PUNTO 1

1. Ciudades

Descripción:
Representa las ciudades donde opera el sistema. Donde se determinan las tarifas, conductores y vehículos disponibles en cada región.

Relaciones:

- **Ciudad 1 — N Conductores**
- **Ciudad 1 — N Vehículos**
- **Ciudad 1 — N Tarifas**
- **Ciudad 1 — N Viajes**

2. Usuarios

Descripción:
Personas registradas en la plataforma, incluyendo pasajeros y conductores

Relaciones:

- **Usuario 1 — N Conductores** (solo si un usuario puede tener un perfil de conductor)
- **Usuario 1 — N Viajes** (como pasajero)
- **Usuario 1 — N Calificaciones**
- **Usuario 1 — N Métodos de Pago**

3. Conductores

Descripción:
Conductores registrados que pueden realizar viajes. Asociados a un usuario y a una ciudad base.

Relaciones:

- **Conductor N — 1 Usuario**
- **Conductor N — 1 Ciudad**

- **Conductor 1 — N Vehículos**
- **Conductor 1 — N Viajes**
- **Conductor 1 — N Calificaciones**

4. Vehículos

Descripción:
Vehículos registrados en la plataforma, asociados a un conductor y a una ciudad.

Relaciones:

- **Vehículo N — 1 Conductor**
- **Vehículo N — 1 Ciudad**
- **Vehículo 1 — N Disponibilidad**
- **Vehículo 1 — N Viajes**

5. Disponibilidad

Descripción:
Indica los horarios en los que un vehículo puede prestar servicio.

Relaciones:

- **Disponibilidad N — 1 Vehículo**

6. Tarifas

Descripción:
Define la tarifa por ciudad según el tipo de vehículo y tipo de servicio. Incluye tarifa base y precio por kilómetro.

Relaciones:

- **Tarifa N — 1 Ciudad**

7. Viajes

Descripción:
Registra un servicio de transporte: pasajero, conductor, vehículo usado, origen, destino, costos y estado del viaje.

Relaciones:

- **Viaje N — 1 Usuario** (pasajero)
- **Viaje N — 1 Conductor**
- **Viaje N — 1 Vehículo**
- **Viaje N — 1 Ciudad**
- **Viaje 1 — N Paradas**

8. Paradas

Descripción:
Representan puntos intermedios opcionales dentro de un viaje.

Relaciones:

- **Parada N — 1 Viaje**

9. Calificaciones

Descripción:
Opiniones que los usuarios dejan a los conductores después del viaje, con puntuación y comentario.

Relaciones:

- **Calificación N — 1 Usuario**
- **Calificación N — 1 Conductor**

10. Métodos de Pago

Descripción:
Formas de pago asociadas a un usuario: tarjeta, efectivo, billetera digital, etc.

Relaciones:

- **Método de Pago N — 1 Usuario**

PUNTO 2:

. **1. Usuario — Métodos de Pago**

Selección: Embebido

Justificación:

- Cada usuario tiene muy pocos métodos de pago (1–2 en promedio).
- Se consultan siempre junto al usuario (perfil, pago en viaje).
- Se actualizan muy poco (una vez al año según carga de trabajo).

2. Usuario — Conductor

Selección: Normalizado

Justificación:

- Solo algunos usuarios son conductores.
- Los conductores tienen muchas entidades asociadas (vehículos, disponibilidad, viajes).
- Embebido haría que el documento Usuario creciera demasiado, lo cual no es buena idea ya que no seria optimo.

3. Conductor — Vehículos

Selección: Normalizado

Justificación:

- Conductores pueden tener varios vehículos.
- Vehículos tienen cosas adicionales, lo cual causaría que el documento fuera muy grande si fuera embebido.
- Actualizaciones frecuentes.

4. Vehículo — Disponibilidad

Selección: Embebido

Justificación:

- Las disponibilidades no son muchas
- Actualizaciones mensuales
- no es una amplia carga de información

5. Usuario — Viajes (como pasajero)

Selección: Normalizado

Justificación:

- Cada usuario puede tener **muchos** viajes
- se requieren varias consultas en torno a los viajes
- Viajes se actualizan mientras están en curso.

6. Conductor — Viajes

Selección: Normalizado

Justificación:

- Mimos argumentos que en anterior.

- Las actualizaciones realizadas.

7. Viaje — Paradas

Selección: Embebido

Justificación:

- Un viaje tiene pocos puntos intermedios.
- Son parte clave del viaje y no cambian.
- tienen poca información

8. Conductor — Calificaciones / Usuario — Calificaciones

Selección: Normalizado

Justificación:

- Conductores y usuarios pueden tener muchas calificaciones.
- las consultas que se deben realizar
- las actualizaciones que existen frente a esto.

9. Ciudad — Conductores / Ciudad — Vehículos / Ciudad — Viajes / Ciudad — Tarifas

Selección: Normalizado

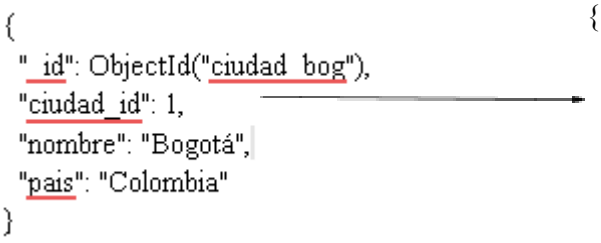
Justificación:

- Una ciudad puede tener muchos elementos, lo que lo haría muy grande.

PUNTO 3

Ciudades — Tarifas

Ciudad:

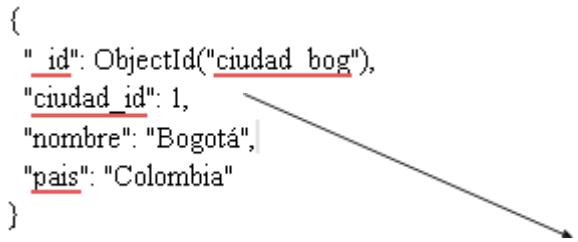


Tarifa:

```
{
 "_id": ObjectId("tarifa_1"),
 "tarifa_id": 101,
 "ciudad_id": 1,
 "tipo_servicio": "transporte_pasajeros",
 "tipo_vehiculo": "Estándar",
 "precio_km": 1200,
 "tarifa_base": 3000
}
```

Ciudades — Viajes

Ciudades:



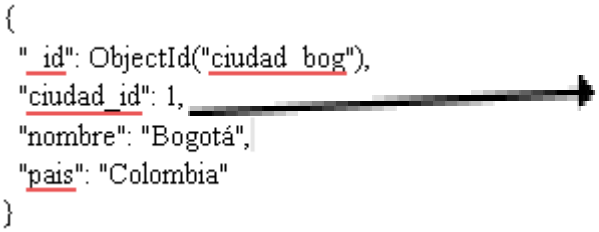
Viajes:

```
{
 "_id": ObjectId("viaje_1001"),
 "viaje_id": 1001,
 "usuario_id": 5001,
 "conductor_id": 2001,
 "vehiculo_id": 3001,
 "ciudad_id": 1,
 "tipo_servicio": "transporte_pasajeros",
 "estado": "completed",
 "distancia_km": 12.4,
 "tarifa_final": 15000.00,
 "fecha_inicio":
ISODate("2025-11-28T12:10:00Z"),
 "fecha_fin": ISODate("2025-11-28T12:40:00Z")
}
```

Ciudades — Vehículos

Ciudades:

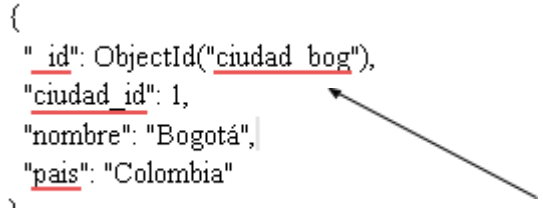
Vehículos:




```
{
 "_id": ObjectId("veh_3001"),
 "vehiculo_id": 3001,
 "conductor_id": 2001,
 "ciudad_id": 1,
 "placa": "ABC123",
 "modelo": "Corolla 2018",
 "tipo_vehiculo": "carro",
 "capacidad": 4
}
:
```

Ciudades — Conductores

ciudades:



conductores:

```
{
 "_id": ObjectId("cond_2001"),
 "conductor_id": 2001,
 "usuario_id": 4001,
 "ciudad_id": 1,
 "numero_licencia": "LIC-998877",
 "calificacion_avg": 4.8,
 "estado": activo,
 “total_viajes”:0
“posactual”:[
{“latitud_temp : 4,609
“ longitud_temp:-74,086
]
}
```

Usuario — Métodos de Pago:

```
{
 "_id": ObjectId("usr_5001"),
 "usuario_id": 5001,
 "nombre": "María Sanchez",
 "correo": "maria@gmail.com",
 "metodosPago": [
   {
    "metodo_pago_id": 1,
    "tipo": "tarjeta",
    "ultimos4": "1234",
    "proveedor": "VISA",
    "fecha_registro": ISODate("2024-05-10T10:00:00Z")
   }
 ]
```

}

Usuario — Conductor:

Usuario:

```
{
 "_id": ObjectId("usr_4001"),
 "usuario_id": 4001,
 "nombre": "Juan Cho",
 "correo": "juan@example.com"
}
```

Conductor:

```
{
 "_id": ObjectId("cond_2001"),
 "conductor_id": 2001,
 "usuario_id": 4001,
 “ciudad”: 1,
 "numero_licencia": "LIC-998877”
 "calificacion_avg": 4.8,
 "estado": activo,
}
```

Conductor — Vehículos:

Vehiculo:

```
{
 "_id": ObjectId("veh_3002"),
 "vehiculo_id": 3002,
 "conductor_id": 2001,
 "ciudad_id": 1,
 "placa": "DEF456",
 "modelo": "Ranger 2020",
 "tipo_vehiculo": "Confort",
 "capacidad": 4
}
```

Conductor:

```
{
 "_id": ObjectId("cond_2001"),
 "conductor_id": 2001,
 "usuario_id": 4001,
 "ciudad_id": 1,
 "numero_licencia": "LIC-998877",
 "calificacion_avg": 4.8,
 "estado": activo,
}
```

Vehículo — Disponibilidad

```
{
 "_id": ObjectId("veh_3001"),
 "vehiculo_id": 3001,
 "placa": "ABC123",
 "disponibilidad": [
   { "disponibilidad_id": 1, "dia_semana": "Lunes", "hora_inicio": "08:00", "hora_fin": "12:00" },
   { "disponibilidad_id": 2, "dia_semana": "Lunes", "hora_inicio": "14:00", "hora_fin": "18:00" }
 ]
}
```

Usuario — Viajes:

viajes:

```
{
```

Usuario:

```
{
 "_id": ObjectId("usr_4001"),
 "usuario_id": 4001,
 "nombre": "Juan Cho",
 "correo": "juan@example.com"
}
```

```
  "_id": ObjectId("viaje_1002"),
  "viaje_id": 1002,
  "usuario_id": 4001,
  "conductor_id": 2001,
  "vehiculo_id": 3002,
  "ciudad_id": 1,
  "tipo_servicio": "entrega_comida",
  "estado": "completed",
  “ubicacion_inicio”:cali,
  “ubicacion_inicio”:Barranquilla,
  "distancia_km": 5.2,
  "tarifa_final": 8000.00
}
```

Viaje — Paradas:

```
{
  "_id": ObjectId("viaje_1003"),
  "viaje_id": 1003,
  "usuario_id": 5002,
  "conductor_id": 2002,
  "tipo_servicio": "entrega_comida",
  "estado": "completed",
  “ubicacion_inicio”:cali,
  “ubicacion_inicio”:Barranquilla,
  "paradas": [
    { "orden": 1, "tipo": "start", "direccion": "Cra 7 #45", "ubicacion": { "type": "Point", "coordinates": [-74.0739, 4.6097] } },
    { "orden": 2, "tipo": "stop",  "direccion": "Cll 100 #10", "ubicacion": { "type": "Point", "coordinates": [-74.0410, 4.6766] } }
  ],
  "estado": "completado"
  "distancia_km": 5.2,
  "tarifa_final": 8000.00

}
```

Calificaciones:

```
{
  "_id": ObjectId("calif_9001"),
  "calificacion_id": 9001,
  "usuario_id": 5001,
  "conductor_id": 2001,
  "puntuacion": 5.0,
  "comentario": "Muy buen servicio, puntual y amable.",
  "fecha_creada": Date("2025-11-28T13:05:00")
}
```

PUNTO 3:

Archivo adjunto a la entrega.

RF1 — REGISTRAR UN USUARIO DE SERVICIOS

1. Nombre: RF1-Crear usuario válido

Requerimiento: RF1

Objetivo: Verificar que la API permita crear un usuario con los campos obligatorios y que se guarde correctamente en la colección Usuarios.

Precondiciones:

- No existe un usuario con el mismo correo ni usuario_id.

Datos de entrada (JSON):


```
{
 "usuario_id": 50010,
 "nombre": "Ana Gómez",
 "correo": "ana.gomez@gmail.com",
 "telefono": "+573001112233",
 "rol": "pasajero",
 "fecha_creado": "2025-11-30T00:00:00Z",
 "metodosPago": [
  { "metodo_pago_id": 1, "tipo": "tarjeta", "ultimos4": "4242", "proveedor": "VISA" }
 ]
}
```


Resultado esperado:

- Respuesta 201 con body que incluye usuario_id: 50010.
- En la colección Usuarios existe un documento con usuario_id: 50010 y correo: "ana.gomez@gmail.com".

RF1 — CREAR USUARIO (CASO NEGATIVO)

2. Nombre: No permite correo duplicado

Objetivo: Verificar que no se pueda crear un usuario con un correo ya existente.

Precondiciones:

- Existe Usuarios con correo: "ana.gomez@gmail.com"

Datos de entrada: mismo correo pero con diferente usuario_id.

Resultado esperado:

- Registro no creado.
- Mensaje de error indicando correo duplicado.

RF2 — REGISTRAR UN USUARIO CONDUCTOR

3. Nombre: RF2-Crear conductor válido

Requerimiento: RF2

Objetivo: Verificar que un usuario pueda registrarse como conductor y crear el documento en Conductores vinculándolo a Usuarios.

Precondiciones:

- Existe usuario en Usuarios con usuario_id: 40010.

Datos de entrada (JSON):


```
{
  "conductor_id": 20010,
  "usuario_id": 40010,
  "ciudad_id": 1,
  "numero_licencia": "LIC000999",
  "calificacion_avg": 0.0,
  "estado": "activo"
}.
```


Resultado esperado:

- Conductores contiene el conductor con usuario_id: 40010.
- La relación está intacta.

RF3 — REGISTRAR UN VEHÍCULO PARA UN USUARIO CONDUCTOR

4. Nombre: RF3-Agregar vehículo a conductor

Requerimiento: RF3

Objetivo: Validar registro de vehículo y que quede asociado al conductor_id.

Precondiciones:

- Existe Conductores con conductor_id: 20010.

Datos de entrada (JSON):

```
{
  "vehiculo_id": 30010,
  "conductor_id": 20010,
  "ciudad_id": 1,
  "placa": "XYZ123",
  "modelo": "Toyota Corolla 2018",
  "tipo_vehiculo": "Estándar",
  "capacidad": 4,
  "anio": 2018,
  "disponibilidad": []
}
```

Resultado esperado:

- Vehiculos contiene el nuevo vehículo con conductor_id: 20010.
- Conductores puede listar sus vehículos mediante query db.Vehiculos.find({ conductor_id: 20010 }).

RF4 — REGISTRAR LA DISPONIBILIDAD DE UN USUARIO CONDUCTOR Y SU VEHÍCULO

5. Nombre: RF4-Registrar disponibilidad válida (sin solapamiento)

Requerimiento: RF4

Objetivo: Verificar que se pueda añadir una disponibilidad y que no se permitan solapamientos por conductor.

Precondiciones:

- Existe Vehiculos con vehiculo_id: 30010 y conductor_id: 20010.

- No hay disponibilidades anteriores para ese conductor en el mismo rango.

Datos de entrada (JSON):

```
{
  "vehiculo_id": 30010,
  "dia_semana": "Lunes",
  "hora_inicio": "08:00",
  "hora_fin": "12:00",
  "tipo_servicio": "transporte_pasajeros"
}
```

Resultado esperado:

- HTTP 201 y disponibilidad guardada.
- db.Vehiculos.findOne({ vehiculo_id: 30010 }).disponibilidad contiene el nuevo rango.

RF4 — CASO NEGATIVO

6. Nombre: RF4-Rechaza disponibilidad solapada

Objetivo: Intentar registrar disponibilidad que se solapa con otra existente del **mismo conductor** y verificar rechazo.

Precondiciones:

- En conductor 20010 existe disponibilidad Lunes 08:00–12:00 en vehiculo_id: 30010.

Datos de entrada (JSON):

```
{
  "vehiculo_id": 30011,          // otro vehículo del mismo conductor
  "dia_semana": "Lunes",
  "hora_inicio": "11:00",
  "hora_fin": "15:00"
}
```

Resultado esperado:

- No se agrega disponibilidad.
- Mensaje de error claro.

RF5 — MODIFICAR LA DISPONIBILIDAD DE UN VEHÍCULO PARA SERVICIOS

7. Nombre: RF5-Modificar disponibilidad válida

Requerimiento: RF5

Objetivo: Verificar que la modificación de disponibilidad respete que no se solape.

Precondiciones:

- Vehículo 30010 tiene disponibilidad Lunes 08:00–12:00.
- Vehículo 30012 (mismo conductor) no tiene solapamientos en el nuevo valor.

Datos de entrada (JSON):

```
{
  "disponibilidad_id": 1,
  "hora_inicio": "07:00",
  "hora_fin": "11:00"
}
```

Resultado esperado:

- HTTP 200 y disponibilidad actualizada.
- DB refleja new times.

Caso negativo: Si la modificación genera solapamiento, HTTP 400 y no se aplica cambio.

RF6 — SOLICITAR UN SERVICIO POR PARTE DE UN USUARIO

8. Nombre: RF6-Solicitar servicio y asignar conductor cercano disponible

Requerimiento: RF6

Objetivo: Validar ruta completa: cálculo, tarifa, cobro, selección y asignación automática de conductor disponible y cercano.

Precondiciones:

- Existen conductores con current_status.available = true y ubicación dentro del radio de búsqueda.
- Tarifas definidas para la ciudad_id.

- Usuario con usuario_id: 50010 y método de pago válido.

Datos de entrada (JSON):


```
{
  "usuario_id": 50010,
  "ciudad_id": 1,
  "tipo_servicio": "transporte_pasajeros",
  "nivel": "Estándar",
  "punto_partida": { "direccion": "Cra 7 #45", "location": { "type": "Point", "coordinates": [-74.0739, 4.6097] } },
  "punto_llegada": { "direccion": "Cll 100 #10", "location": { "type": "Point", "coordinates": [-74.0410, 4.6766] } }
}
```


Resultado esperado:

- HTTP 201, Viajes creado con estado: "requested" o "accepted".
- Conductores asignado marcado temporalmente no disponible.
- tarifa_estim coincide con cálculo.

Caso negativo: Si no hay conductores disponibles, retornar HTTP 503 con mensaje "No hay conductores disponibles".

RF7 — REGISTRAR UN VIAJE

9. Nombre: RF7-Finalizar viaje y registrar histórico

Requerimiento: RF7

Objetivo: Validar que al finalizar el viaje se calculen costos finales, se actualice conductor a disponible, y se registre el histórico con datos completos.

Precondiciones:

- Existe Viajes con estado: "in_progress" y viaje_id: 7001.

Datos para completar (JSON):


```
{
  "viaje_id": 7001,
  "fecha_fin": "2025-11-30T12:40:00Z",
  "distancia_km": 12.4,
```

"tarifa_final": 15000.0
}

Resultado esperado:

- HTTP 200.
- Viajes document actualizado con estado: "completed", tarifa_final, fecha_fin.
- Conductores ahora available = true.

REQUERIMIENTOS FUNCIONALES DE CONSULTA

RFC1 — HISTÓRICO DE TODOS LOS SERVICIOS PEDIDOS POR UN USUARIO

10. Nombre: RFC1-Consultar histórico de usuario (paginado)

Requerimiento: RFC1

Objetivo: Verificar que la consulta devuelve todos los viajes de un usuario con la información relevante y ordenados por fecha descendente.

Precondiciones:

- Usuario 50010 tiene al menos 5 viajes completados insertados en Viajes.

Parámetros de consulta:

- GET /api/usuarios/50010/viajes

Resultado esperado:

- Respuesta 200 con lista de viajes ordenada por fecha_inicio desc.
- Cada viaje incluye viaje_id, fecha_inicio, fecha_fin, distancia_km, tarifa_final, conductor_id, vehiculo_id, paradas.

RFC2 — TOP 20 CONDUCTORES POR NÚMERO DE SERVICIOS

11. Nombre: RFC2-Generar ranking top 20 conductores

Requerimiento: RFC2

Objetivo: Validar, mediante una consulta de agregación, el atributo *total_viajes* (contador de viajes finalizados) definido a través del *computed pattern*, y aplicar un ordenamiento descendente con un límite para obtener únicamente a los 20 conductores con mayor número de viajes.

Precondiciones:

- Viajes contiene suficientes registros diversos para varios conductore..

Resultado esperado:

- Lista de 20 documentos ordenada por total desc.
- Cada documento incluye conductor_id y total.

RFC3 — UTILIZACIÓN DE SERVICIOS EN UNA CIUDAD EN RANGO DE FECHAS

12. Nombre: RFC3-Utilización por ciudad y rango de fechas

Requerimiento: RFC3

Objetivo: Obtener conteo de servicios por tipo_servicio y nivel (si aplica) en una ciudad entre dos fechas, con porcentaje respecto al total, ordenado de más a menos.

Precondiciones:

- Viajes contiene viajes en ciudad_id: 1 entre 2025-11-01 y 2025-11-30.

Parámetros de consulta:

- GET /api/estadisticas/ciudad/1?fecha_inicio=2025-11-01&fecha_fin=2025-11-30

Resultado esperado:

- Lista ordenada desde servicio/nivel más usado hasta menos usado.
- Suma de percentage ≈ 100%: