**MongoDB Data Modelling for Car Pooling Platform**

**Overview**

This document outlines the MongoDB data model for the Car-Pooling Platform, designed to support comprehensive features including user management, ride posting, booking management, and user rating systems. The model strategically combines referencing for relationships and embedding for ratings to ensure optimal flexibility, scalability, and query efficiency.

**Modelling Approach**

The Car-Pooling Platform uses a **hybrid approach** combining:

* **Referencing** for main entity relationships (Users, Rides, Bookings)
* **Embedding** for ratings within user documents

**Why Referencing was Chosen:**

1. **Scalability**: Users can create multiple rides and bookings without document size limitations
2. **Data Integrity**: Separate collections prevent data duplication and maintain consistency
3. **Query Flexibility**: Individual collections can be queried and indexed independently
4. **Normalization**: Reduces data redundancy across the platform

**Why Embedding was Used for Ratings:**

1. **Performance**: Ratings are frequently accessed with user profiles
2. **Atomicity**: Rating updates happen atomically within user documents
3. **Size Management**: Limited number of ratings per user keeps documents manageable
4. **Query Efficiency**: User profiles with ratings retrieved in single query

**Collection Schemas**

**1. Users Collection**

**Purpose**: Stores comprehensive user profiles including roles, contact information, verification status, and embedded ratings.

**Schema Structure:**

{

\_id: ObjectId,

email: String (required, unique),

password: String (required, hashed),

name: String (required),

phone: String (required),

role: String (enum: ['admin', 'driver', 'rider']),

isVerified: Boolean (default: false),

emergencyContact: {

name: String,

phone: String

},

ratings: [

{

rating: Number (1-5),

comment: String,

ratedBy: ObjectId (reference to Users),

createdAt: Date

}

],

createdAt: Date,

updatedAt: Date

}

**2. Rides Collection**

**Purpose**: Represents rides posted by drivers, including detailed location information with coordinates, timing, and passenger management.

**Schema Structure:**

{

\_id: ObjectId,

driver: ObjectId (reference to Users),

startLocation: String (required),

startCoordinates: {

lat: Number (required),

lon: Number (required)

},

endLocation: String (required),

endCoordinates: {

lat: Number (required),

lon: Number (required)

},

departureTime: Date (required),

seatsAvailable: Number (required),

pricePerSeat: Number (default: 0),

status: String (enum: ['open', 'booked', 'completed', 'cancelled']),

passengers: [ObjectId] (references to Users),

createdAt: Date,

updatedAt: Date

}

**3. Bookings Collection**

**Purpose**: Tracks all booking transactions made by riders for specific rides, maintaining booking history and status.

**Schema Structure:**

{

\_id: ObjectId,

ride: ObjectId (reference to Rides),

rider: ObjectId (reference to Users),

seatsBooked: Number (required),

status: String (enum: ['pending', 'confirmed', 'cancelled']),

bookingDate: Date (default: current date),

createdAt: Date,

updatedAt: Date

}

**Entity Relationships**

1. **Users → Rides**: One-to-Many
   * One driver can post multiple rides
   * Implemented via driver field in Rides collection
2. **Rides → Bookings**: One-to-Many
   * One ride can have multiple bookings
   * Implemented via ride field in Bookings collection
3. **Users → Bookings**: One-to-Many
   * One rider can make multiple bookings
   * Implemented via rider field in Bookings collection
4. **Users → Ratings**: One-to-Many (Embedded)
   * One user can receive multiple ratings
   * Implemented as embedded array in Users collection
5. **Users ←→ Rides**: Many-to-Many (Passengers)
   * Multiple users can be passengers on multiple rides
   * Implemented via passengers array in Rides collection

