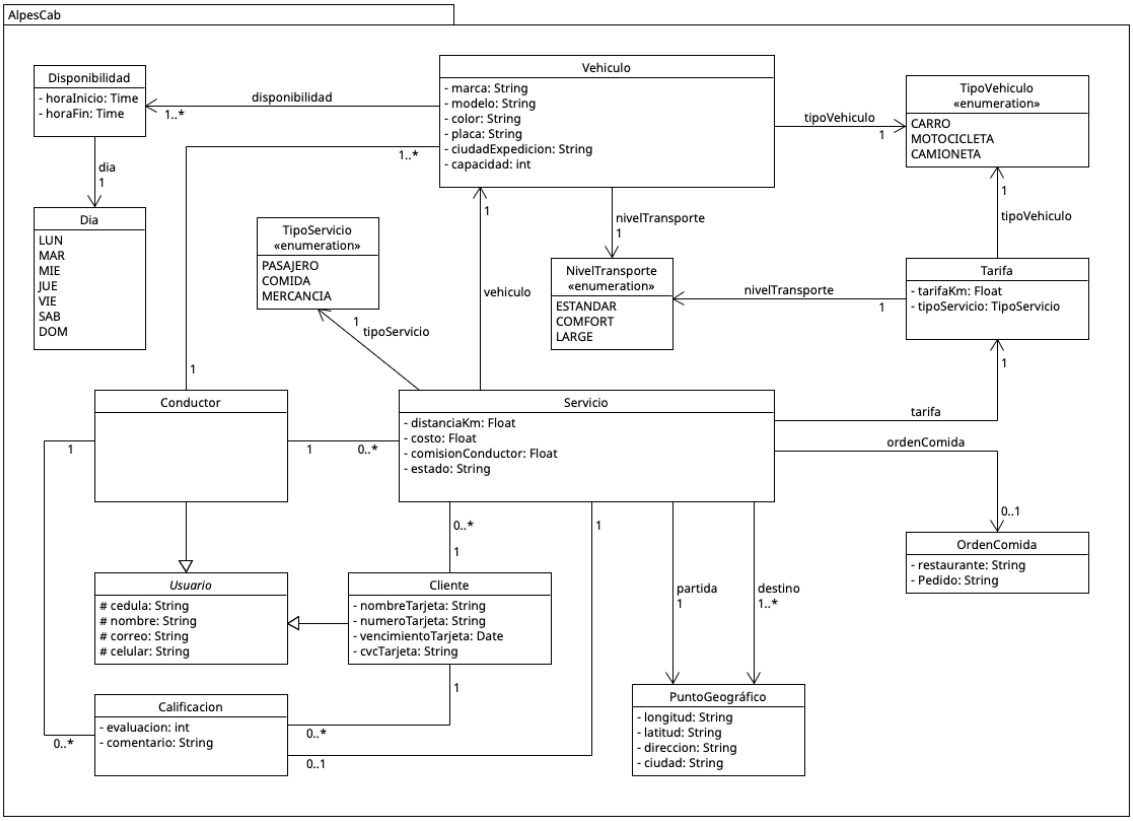
Sistemas Transaccionales – Proyecto: Entrega 3
Implementación MongoDB

Carlos Ramírez, 202121639

1. DISEÑO DE LA BASE DE DATOS

a. Modelo conceptual UML:



b. Entidades y relaciones:

USUARIO:
Entidad abstracta que describe los datos básicos de cualquier persona registrada en la plataforma (cliente o conductor).

CLIENTE:
Usuario que solicita servicios (viajes, entrega de comida, envíos de mercancías). Registra sus datos personales y los datos de su tarjeta de crédito.

Origen	Destino	Cardinalidad	Descripción
CLIENTE	SERVICIO	1 : 0..*	Un cliente puede solicitar cero, uno o muchos servicios.
CLIENTE	CALIFICACION	1 : 0..*	Un cliente puede hacer muchas calificaciones (una por servicio realizado como mínimo).

CONDUCTOR:
Usuario que presta los servicios, asociado a uno o varios vehículos, registra sus disponibilidades y recibe pagos.

Origen	Destino	Cardinalidad	Descripción
CONDUCTOR	VEHICULO	1 : 1..*	Un conductor debe tener al menos un vehículo registrado, pero puede tener varios.
CONDUCTOR	SERVICIO	1 : 0..*	Un conductor puede prestar cero, uno o muchos servicios.
CONDUCTOR	CALIFICAICION	1 : 0..*	Un conductor puede calificar o recibirlas.

VEHICULO:
Medio de transporte utilizado por un conductor para prestar el servicio (carro, moto, camioneta en los que se presta un servicio.

Origen	Destino	Cardinalidad	Descripción
VEHICULO	SERVICIO	1 : 0..*	Un mismo vehículo puede participar en muchos servicios a lo largo del tiempo.
VEHICULO	CONDUCTOR	1 : 1	Un vehículo está asociado a un conductor.

SERVICIO:
Medio de transporte utilizado por un conductor para prestar el servicio (carro, moto, camioneta en los que se presta un servicio.

Origen	Destino	Cardinalidad	Descripción
SERVICIO	ORIGEN (PUNTO_GEOGRAFICO)	1 : 1	Cada servicio tiene exactamente un punto de origen.
SERVICIO	PARADAS (PUNTO_GEOGRAFICO)	1 : 1..*	Cada servicio tiene al menos un destino, pero pueden existir múltiples destinos.

SERVICIO	ORDEN_COMIDA	1 : 0..1	Un servicio puede tener asociada una orden de comida, o no tenerla.
SERVICIO	TARIFA	1 : 1	Cada servicio se cobra con exactamente una tarifa determinada.

PUNTO_GEOGRAFICO:
Entidad que representa los distintos puntos de la ruta de un servicio (origen, paradas, destino) y muestran información como: dirección, coordenadas, ciudad

Origen	Destino	Cardinalidad	Descripción
PUNTO_ GEOGRAFICO	SERVICIO	2..* : 1	.Un servicio tiene por lo menos 2 puntos (origen y destino).
PUNTO_ GEOGRAFICO	CIUDAD	1 : 1	Todo punto está asociado a un ciudad.

TARIFA:
Condiciones de cobro: cuánto se cobra por kilómetro según tipo de servicio y tipo/nivel de vehículo.

Origen	Destino	Cardinalidad	Descripción
TARIFA	SERVICIO	1 : 0..*	Una misma configuración de tarifa puede aplicarse a muchos servicios diferentes.

ORDEN_COMIDA:
Condiciones de cobro: cuánto se cobra por kilómetro según tipo de servicio y tipo/nivel de vehículo.

Origen	Destino	Cardinalidad	Descripción
ORDEN_ COMIDA	SERVICIO	1 : 1	Una orden de comida está asociada a un único servicio de entrega.

CALIFICACIÓN:
Opinión que da el cliente después de un servicio, normalmente hacia el conductor (puntaje y comentario).

Origen	Destino	Cardinalidad	Descripción
CALIFICACION	CLIENTE	1 : 1	Cada calificación hace referencia a un único cliente calificado.
CALIFICACION	CONDUCTOR	1 : 1	Cada calificación hace referencia a un único conductor calificado.
CALIFICACION	SERVICIO	1 : 1	Cada calificación está asociada a un servicio concreto que ya ocurrió.

c. Esquema de asociación NoSQL:

Debido a la carga de trabajo descrita en el documento (muchos servicios y calificaciones, en comparación a pocos clientes, conductores y vehículos), se usaron referencias en el lado de "muchos" para las relaciones (clientes—servicios, conductores—servicios, vehículos—servicios, tarifas, ciudades y calificaciones). Mientras que colecciones como disponibilidades, rutas y puntos geográficos, fueron embebidas dentro de sus colecciones padres correspondientes dado que las consultas que se realizaban siempre necesitaban la información de estas a la vez.

d. Representación JSON de las relaciones:

- disponibilidades embebidas en vehículos:

```json
{
  "placa": "ABC123",
  "idConductor": "COND1",
  "disponibilidades": [
    {
      "dia": "LUNES",
      "horaInicio": "09:00",
      "horaFin": "12:00",
      "tipoServicio": "PASAJEROS"
    },
    {
      "dia": "MARTES",
      "horaInicio": "14:00",
      "horaFin": "18:00",
      "tipoServicio": "PASAJEROS"
    }
  ]
}
```

- punto_geografico embebida en servicio:

- ciudad referenciada en punto_geografico:

- conductor referenciada en vehiculo:

- conductor y cliente referenciadas en calificación (las referencias pueden cambiar entres sí, dado que tanto el autor como el calificado pueden ser documentos de cliente o de conductor):

- conductor y cliente referenciadas en servicio:

2. IMPLEMENTACIÓN DE LA BASE DE DATOS

Se utilizó el script “Implementación-BaseDatos,js” (que se adjuntó en el repositorio del proyecto para crear todas las colecciones de la base de datos, por medio de Mongosh dentro del Clúster que se usó para el proyecto.

3. ESCENARIOS DE PRUEBA

RF1 – Registrar un usuario de servicios

Caso exitoso: Se registra correctamente un usuario de servicios con todos los datos personales y de tarjeta.

Caso fallido: Se intenta registrar un usuario de servicios sin todos los campos obligatorios (faltan correo, celular y cédula), por lo que la operación debe fallar por validación.

RF2 – Registrar un usuario conductor

Caso exitoso: Se registra correctamente un usuario conductor con todos los datos personales diligenciados.

Caso fallido: Se intenta registrar un usuario de servicios sin todos los campos obligatorios (faltan correo, celular y cédula), por lo que la operación debe fallar por validación.

RF3 – Registrar un vehículo para un usuario conductor

Caso exitoso: Se registra un vehículo para un conductor que ya existe en la colección conductores. Se envían todos los campos obligatorios: placa, marca, modelo, capacidad, tipoVehiculo, nivelTransporte e idConductor válido.

Caso fallido: Se intenta registrar un vehículo al que le faltan datos obligatorios.

Caso fallido: Se intenta registrar un vehículo con un idConductor que no existe en la colección conductores y el sistema debe rechazar el registro.

RF4 – Registrar la disponibilidad de un usuario conductor y su vehículo para un servicio

Caso exitoso: Se registra una disnponibilidad para un vehículo existente, con conductor asociado, y la nueva disponibilidad no se cruza con ninguna anterior.

Caso fallido: Se intenta registrar disponibilidad para una placa que no está en la colección vehiculos.

Caso fallido: Se intenta registrar una disponibilidad para un mismo vehículo y la disponibilidad a registrar se cruza que las disnponibilidades del vehículo.

RF5 – Modificar la disponibilidad de un vehículo para servicios

Caso exitoso: Se modifica la disponibilidad de un vehículo existente para un día y tipo de servicio sin generar superposición con otras disponibilidades existentes.

Caso fallido: Se intenta modificar la disponibilidad de un vehículo existente, pero la nueva franja horaria se superpone con otra disponibilidad ya registrada para el mismo día y tipo de servicio, por lo que la operación es rechazada.

RF6 – Solicitar un servicio por parte de un usuario de servicios

Caso exitoso: Un cliente existente solicita un servicio de PASAJEROS, nivel ESTANDAR, indicando origen y destino (dirección + coordenadas). Existe una tarifa configurada para (PASAJEROS, ESTANDAR) y al menos un vehículo con disponibilidad.

Caso fallido: Un usuario intenta solicitar un servicio con un idCliente que no existe en la colección clientes.

RF7 – Registrar un viaje de para un usuario de servicios y un usuario conductor

Caso exitoso: Se finaliza un servicio existente actualizando hora de inicio, hora de fin, distancia y costo, quedando el servicio con estado FINALIZADO.

Caso fallido: Se intenta finalizar un servicio que no existe en la base de datos. El sistema notifica que no lo encuentra.

4. POBLACIÓN DE LA BASE DE DATOS

Pese a que no se realizaron los requerimientos funcionales de consulta, se utilizó el script “Población-BaseDatos,js”, por medio de mongosh, para poblar la base de datos.