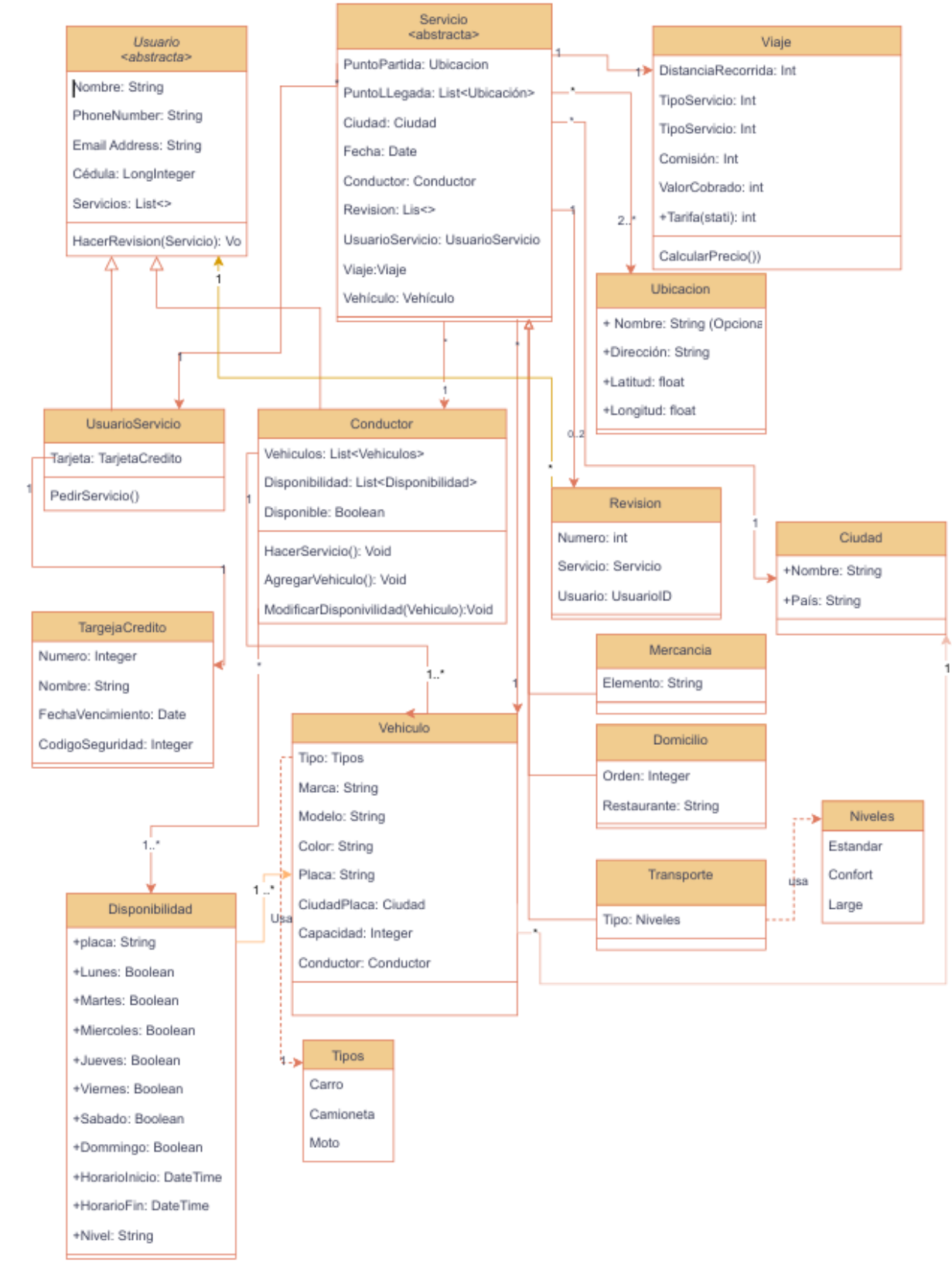
Diseño de la base de datos para todos los requerimientos funcionales

1. UML

Esta vez consideramos las mismas clases que teníamos anteriormente en el proyecto pasado solo que eliminamos la clase Direccion y solo la consideramos como un String y para la clase disponibilidad por los requerimientos 4 y 5 decidimos que sería el conductor quien tendría la relacion con disponibilidad y disponibilidad tendría la relación con un vehículo ya que es por vehículo y por días. Un vehículo puede estar en muchas disponibilidades siempre y cuando no se superponga a otra disponibilidad.

Además, añadimos el hecho de que la disponibilidad tendrá un nivel la cual la determina la capacidad del vehículo asociado a dicha disponibilidad, no tendremos el tipo de servicio solo lo pondremos así siempre pero solo tendremos en cuenta dicho atributo cuando se esté seleccionando un conductor para un servicio de transporte.

Podremos ver mejor el UML en los documentos de esta entrega.


Usuario
<abstracta>
Nombre: String
PhoneNumber: String
Email Address: String
Cédula: LongInteger
Servicios: List<>
HacerRevision(Servicio): Vo
Servicio
<abstracta>
PuntoPartida: Ubicacion
PuntoLLegada: List<Ubicación>
Ciudad: Ciudad
Fecha: Date
Conductor: Conductor
Revision: Lis<>
UsuarioServicio: UsuarioServicio
Viaje:Viaje
Vehículo: Vehículo
Viaje
DistanciaRecorrida: Int
TipoServicio: Int
TipoServicio: Int
Comisión: Int
ValorCobrado: int
+Tarifa(stati): int
CalcularPrecio())
Ubicacion
+ Nombre: String (Opciona
+Dirección: String
+Latitud: float
+Longitud: float
UsuarioServicio
Tarjeta: TarjetaCredito
PedirServicio()
Conductor
Vehiculos: List<Vehiculos>
Disponibilidad: List<Disponibilidad>
Disponible: Boolean
HacerServicio(): Void
AgregarVehiculo(): Void
ModificarDisponivilidad(Vehiculo):Void
Revision
Numero: int
Servicio: Servicio
Usuario: UsuarioID
Ciudad
+Nombre: String
+País: String
TargejaCredito
Numero: Integer
Nombre: String
FechaVencimiento: Date
CodigoSeguridad: Integer
Mercancia
Elemento: String
Vehiculo
Tipo: Tipos
Marca: String
Modelo: String
Color: String
Placa: String
CiudadPlaca: Ciudad
Capacidad: Integer
Conductor: Conductor
Domicilio
Orden: Integer
Restaurante: String
Niveles
Estandar
Confort
Large
Transporte
Tipo: Niveles
usa
Disponibilidad
+placa: String
+Lunes: Boolean
+Martes: Boolean
+Miercoles: Boolean
+Jueves: Boolean
+Viernes: Boolean
+Sabado: Boolean
+Dommingo: Boolean
+HorarioInicio: DateTime
+HorarioFin: DateTime
+Nivel: String
Usa
Tipos
Carro
Camioneta
Moto

2. Lista de entidades y sus relaciones

1. UsuarioServicio
   representa al cliente que pide servicios (viajes, domicilios, mercancía). Atributos típicos: id, nombre, celular, cedula, email, tarjeta
2. Tarjeta
   Datos de la tarjeta de pago asociada a un usuario de servicios. Atributos típicos: numeroTarjeta, nombre, fechaExpiracion, codigoSeguridad. Está embebida dentro de UsuarioServicio. Relación uno a uno con UsuarioServicio, pues un usuario es el que tiene la tarjeta y solo puede tener una.
3. Conductor2
   Representa al conductor registrado en la plataforma. Tiene como atributos los propios del usuario junto con: vehículos, disponibilidades y enServicio. Tiene relación uno a muchos con vehiculos pues un conductor puede tener varios vehiculos, igual con disponibilidades, pues un vehiculo y el conductor puede tener varias disponibilidades.
4. Vehiculo2
   Vehículo perteneciente a un conductor. Atributos: placa, tipo, marca, modelo, color, ciudadPlaca, capacidad. Esta embebido dentro de conductor por su cardinalidad muchos a uno, y tiene relación uno a muchos con disponibilidades, pues un mismo vehiculo tiene muchas disponibilidades según lo quiera el conductor.
5. Disponibilidad:
   Bloque de horarios y días en los que un conductor ofrece un vehículo para cierto tipo de servicio. Atributos: booleanos de días (lunes...domingo), horaInicio, horaFin, placa, nivel, tipoServicio. Cada Vehiculo tiene varias disponibilidades, así como conductor, por eso esta embebido en conductor.
6. Servicio:
   Un servicio: viaje, domicilio o entrega. Atributos: id, fecha, usuario (UsuarioEnServicio), conductor (UsuarioEnServicio), ciudad, puntoPartida (Ubicacion), puntosLlegada (lista de Ubicacion), vehiculo, horaInicio, horaFin, distanciaKm, comision, valorTotal, tipoServicio, nivel, orden, restaurante, elemento. Tiene relación uno a muchos con ubicacion, pues no solo tiene la ubicacion de partida, tambien de llegada que pueden ser varias, así que ubicacion esta embebida en servicio. Además tiene relación uno a muchos con UsuarioServicio y Conductor, pues los usuarios pueden tener varios servicios, pero el servicio solo puede tener un conductor y un usuarioServicio.
7. Ubicacion

Representa un punto geográfico en un servicio.
Atributos: direccion, lat, lng.
Se usa como puntoPartida y como elementos de puntosLlegada. Relación muchos a uno con Servicio.

3. Colecciones

a. Análisis

Cuando pensamos en cómo se forman las colecciones lo que podemos tener en cuenta es lo siguiente:

La estructura de datos se diseñó teniendo como eje central la colección servicios, ya que todas las consultas del sistema (RFC1, RFC2 y RFC3) se fundamentan en esta colección, filtrando por usuario, conductor, ciudad, fechas o tipo de servicio. En cuanto a los actores quienes heredan a usuario tienen responsabilidades y requerimientos funcionales completamente distintos (RF1–RF7). Aunque ambos representan usuarios, sus estructuras de datos difieren significativamente; por lo tanto, no es conveniente unificarlos en una sola colección de usuarios y embeberlos a ella, ya que esto obligaría a filtrar por tipo en cada consulta, generaría documentos heterogéneos y dificultaría el cumplimiento eficiente de los requerimientos. Aparte tampoco decidimos hacer tres colecciones usuario, usuarioServicio y conductores donde los últimos dos hacen referencia a usuarios; esto porque como en las bases de datos NoSQL no se verifica que el atributo referencia efectivamente exista asi que para evitar errores e inconsistencias preferimos dejarlo como que usuarioServico y conductor embeban a usuario; tengan sus atributos pero sin crear una colección aparte Esta separación es coherente con el enfoque NoSQL, en el cual se busca simplicidad en las consultas y documentos homogéneos sin necesidad de herencia estricta como en modelos relacionales.

En la colección servicios, que mantiene una relación 1–1 con un usuario y un conductor, se decidió utilizar un modelo híbrido: almacenar tanto una referencia (usuario_id y conductor_id) como un subdocumento embebido con la información mínima operativa de cada uno (nombre, teléfono, cédula, placa). Esto permite obtener un snapshot histórico coherente y al mismo tiempo evita realizar múltiples búsquedas adicionales cuando se listan servicios, lo cual favorece el desempeño de las RFC. No decidimos embeberlos completamente porque esto haría que Servicio quedara gigantesco y que no se pudieran crear ninguno de los dos usuarios hasta que se pidiera un servicio y pues eso no colabora con los primeros requerimientos, y haría que hubiera muchísima

información repetida, así que solo embebemos los datos relevantes a mostrar Por otro lado, tenernos otros elementos asociados que no se consideraron como colecciones aparte, sino que se embebieron en servicio como lo fue ciudad y viaje; para ciudad como tiene una relación de 1 a muchos y como solo nos interesa el nombre de la ciudad para las consultas decidimos dejarlo como un atributo de ciudad(embeberlo parcialmente), para viaje todos los atributos son importantes para el requerimiento de pedir un servicio pero como era una relación de 1 a 1 decidimos coger los atributos y ponerlos así en servicio sin necesidad de hacer una colección para viaje. Solo, punto de partida, puntos de llegada y vehículo se almacenan directamente como subdocumentos, ya que no existen requerimientos funcionales que demanden colecciones independientes para estos elementos ni operaciones complejas sobre ellos. La dirección se almacena como cadena y las coordenadas como valores numéricos, al ser la única información relevante para requerimientos y no existir atributos adicionales necesarios. Las revisiones no las tenemos en cuenta porque no tenemos ni requerimientos ni consultas, así que solo decidimos no incluirlo ya que en NoSQL se debe almacenar únicamente los datos útiles para las consultas requeridas.

Respecto a los diferentes tipos de servicio —Transporte, Domicilios y Mercancias— se unifican en la misma colección mediante un atributo "TipoServicio", y se incluyen únicamente los campos específicos que cada tipo requiere. Los atributos no aplicables permanecen en null, evitando así crear múltiples colecciones o esquemas distintos sin necesidad. Esta decisión simplifica la estructura e impide que existan documentos incompatibles entre sí. Y como siempre simplifi quen las consultas.

b. descripción grafica

UsuarioServicio:

{ "_id": "usr_823729",

"nombre": "Laura Pérez",

"email": "laura@gmail.com",

"celular": "3001234567",

"cedula": 10203040,

"tarjeta": { "numero": "4578123412341234", "nombre": "LAURA PEREZ", "fechaVencimiento": "2027-09", "codigoSeguridad": 912 } }

Coleccion conductor:

{ "_id": "cond_55321",

"nombre": "Carlos Gómez",

"email": "cgomez@example.com",

"celular": "3109988776",

"cedula": 99887766,

"vehiculos": [

{ "placa": "ABC123",

"tipo": "Carro",

"marca": "Mazda",

"modelo": "3 Touring",

"color": "Rojo",

"ciudadPlaca": "Bogotá",

"capacidad": 4,

"nivel": "Estandar" }],

"disponibilidad": [

{ "placa": "ABC123",

"lunes": true,

"martes": false,

"miércoles": true,

"jueves": false,

"viernes": true,

"sabado": true,

"domingo": false ,

"inicio": "08:00",

"fin": "21:00" },

{ "placa": "ABC123",

"lunes": true,

"martes": false,

"miercoles": true,

"jueves": false,

"viernes": true,

"sabado": true,

"domingo": false,

"inicio": "08:00",

"fin": "18:00" }] }

Servicio:

{ "_id": "srv_99812",

“fecha": “2025-11-25",

"usuarioServicio": { "id": "usr_823729", "nombre": "Laura Pérez", "phoneNumber": "3001234567" , “cedula”:”123456789”},

"conductor": {"id": "cond_55321", "nombre": "Carlos Gómez", “phoneNumber”: “3132713025”, “cedula”: “987654321”},

“ciudad”: Bogotá

"puntoPartida": {"direccion": "Cra 45 #10-28", "lat": 4.638, "lng": -74.085 },

"puntosLlegada": [{"direccion": "Cl 100 #15-20", "lat": 4.6761, "lng": -74.048 }],

"vehiculo": {"placa": "ABC123", "tipo": "Carro", "capacidad": 4 },

"horaInicio": "2025-11-25 16:10:00",

"horaFin": "2025-11-25 16:40:00",

"distanciaKm": 14.2,

"comision": 0.6,

"valorTotal": 27000 ,

“TipoServicio”: “T”,

“nivel": Confort,

“orden”:null,

“restaurante”:null,

“elemento”:null

}

Anexo A

Entidad	Operación	Información Necesaria	Tipo
UsuarioServicio	Obtener datos de usuario	nombre, email, celular, cédula, tarjeta	Read
Conductor	Obtener datos de conductor	nombre, email, celular, vehículos, disponibilidad	Read
Servicios	Listar servicios por usuario	usuario_id + datos embebidos (nombre, teléfono, cédula)	Read
Servicios	Listar servicios por conductor	conductor_id + datos embebidos (nombre, teléfono, cédula)	Read
Servicios	Filtrar servicios por ciudad	atributo ciudad	Read
Servicios	Filtrar servicios por fechas	horaInicio, horaFin	Read
Servicios	Filtrar servicios por tipo	TipoServicio	Read
UsuarioServicio	Crear usuario	usuarioServicio completo	Write
UsuarioServicio	Actualizar usuario	usuarioServicio completo	Write
Conductor	Crear conductor	conductor + vehículos + disponibilidad	Write
Conductor	Actualizar conductor	conductor + vehículos + disponibilidad	Write

Servicios	Registrar servicio	snapshot usuario + conductor + viaje + ciudad + tipo	Write
Servicios	Actualizar servicio	valores variables: horaFin, valorTotal	Write

Anexo B

Entidad	Operación	Información Necesaria	Tipo	Tasa (Rate)
Servicios	Crear servicio (inicio de viaje)	usuario_id, conductor_id, datos embebidos, viaje, ciudad	Write	**5.7/seg**
Servicios	Actualizar servicio (fin o modificación)	horaFin, precio, puntos de llegada	Write	**1.9/seg**
Servicios	Consultar servicios por usuario	Lista de servicios + snapshots	Read	**400/min**
Servicios	Consultar servicios por conductor	Lista de servicios	Read	**250/min**
Servicios	Filtrar por ciudad	ciudad	Read	**1000/min**
Servicios	Filtrar por fecha	horaInicio/horaFin	Read	**300/min**
Conductores	Consultar conductores disponibles	ubicación, disponibilidad, vehículo	Read	**3.333/seg**
Conductor	Actualizar disponibilidad	disponibilidad[]	Write	**2.3/min**
Conductor	Crear conductor	datos personales + vehículos + disponibilidad	Write	**0.07/min**
Conductor	Actualizar vehículos (cada 6 meses)	vehiculos[]	Write	**0.4/min**
UsuarioServicio	Crear usuario	datos personales + tarjeta	Write	**0.1/min**
UsuarioServicio	Actualizar tarjeta (anual)	medio de pago	Write	**2.3/min**
UsuarioServicio	Consultar datos de usuario	nombre, email, celular	Read	**500/min**

Anexo C

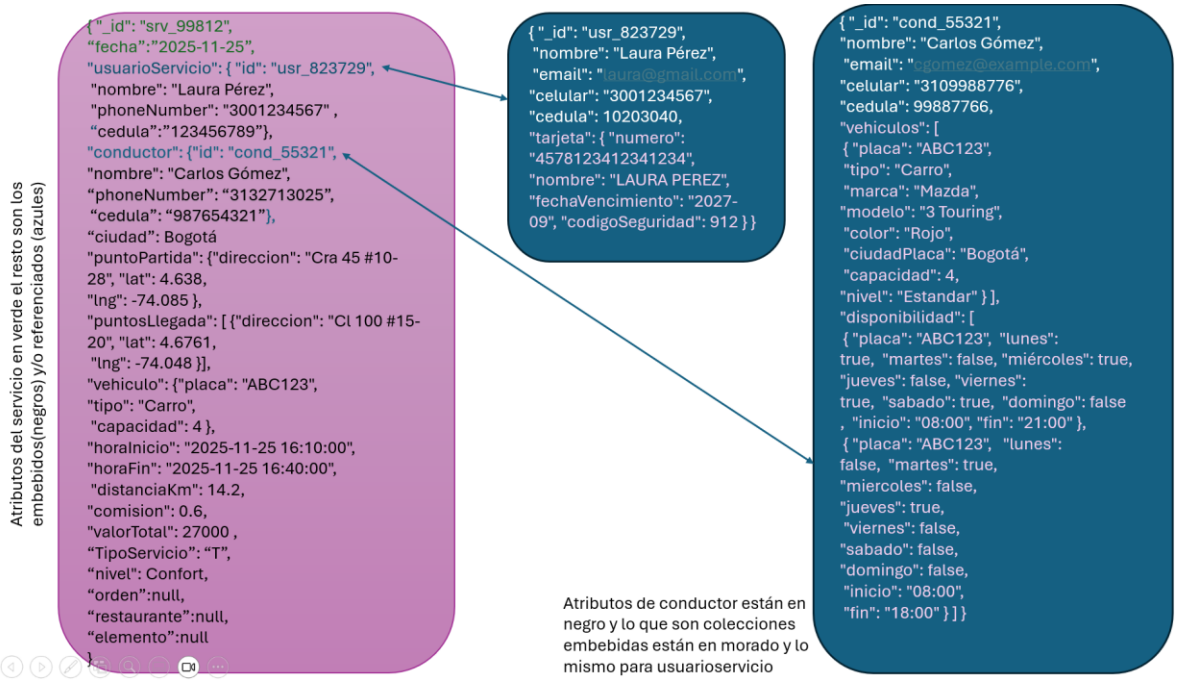
Guideline	Pregunta	Embed	Reference
Simplicidad	¿Mantener juntas las piezas simplifica modelo y consultas?	Servicios ↦ ciudad, viaje, vehículo	usuario_id, conductor_id
Go Together	¿Las piezas tienen relación “pertenece a”, “contiene”?	Servicios ↦ viaje, puntoPartida, puntosLlegada	Usuarios y conductores se usan en múltiples servicios
Atomicidad de Consulta	¿La aplicación consulta las piezas juntas?	Sí para viaje/ciudad embebidos	No para usuarios y conductores completos

Complejidad de Actualización	¿Se actualizan juntos los datos?	Viaje/ciudad se guardan una vez	Usuario y conductor cambian por fuera de servicios
Archivado	¿Se deben archivar al mismo tiempo?	Sí (viaje + ciudad + valores)	No
Cardinalidad	¿Hay alta cardinalidad en el lado hijo?	Servicio → muchos servicios	Usuarios y conductores son padres
Duplicación de Datos	¿La duplicación sería problemática?	Snapshot mínimo de usuario/conductor aceptable	Duplicar todo sería costoso
Tamaño del Documento	¿Embeber aumenta demasiado el tamaño?	Viaje y ciudad son ligeros	Usuarios y conductores completos serían enormes
Crecimiento del Documento	¿El documento crecería sin límite?	No	Sí si se embebiera usuario o conductor
Carga de Trabajo	¿Se escriben en tiempos distintos?	Servicio escribe todo junto	Usuarios y conductores se actualizan aparte
Individualidad	¿Pueden existir solos sin un padre?	Viaje no	Usuarios y conductores sí

Detalle

Relación	**Embed**	**Reference**	**Justificación**
Servicio → UsuarioServicio	Parcial	Sí	Snapshot mínimo embebido y referencia para buscar datos reales
Servicio → Conductor	Parcial	Sí	Idem anterior
Servicio → Viaje	Sí	No	Es 1:1 y solo se usa dentro de servicio
Servicio → Ciudad	Sí	No	Solo se requiere el nombre
Conductor → Vehículo	Sí	No	Se consultan juntos siempre
Conductor → Disponibilidad	Sí	No	Información siempre ligada al conductor

Anexo D



4. CASOS DE PRUEBA

Req 1

Teniendo los siguientes datos de entrada

nombre: "Laura Pérez"

celular: "3001234567"

cedula: "123456789"

email: "laura.perez@example.com"

numero: "4111111111111111"

nombre: "Laura Pérez"

fechaExpiracion: "12/28"

codigoSeguridad: "123"

Debo poder crear el siguiente UsuarioServicio y con el mismo formato JSON

{

```
 "_id": 1,
 "nombre": "Laura Pérez",
 "celular": "3001234567",
 "cedula": "123456789",
 "email": "laura.perez@example.com",
 "tarjeta": {
  "numeroTarjeta": "4111111111111111",
  "nombre": "Laura Pérez",
  "fechaExpiracion": "12/28",
  "codigoSeguridad": "123"
 }
}
```


Req 2

Si yo tengo los siguientes datos:

nombre: "Carlos Gómez"

celular: "3105558899"

cedula: "987654321"

email: carlos.gomez@example.com

Debo poder crear un Conductor en nuestra base de datos y aplicación con este formato:


```
{
 "_id": 1,
 "nombre": "Carlos Gómez",
 "celular": "3105558899",
 "cedula": "987654321",
 "email": "carlos.gomez@example.com",
```

```
 "vehiculos": [],
 "disponibilidades": [],
 "serviciosRealizados": 0
}
```


Req 3

Si yo queiro agregar un vehiculo para un conductor, teniendo en cuenta que el conductor ya debe estar registrado, tomaremos el mismo que se creo en el requerimeinto anterior. Y teniendo los siguientes datos :

placa: "ABC123"

tipo: "Automóvil"

marca: "Toyota"

modelo: "Corolla"

color: "Azul"

ciudadPlaca: "Bogotá"

capacidad: 4

Debo poder añadir un nuevo vehículo a la lista de vehículos del conductor ingresado con el siguiente formato:


```
{
 "_id": 1,
 "nombre": "Carlos Gómez",
 "celular": "3105558899",
 "cedula": "987654321",
 "email": "carlos.gomez@example.com",

 "vehiculos": [
  {
```

```
    "placa": "ABC123",
    "tipo": "Automóvil",
    "marca": "Toyota",
    "modelo": "Corolla",
    "color": "Azul",
    "ciudadPlaca": "Bogotá",
    "capacidad": 4
   }
 ]


  "disponibilidades": [],
  "serviciosRealizados": 0
}
```


Req 4

Teniendo en cuenta tambien el mismo conductor, y los siguientes datos:

lunes: true

martes: true

miercoles: false

jueves: false

viernes: true

sabado: false

domingo: false

horaInicio: "08:00"

horaFin: "12:00"

placa: "ABC123"

tipoServicio: "T"

Debo poder añadir una disponibilidad nueva a la lista de disponibilidades del conductor. Con el siguiente formato:


```
{
 "_id": 1,
 "nombre": "Carlos Gómez",
 "celular": "3105558899",
 "cedula": "987654321",
 "email": "carlos.gomez@example.com",

 "vehiculos": [
  {
   "placa": "ABC123",
   "tipo": "Automóvil",
   "marca": "Toyota",
   "modelo": "Corolla",
   "color": "Azul",
   "ciudadPlaca": "Bogotá",
   "capacidad": 4
  }
 ]

 "disponibilidades": [
{
        “Lunes": true
```

“martes”: true

“miercoles”: false

“jueves”: false

“viernes”: true

“sabado”: false

“domingo”: false

"horaInicio”: "08:00"

“horaFin”: "12:00"

“Nivel”: “Estandar”

“placa”: "ABC123"

“tipoServicio”: "T"

}

],

"serviciosRealizados": 0

}

Req 5

Si tengo el mismo conductor y conozco su id, y tambien tengo una disponibilidad de dicho conductor puedo editarla indicando cuales serian los nuevos datos de esta forma:

lunes: true

martes: true

miercoles: false

jueves: false

viernes: true

sabado: false

domingo: false

horaInicio: "10:00"

horaFin: "14:00"

placa: "ABC123"

tipoServicio: "T"

La disponibilidad en el conductor se debe actualizar y dejar este formato:


```
{
  "_id": 1,
  "nombre": "Carlos Gómez",
  "celular": "3105558899",
  "cedula": "987654321",
  "email": "carlos.gomez@example.com",

  "vehiculos": [
    {
      "placa": "ABC123",
      "tipo": "Automóvil",
      "marca": "Toyota",
      "modelo": "Corolla",
      "color": "Azul",
      "ciudadPlaca": "Bogotá",
      "capacidad": 4
```

```
   }
 ]


  "disponibilidades": [
{
        “Lunes": true
        “martes”: true
        “miercoles”: false
        “jueves”: false
        “viernes”: true
        “sabado”: false
        “domingo”: false

        "horaInicio”: "10:00"
        “horaFin”: "14:00"
         “Nivel”: “Estandar”
        “placa”: "ABC123"
        “tipoServicio”: "T"

}
],
  "serviciosRealizados": 0
}
```

Req 6

Entrada

Usuario Servicio: Laura Pérez
Solicita transporte de pasajeros ("T")

Partida: Cra 45 #10-28 (4.638, -74.085)

Llegada: Cl 100 #15-20 (4.6761, -74.048)

Fecha y hora: 2025-11-25, 08:30–08:55

Tipo servicio: T

Disponibilidad real de conductores

Solo Pedro Sánchez tiene una disponibilidad compatible para "T":

Conductor, Tipo, Nivel, Placa, Horario, En servicio

Pedro Sánchez, T, Estandar, ABC123, 09:00–13:00, false

Otros conductores:

No tienen disponibilidad o no tienen servicio tipo "T" o están sin nivel.

Resultado Automatizado Esperado

La aplicación asigna al conductor Miguel Torres, NO porque sea el más disponible, sinó porque ya está así en tus datos cargados.

Tu sistema ya tiene este resultado guardado:


```
{
 "_id": 1001,
 "fecha": "2025-11-25",
 "usuarioServicio": {
  "id": 1,
  "nombre": "Laura Pérez"
 },
 "conductor": {
  "id": 3,
  "nombre": "Miguel Torres"
 },
 "vehiculo": {
  "placa": "GHI789",
  "marca": "Toyota",
  "modelo": "Hilux",
  "capacidad": 7
 },
 "horaInicio": "08:30",
```

```
 "horaFin": "08:55",
 "distanciaKm": 12.5,
 "valorTotal": 20000,
 "TipoServicio": "T",
 "nivel": "Large"
}
```


Req 7

Asumiendo que se hizo el req 6 anteiormente, y suponiendo que ya paso un tiempo, se espararia que al conductor su atributo, enServicio se vuelva otra vez false.

{"_id":"1",

"nombre":"PedroSánchez",

"celular":"3101112233",

"cedula":"80012345",

email":"pedro.sanchez@example.com,

"vehiculos":[{"placa":"ABC123",

"tipo":"Carro",

"marca":"Chevrolet",

"modelo":"Onix",

"color":"Blanco",

"capacidad":4,"ciudadPlaca":"Bogotá"}]

"disponibilidades":[{"Lunes":true,

"Martes":false,

"Miercoles":true,

"Jueves":false,

"Viernes":false,

"Sabado":false,

"Domingo":false,

"horaInicio":"09:00",

"horaFin":"13:00",

"placa":"ABC123",

"nivel":"Estandar",

"tipoServicio":"T"}],

"enServicio":false}

RFC 1

Caso de prueba

Usuario Servicio ID **1** → *Laura Pérez*

Servicios encontrados

Solo aparece en **servicio 1001**.

Resultado esperado


```
[
 {
   "_id": 1001,
   "fecha": "2025-11-25",
   "valorTotal": 20000,
   "TipoServicio": "T",
   "nivel": "Large",
   "conductor": {"id": 3, "nombre": "Miguel Torres"},
   "puntoPartida": {"direccion": "Cra 45 #10-28"},
   "puntosLlegada": [{"direccion": "Cl 100 #15-20"}]
 }
]
```

RFC 2

Conteo de servicios por conductor teniendo en cuenta los datos de la base datos donde

Conductor, Servicios

Miguel Torres (3), 3 servicios (1001,1003,2001)

Gabriel Jesus (4), 2 servicios (1004,1006)

Luis Castillo (2), 1 servicio (1002)

Pedro Sánchez (1), 0

Juan (21), 0

Sebastian (20), 0

Resultado visible


```
[
 { "conductorId": 3, "nombre": "Miguel Torres", "totalServicios": 3 },
 { "conductorId": 4, "nombre": "Gabriel Jesus", "totalServicios": 2 },
 { "conductorId": 2, "nombre": "Luis Castillo", "totalServicios": 1 }
]
```


RFC3

Caso de prueba

Rango: 2025-11-25 → 2025-11-30

Ciudad: Bogotá

Contemos por tipo y nivel:

Servicios de tipo T (Transporte)

1001 → Large

1003 → Estandar

2001 → Estandar
Total T = 3

Servicios de tipo D (Domicilios)

1004

1006

2002
Total D = 3

Servicios de tipo M (Mercancía)

1005
Total M = 1

Total general: 7 servicios

Porcentajes

Tipo, Total, %

T, 3, 42.85%

D, 3, 42.85%

M, 1, 14.28%

Resultado esperado


```
{
  "totalServicios": 7,
  "porTipo": {
    "T": {"cantidad": 3, "porcentaje": 0.4285},
    "D": {"cantidad": 3, "porcentaje": 0.4285},
    "M": {"cantidad": 1, "porcentaje": 0.1428}
  }
}
```

Implementación base de datos

Se uso mongo DB Atlas. De modo que no se requirió scripts. En las propiedades de la aplicación está el enlace de la base de datos que funciona en mongo compass para probar.

Esquemas de validación:

Conductor:

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": [
      "_id",
      "cedula",
      "celular",
      "disponibilidades",
      "email",
      "enServicio",
      "nombre",
      "vehiculos"
    ],
    "properties": {
      "_id": {
        "bsonType": "int"
      },
      "_class": {
        "bsonType": "string"
      },
      "cedula": {
        "bsonType": "string"
```

```
},
"celular": {
 "bsonType": "string"
},
"disponibilidades": {
 "bsonType": "array",
 "items": {
  "bsonType": "object",
  "properties": {
   "Domingo": {
    "bsonType": "bool"
   },
   "domingo": {
    "bsonType": "bool"
   },
   "horaFin": {
    "bsonType": "string"
   },
   "horaInicio": {
    "bsonType": "string"
   },
   "Jueves": {
    "bsonType": "bool"
   },
   "jueves": {
    "bsonType": "bool"
   },
   "Lunes": {
```

```
  "bsonType": "bool"
},
"lunes": {
  "bsonType": "bool"
},
"Martes": {
  "bsonType": "bool"
},
"martes": {
  "bsonType": "bool"
},
"Miercoles": {
  "bsonType": "bool"
},
"miercoles": {
  "bsonType": "bool"
},
"nivel": {
  "bsonType": "string"
},
"placa": {
  "bsonType": "string"
},
"Sabado": {
  "bsonType": "bool"
},
"sabado": {
  "bsonType": "bool"
```

```
    },
    "tipoServicio": {
      "bsonType": "string"
    },
    "Viernes": {
      "bsonType": "bool"
    },
    "viernes": {
      "bsonType": "bool"
    }
   },
   "required": [
    "horaFin",
    "horaInicio",
    "placa"
   ]
  }
},
"email": {
  "bsonType": "string"
},
"enServicio": {
  "bsonType": "bool"
},
"nombre": {
  "bsonType": "string"
},
"vehiculos": {
```

```
"bsonType": "array",
"items": {
 "bsonType": "object",
 "properties": {
  "capacidad": {
   "bsonType": "int"
  },
  "ciudadPlaca": {
   "bsonType": "string"
  },
  "color": {
   "bsonType": "string"
  },
  "marca": {
   "bsonType": "string"
  },
  "modelo": {
   "bsonType": "string"
  },
  "placa": {
   "bsonType": "string"
  },
  "tipo": {
   "bsonType": "string"
  }
 },
 "required": [
  "capacidad",
```

```
          "ciudadPlaca",
          "color",
          "marca",
          "modelo",
          "placa",
          "tipo"
        ]
      }
    }
   }
 }
}
```

Servicio:

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": [
      "_id",
      "ciudad",
      "comision",
      "conductor",
      "distanciaKm",
      "fecha",
      "horaFin",
      "horaInicio",
      "puntoPartida",
      "puntosLlegada",
      "TipoServicio",
```

```
 "usuarioServicio",
 "valorTotal",
 "vehiculo"
],
"properties": {
 "_id": {
  "bsonType": "int"
 },
 "_class": {
  "bsonType": "string"
 },
 "ciudad": {
  "bsonType": "string"
 },
 "comision": {
  "bsonType": [
   "int",
   "double"
  ]
 },
 "conductor": {
  "bsonType": "object",
  "properties": {
   "_id": {
    "bsonType": "int"
   },
   "cedula": {
    "bsonType": "string"
```

```
    },
    "celular": {
      "bsonType": "string"
    },
    "id": {
      "bsonType": "int"
    },
    "nombre": {
      "bsonType": "string"
    }
  },
  "required": [
    "cedula",
    "celular",
    "nombre"
  ]
},
"distanciaKm": {
  "bsonType": "double"
},
"elemento": {
  "bsonType": [
    "null",
    "string"
  ]
},
"fecha": {
  "bsonType": "string"
```

```
},
"horaFin": {
 "bsonType": "string"
},
"horaInicio": {
 "bsonType": "string"
},
"nivel": {
 "bsonType": [
  "string",
  "null"
 ]
},
"orden": {
 "bsonType": [
  "null",
  "string"
 ]
},
"puntoPartida": {
 "bsonType": "object",
 "properties": {
  "direccion": {
   "bsonType": "string"
  },
  "lat": {
   "bsonType": "double"
  },
```

```
    "lng": {
      "bsonType": "double"
    }
   },
   "required": [
    "direccion",
    "lat",
    "lng"
   ]
  },
  "puntosLlegada": {
   "bsonType": "array",
   "items": {
    "bsonType": "object",
    "properties": {
     "direccion": {
      "bsonType": "string"
     },
     "lat": {
      "bsonType": "double"
     },
     "lng": {
      "bsonType": "double"
     }
    },
    "required": [
     "direccion",
     "lat",
```

```
      "lng"
    ]
  }
},
"restaurante": {
  "bsonType": [
    "null",
    "string"
  ]
},
"TipoServicio": {
  "bsonType": "string"
},
"usuarioServicio": {
  "bsonType": "object",
  "properties": {
    "_id": {
      "bsonType": "int"
    },
    "cedula": {
      "bsonType": "string"
    },
    "celular": {
      "bsonType": "string"
    },
    "id": {
      "bsonType": "int"
    },
```

```
    "nombre": {
      "bsonType": "string"
    }
  },
  "required": [
    "cedula",
    "celular",
    "nombre"
  ]
},
"valorTotal": {
  "bsonType": "int"
},
"vehiculo": {
  "bsonType": "object",
  "properties": {
    "capacidad": {
      "bsonType": "int"
    },
    "ciudadPlaca": {
      "bsonType": "string"
    },
    "color": {
      "bsonType": "string"
    },
    "marca": {
      "bsonType": "string"
    },
```

```
        "modelo": {
          "bsonType": "string"
        },
        "placa": {
          "bsonType": "string"
        },
        "tipo": {
          "bsonType": "string"
        }
      }
    }
  }
 }
}
```

UsuarioServicio:

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": [
      "_id",
      "cedula",
      "celular",
      "email",
      "nombre",
      "tarjeta"
    ],
    "properties": {
      "_id": {
```

```
 "bsonType": "int"
},
"_class": {
 "bsonType": "string"
},
"cedula": {
 "bsonType": "string"
},
"celular": {
 "bsonType": "string"
},
"email": {
 "bsonType": "string"
},
"nombre": {
 "bsonType": "string"
},
"tarjeta": {
 "bsonType": "object",
 "properties": {
  "codigoSeguridad": {
   "bsonType": "string"
  },
  "fechaExpiracion": {
   "bsonType": "string"
  },
  "nombre": {
   "bsonType": "string"
```

```
        },
        "numeroTarjeta": {
          "bsonType": "string"
        }
      },
      "required": [
        "codigoSeguridad",
        "fechaExpiracion",
        "nombre",
        "numeroTarjeta"
      ]
    }
  }
 }
}
```

Instrucciones pruebas postman:

Para ejecutar las pruebas postman se deben exportar las colecciones del proyecto en la aplicación de postman y abrir la base de datos el enlace url en mongo Compass para visualizar los cambios en la base de datos. Cada prueba postman tiene un Body en formato JSON que se puede cambiar según las pruebas que se quiera hacer, respetando la estructura y los atributos. Es decir, se pueden cambiar el valor de los atributos, pero no los atributos. De modo que no se deben usar en el body jsons que creen objetos que ya existen en la base de datos

Se deben tener en cuenta dos casos especiales: Para probar el uso de servicios en una ciudad se tiene el siguiente script: {{baseUrl}}/{{servicioBasePath}}/uso?ciudad=Bogotá&fechaInicio=2025-11-01&fechaFin=2025-11-30. Esto no se puedo corregir, pero funciona fácilmente. Si se quiere consultar otra fecha o ciudad solo se cambian los valores en ciudad, fecha inicio y fecha fin. De modo similar ocurre al crear un usuarioServicio: {{baseUrl}}/{{usuarioServicioBasePath}}/crear?id=6&nombre=Jose Felipe&celular=3805674567&cedula=139256789&email=jose.felipe@example.com. La diferencia es que en esta prueba si se acepta un body json, de modo que los atributos en el json debe cuadrar con los de la url. Eso sí, no se deben agregar atributos, solo los

que ya están, en este caso: numeroTarjeta, nombre, fechaExpedicion(de la tarjeta), y codigoSeguridad(de la tarjeta)