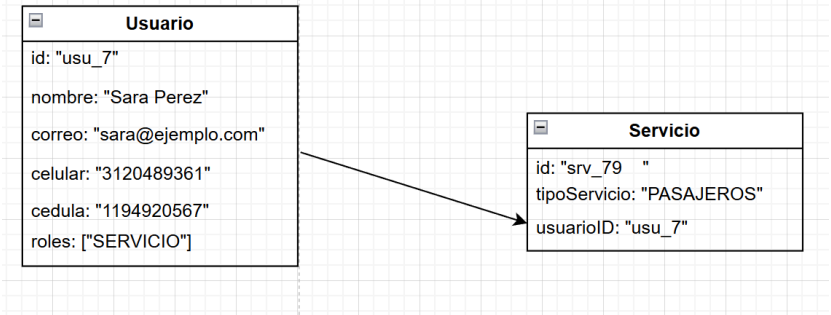
Servicio-puntosRuta Embebido



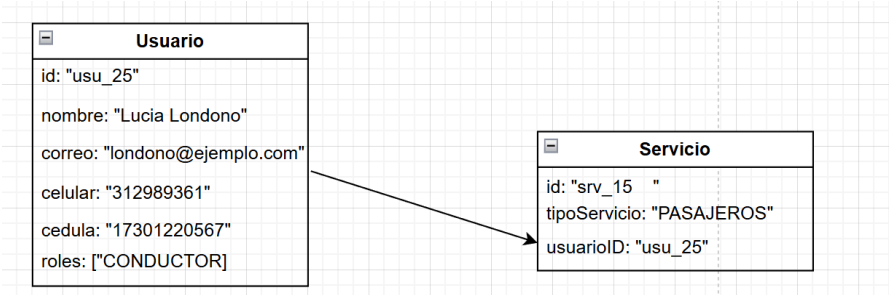
Usuario(CONDUCTOR)-Vehiculo Referenciado



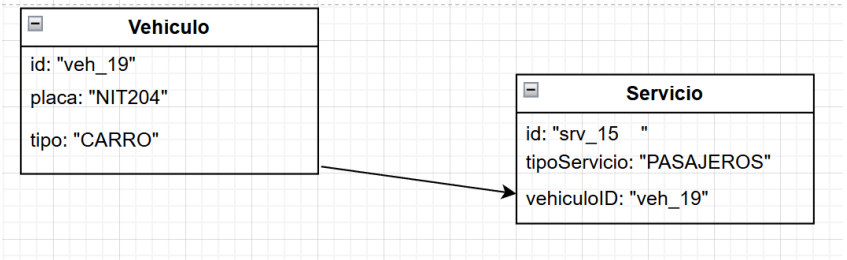
Servicio-Usuario(SOLICITANTE) Referenciado



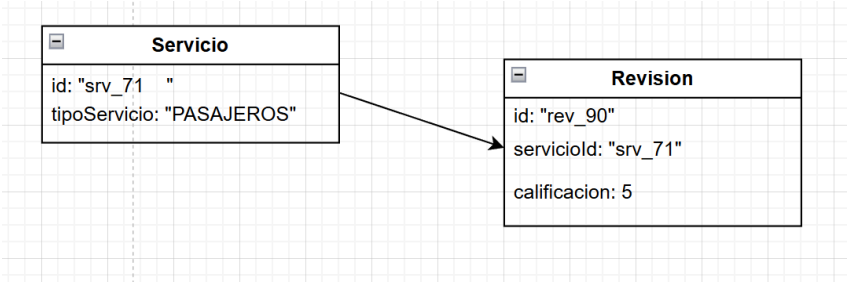
Servicio-Usuario(CONDUCTOR) Referenciado



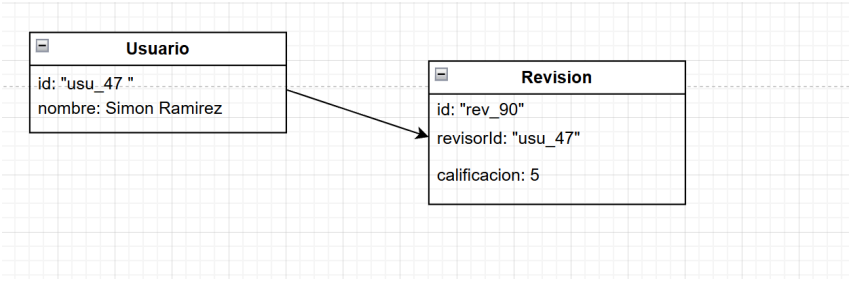
Servicio-Vehículo Referenciado



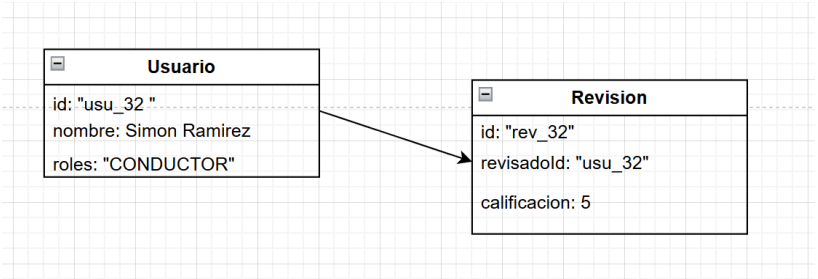
Revisión-Servicio Referenciado



Revision-Usuario (REVISOR) Referenciado



Revision-Usuario(REVISADO)



2.(10%) Implementación de la base de datos: Cree en MongoDB las colecciones principales de su base según lo descrito antes. Puede usar Compass o Mongo Shell (Mongosh) para realizar este proceso. Anexe a los entregables los archivos con los scripts utilizados. Cree los esquemas de validación para cada colección. Puede usar Compass o Mongo Shell (Mongosh) para realizar este proceso. Guarde lo hecho en un archivo. Anexe a los entregables los archivos con los scripts utilizados.

Para la implementación de la base de datos, creamos todas las colecciones en MongoDB siguiendo el modelo que definimos previamente en el UML. Cada colección se configuró con sus campos, tipos de dato y validaciones básicas para garantizar que la información se registrara de forma correcta. Esto permitió dejar lista la estructura necesaria para soportar los usuarios, vehículos, servicios y revisiones que usa la aplicación.

El código utilizado para crear estas colecciones quedó guardado en la carpeta **docs** del repositorio, donde se encuentra el archivo con todo el script de creación. Ese archivo contiene exactamente la definición aplicada en MongoDB y corresponde al documento que adjuntamos como parte de la entrega.

3. (7%) Diseño de escenarios de prueba: Para cada requerimiento funcional y requerimiento funcional de consulta, plantee un escenario de prueba para validar su funcionamiento. Adjunte la documentación de cada escenario.

RF1- Registrar un usuario de servicios
Precondiciones: correo no registrado
Entrada: POST/usuarios
{"nombre": "Ana Lopez", "correo":"ana@ejemplo.com", "celular": "3001234567", "cedula": "1234567890", "roles" :["SERVICIO"]}
Resultado esperado: HTTP 201 con id, documento en 'usuarios' rol SERVICIOS
Evidencia: db.usuarios.findOne({ correo:"ana@ejemplo.com" }, { _id:1, roles:1 })

RF2-Registrar un usuario conductor
Precondiciones: correo no registrado, sin vehiculos
Entrada: POST /usuarios
{"nombre": "Luis Giraldo", "correo": "luis@ejemplo.com", "celular":"3007654321", "cedula": "9876543210", "roles":["CONDUCTOR"]}
Resultado esperado: HTTP 201 documento con rol CONDUCTOR
Evidencia: db.usuarios.findOne({ correo:"luis@ejemplo.com" })

RF3-Registrar un vehículo para un conductor
Precondiciones: Conductor creado (RF2), placa no registrada
Entrada: POST /vehiculos
{"conductorId": "1","tipo":"CARRO ", "marca":"Toyota", "modelo": "Corolla 2020", "color": "Rojo", "placa":"ABC123", "ciudadPlaca":"Bogotá", "capacidadPasajeros":4 , "nivelTransporte": "ESTANDAR"}
Resultado esperado: HTTP 201 conductorId asignado, placa unica
Evidencia: db.vehiculos.findOne({ placa:"ABC123" });
db.vehiculos.countDocuments({ placa:"ABC123" }) === 1

RF4-Revisar disponibilidad de de un conductor y su vehículo
Precondiciones: Vehículo creado
Entrada: POST /vehiculos/{id}/disponibilidades
[{"diaSemana":"LUNES", "horaInicio": "06:00", "horaFin": "14:00","tiposServicioHabilitados" : ["PASAJEROS","COMIDA"]}]
Resultado esperado: HTTP 201 franjas agregadas
Evidencia: db.vehiculos.findOne({ _id:ObjectId("") }, { disponibilidades:1 })

RF5-Modificar disponibilidad
Precondiciones: Existe LUNES 6:00-14:00
Entrada: PUT /vehiculos/{id}/disponibilidades

{"diaSemana": "LUNES ", "horaInicio": "10:00", "horaFin": "16:00", "tiposServicioHabilitados": ["PASAJEROS"]}
Resultado esperado: HTTP 400/409
Evidencia: db.vehiculos.findOne({ _id:ObjectId("") }).disponibilidades

RF6-Solicitar un servicio (asignacion automatica)
Precondiciones: Conductor y vehiculo disponibles en la franja
Entrada: POST /servicios
{"tipoServicio":"PASAJEROS", "nivelTransporte": "CONFORT", "usuarioId": "", "puntosRuta" :[{"orden ":1,"latitud":4.65,"longitud":- 74.06},{"orden":2,"latitud":4.68,"longitud":-74.05}]}
Resultado esperado: HTTP 201 asigna conductorId/vehiculoId; costo calculado; conductor queda OCUPADO.
Evidencia: db.servicios.findOne({ _id:ObjectId("") });
db.usuarios.findOne({ _id:ObjectId("") }, { estado:1 })

RF7-Finalizar viaje e histórico
Precondiciones: Servicio en curso (RF6)
Entrada: POST /servicios/{id}/finalizar
{"horaFin":"2025-11-05T08:45:00Z"}
Resultado esperado: HTTP 20o estado FINALIZADO, calcula duración/distancia/costo, conductor DISPONIBLE
Evidencia: db.servicios.findOne({ _id:ObjectId("") }, { estado:1, costoTotal:1 })
db.usuarios.findOne({ _id:ObjectId("") }, { estado:1 })

RFC1-Historico de servicios por usuario
Precondiciones: Usuario con al menos 2 servicios finalizados
Entrada: GET /usuarios/{id}/servicios?estado=FINALIZADO
Resultado esperado: HTTP 200; lista descendiente por fecha con costo, puntos, conductor, vehículo
Evidencia: db.servicios.find({ usuarioId:"" }).sort({ fechaSolicitud:-1 })

RFC2-Top 20 conductores por número de servicios
Precondiciones: Base de datos con >20 conductores y servicios finalizados
Entrada: GET /reportes/top-conductores?limit=20
Resultado esperado: HTTP 200; 20 ítems {conductor, totalServicios} en orden descendente.
Evidencia: db.servicios.aggregate([{ $match:{estado:"FINALIZADO"} }, { $group:{_id:"$conductorId", total:{ $sum:1 }}}, { $sort:{ total:-1 }}, { $limit:20 }])

RFC3- Utilización por ciudad y rango de fechas
Precondiciones: Servicios en varias fechas y tipos
Entrada: GET /reportes/utilizacion?ciudad=Bogotá&desde=2025-11- 01&hasta=2025-11-30
Resultado esperado: HTTP 200; {tipoServicio, nivelTransporte, cantidad, porcentaje} orden desc
Evidencia: Aggregation: match por ciudad/fecha → group por tipo/nivel → sort desc

4. (7%) Población de la base de datos:

Para cumplir con el requerimiento de población de la base de datos, se diseñó e implementó un script de inserción en MongoDB, con el cual se cargaron datos de prueba suficientes y representativos en las colecciones de usuarios, vehículos, servicios y revisiones. Previamente, el script ejecuta un proceso de limpieza mediante deleteMany() sobre cada colección, garantizando que no se generen duplicados en caso de ser ejecutado múltiples veces.

Se crearon usuarios con diferentes combinaciones de roles (conductores, usuarios de servicio y ambos), permitiendo validar los distintos escenarios funcionales del sistema. Posteriormente, se registraron vehículos asociados a conductores, incluyendo configuraciones reales de disponibilidad por día, tipo de servicio habilitado y capacidad. A su vez, se insertaron servicios de tipo pasajeros, comida y mercancías, distribuidos en distintas ciudades, con rutas, tiempos, costos y estados finalizados, lo que permite probar adecuadamente los requerimientos funcionales de consulta RFC1 y RFC2. Finalmente, se incluyeron revisiones cruzadas entre usuarios y conductores, con calificaciones y comentarios, asegurando la trazabilidad completa del servicio.

Este conjunto de datos garantiza una base consistente, relacionalmente coherente y suficiente para la validación integral de las consultas y funcionalidades del sistema. El script utilizado se encuentra en la carpeta docs dentro del repositorio.

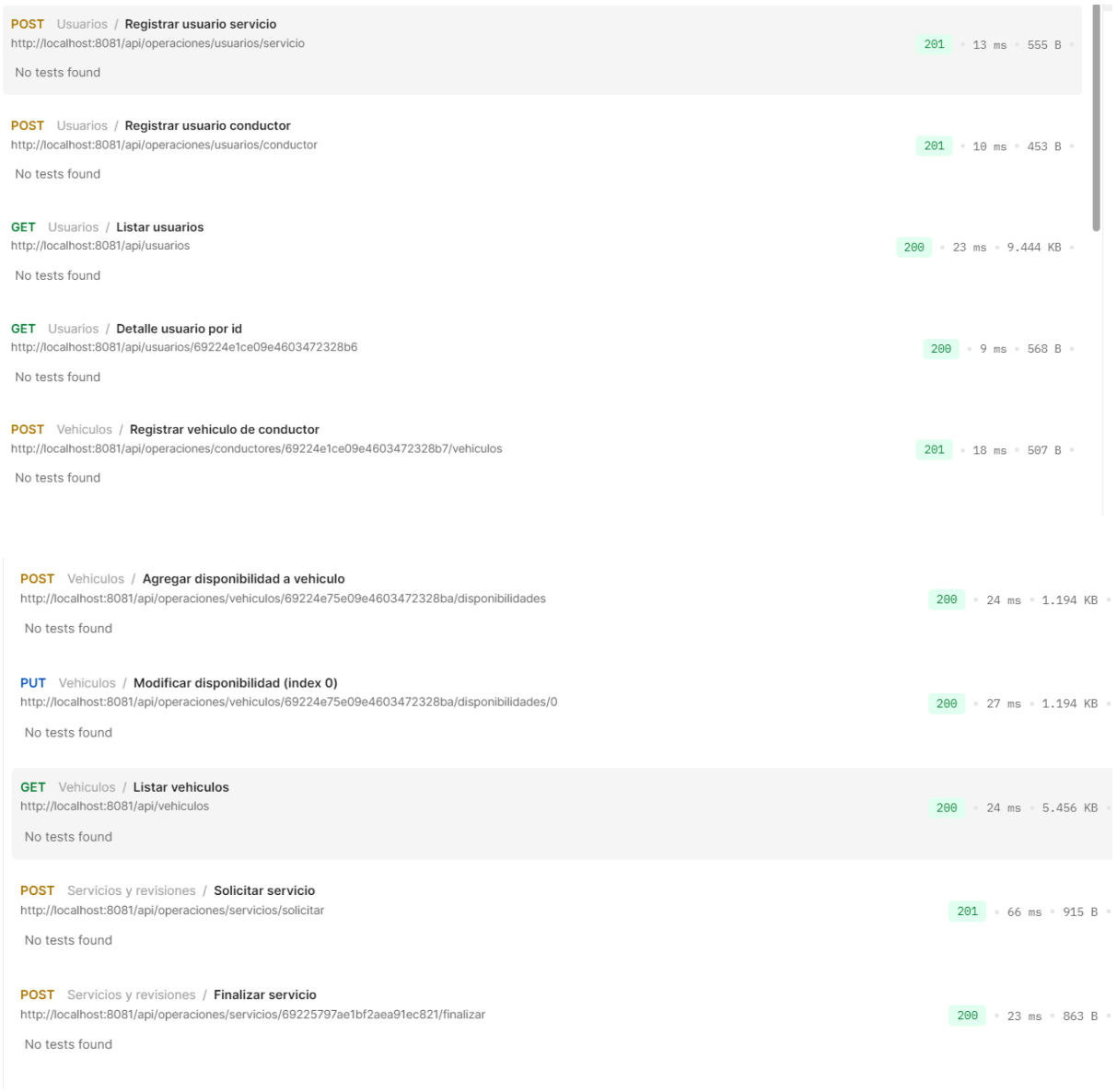
5. (46%) Implementación de los requerimientos funcionales: Implemente los 7 requerimientos funcionales y los 3 requerimientos funcionales de consulta usando Spring. Para la implementación es necesario utilizar las clases que corresponden a la lógica usadas en entregas pasadas, siempre y cuando se hagan las adaptaciones pertinentes para la implementación de los requerimientos de esta entrega. Cada requerimiento debe ser probado exitosamente usando su respectivo escenario de prueba (definidos en el punto3), más las pruebas que se consideren necesarias durante la sustentación. El valor de cada implementación es el siguiente:
c.
(28%) La implementación de cada requerimiento funcional tiene un peso de 4%.
•
(18%) La implementación de cada requerimiento funcional de consulta tiene un peso de 6%.

Para esta entrega se implementaron en su totalidad los 7 requerimientos funcionales (RF) y los 3 requerimientos funcionales de consulta (RFC) utilizando el framework Spring Boot, apoyándose en las capas de Controlador, Servicio y Repositorio, de acuerdo con la arquitectura trabajada en entregas anteriores. Sentíamos que nuestro modelo viejo era muy difícil de aplicar a una BD no relacional y, por lo tanto, tomamos la decisión de empezar la aplicación desde 0.

Cada requerimiento fue expuesto mediante su respectivo endpoint REST, permitiendo la creación, modificación, consulta y gestión de las entidades del sistema conforme a lo definido en los escenarios del punto 3 del proyecto. Los requerimientos funcionales de consulta (RFC) fueron implementados mediante consultas específicas sobre las colecciones, permitiendo validar correctamente los resultados esperados.

Todos los requerimientos fueron probados exitosamente mediante Postman. La evidencia completa de la implementación se encuentra disponible en el repositorio del proyecto, donde se incluyen los controladores, servicios, repositorios y configuraciones necesarias para la ejecución del sistema.

Además, acá hay constancia de las pruebas de Postman y su resultado:



POST Servicios y revisiones / **Crear revision de servicio**
http://localhost:8081/api/revisiones
201 • 10 ms • 548 B •

No tests found

GET Servicios y revisiones / **Listar servicios**
http://localhost:8081/api/servicios
200 • 27 ms • 9.415 KB •

No tests found

GET Consultas / **Historico por usuario**
http://localhost:8081/api/consultas/usuarios/69224e1ce09e4603472328b6/servicios
200 • 16 ms • 5.503 KB •

No tests found

GET Consultas / **Top conductores**
http://localhost:8081/api/consultas/conductores/top
200 • 12 ms • 351 B •

No tests found

GET Consultas / **Servicios por ciudad y rango**
http://localhost:8081/api/consultas/ciudad?ciudad=Bogotá&inicio=2025-01-01T00:00:00Z&fin=2025-12-31T23:59:59Z
200 • 15 ms • 354 B •

No tests found