**🔧 Full Database Schema (Including Chat System)**

**1. Users Table**

sql

CopyEdit

CREATE TABLE Users (

UserID SERIAL PRIMARY KEY,

Name VARCHAR(100),

Email VARCHAR(100) UNIQUE NOT NULL,

PhoneNumber VARCHAR(15),

PasswordHash TEXT NOT NULL,

Role ENUM('CarOwner', 'Buyer', 'ServiceProvider', 'Admin') NOT NULL,

ProfileImage TEXT,

Verified BOOLEAN DEFAULT FALSE,

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**2. Vehicles Table**

sql

CopyEdit

CREATE TABLE Vehicles (

VehicleID SERIAL PRIMARY KEY,

OwnerID INT REFERENCES Users(UserID),

RegistrationNumber VARCHAR(50) UNIQUE NOT NULL,

Make VARCHAR(50),

Model VARCHAR(50),

Year INT,

FuelType VARCHAR(50),

Transmission VARCHAR(50),

KilometersDriven INT,

Location VARCHAR(255),

Description TEXT,

Status ENUM('Available', 'Sold') DEFAULT 'Available',

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**3. VehicleImages Table**

sql

CopyEdit

CREATE TABLE VehicleImages (

ImageID SERIAL PRIMARY KEY,

VehicleID INT REFERENCES Vehicles(VehicleID),

ImageURL TEXT NOT NULL

);

**4. Listings Table**

sql

CopyEdit

CREATE TABLE Listings (

ListingID SERIAL PRIMARY KEY,

VehicleID INT REFERENCES Vehicles(VehicleID),

ListedBy INT REFERENCES Users(UserID),

Price DECIMAL(12,2),

ListingDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

IsActive BOOLEAN DEFAULT TRUE

);

**5. ServiceProviders Table**

sql

CopyEdit

CREATE TABLE ServiceProviders (

ProviderID SERIAL PRIMARY KEY,

UserID INT REFERENCES Users(UserID),

BusinessName VARCHAR(255),

Address TEXT,

Verified BOOLEAN DEFAULT FALSE,

Rating FLOAT DEFAULT 0

);

**6. ServicesOffered Table**

sql

CopyEdit

CREATE TABLE ServicesOffered (

ServiceID SERIAL PRIMARY KEY,

ProviderID INT REFERENCES ServiceProviders(ProviderID),

ServiceName VARCHAR(255),

Description TEXT,

Price DECIMAL(10,2)

);

**7. Appointments Table**

sql

CopyEdit

CREATE TABLE Appointments (

BookingID SERIAL PRIMARY KEY,

VehicleID INT REFERENCES Vehicles(VehicleID),

ProviderID INT REFERENCES ServiceProviders(ProviderID),

UserID INT REFERENCES Users(UserID),

ServiceID INT REFERENCES ServicesOffered(ServiceID),

ScheduledDateTime TIMESTAMP,

Status ENUM('Booked', 'Completed', 'Cancelled') DEFAULT 'Booked'

);

**8. ServiceHistory Table**

sql

CopyEdit

CREATE TABLE ServiceHistory (

HistoryID SERIAL PRIMARY KEY,

VehicleID INT REFERENCES Vehicles(VehicleID),

ServiceDate DATE,

Description TEXT,

DocumentID INT REFERENCES Documents(DocumentID)

);

**9. Documents Table**

sql

CopyEdit

CREATE TABLE Documents (

DocumentID SERIAL PRIMARY KEY,

OwnerID INT REFERENCES Users(UserID),

DocumentType ENUM('RC', 'PUC', 'Insurance'),

FileURL TEXT,

UploadDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**10. Reviews Table**

sql

CopyEdit

CREATE TABLE Reviews (

ReviewID SERIAL PRIMARY KEY,

ReviewerID INT REFERENCES Users(UserID),

TargetID INT, -- Could be UserID or ProviderID

RoleReviewed ENUM('Seller', 'ServiceProvider'),

Rating INT CHECK (Rating BETWEEN 1 AND 5),

Comments TEXT,

Date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**11. AdminActions Table**

sql

CopyEdit

CREATE TABLE AdminActions (

ActionID SERIAL PRIMARY KEY,

AdminID INT REFERENCES Users(UserID),

TargetType ENUM('User', 'Listing', 'Review'),

TargetID INT,

ActionType ENUM('Verify', 'Ban', 'Flag', 'Resolve'),

Notes TEXT,

Timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**12. ChatRooms Table**

sql

CopyEdit

CREATE TABLE ChatRooms (

ChatRoomID SERIAL PRIMARY KEY,

BuyerID INT REFERENCES Users(UserID),

SellerID INT REFERENCES Users(UserID),

OwnerID INT REFERENCES Users(UserID),

ProviderID INT REFERENCES ServiceProviders(ProviderID),

ListingID INT REFERENCES Listings(ListingID),

AppointmentID INT REFERENCES Appointments(BookingID),

Type ENUM('BuyerSeller', 'OwnerProvider') NOT NULL,

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

* Note: Use CHECK constraints or triggers if needed to ensure the proper fields are filled depending on Type.

**13. ChatMessages Table**

sql

CopyEdit

CREATE TABLE ChatMessages (

MessageID SERIAL PRIMARY KEY,

ChatRoomID INT REFERENCES ChatRooms(ChatRoomID) ON DELETE CASCADE,

SenderID INT REFERENCES Users(UserID),

Message TEXT NOT NULL,

SentAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

IsRead BOOLEAN DEFAULT FALSE

);

This schema provides a strong base for managing all your **user roles, listings, vehicle services**, and now **real-time communication** via chat rooms.

Would you like a sample data insert script or ER diagram next?

4o

**You said:**

can i get Like entities Schema and MongoDb Schema rather than SQL

**ChatGPT said:**

Absolutely! Here's your schema converted into a **MongoDB-style design**, using **collections and document structure** that fit NoSQL best practices. I'll also break it into **logical entities** (like User, Vehicle, Listing, Service, Chat, etc.) for clarity.

**🚗 MongoDB Entity-Based Schema Design**

**🔹 1. Users Collection**

json

CopyEdit

{

\_id: ObjectId,

name: String,

email: String,

phoneNumber: String,

passwordHash: String,

role: "CarOwner" | "Buyer" | "ServiceProvider" | "Admin",

profileImage: String,

verified: Boolean,

createdAt: ISODate

}

**🔹 2. Vehicles Collection**

json

CopyEdit

{

\_id: ObjectId,

ownerId: ObjectId, // Ref to Users

registrationNumber: String,

make: String,

model: String,

year: Number,

fuelType: String,

transmission: String,

kilometersDriven: Number,

location: String,

description: String,

status: "Available" | "Sold",

images: [String], // URLs

createdAt: ISODate

}

**🔹 3. Listings Collection**

json

CopyEdit

{

\_id: ObjectId,

vehicleId: ObjectId, // Ref to Vehicles

listedBy: ObjectId, // Ref to Users

price: Number,

listingDate: ISODate,

isActive: Boolean

}

**🔹 4. ServiceProviders Collection**

json

CopyEdit

{

\_id: ObjectId,

userId: ObjectId, // Ref to Users

businessName: String,

address: String,

verified: Boolean,

rating: Number,

services: [

{

serviceId: ObjectId,

serviceName: String,

description: String,

price: Number

}

]

}

**🔹 5. Appointments Collection**

json

CopyEdit

{

\_id: ObjectId,

vehicleId: ObjectId, // Ref to Vehicles

providerId: ObjectId, // Ref to ServiceProviders

userId: ObjectId, // Ref to Users

serviceId: ObjectId, // Ref to embedded service

scheduledDateTime: ISODate,

status: "Booked" | "Completed" | "Cancelled"

}

**🔹 6. ServiceHistory Collection**

json

CopyEdit

{

\_id: ObjectId,

vehicleId: ObjectId, // Ref to Vehicles

serviceDate: Date,

description: String,

document: {

documentType: "RC" | "PUC" | "Insurance",

fileURL: String,

uploadedAt: ISODate

}

}

**🔹 7. Documents Collection *(if not embedded in ServiceHistory)***

json

CopyEdit

{

\_id: ObjectId,

ownerId: ObjectId, // Ref to Users

documentType: "RC" | "PUC" | "Insurance",

fileURL: String,

uploadDate: ISODate

}

**🔹 8. Reviews Collection**

json

CopyEdit

{

\_id: ObjectId,

reviewerId: ObjectId, // Ref to Users

targetId: ObjectId, // Can be a User or Provider

roleReviewed: "Seller" | "ServiceProvider",

rating: Number,

comments: String,

date: ISODate

}

**🔹 9. AdminActions Collection**

json

CopyEdit

{

\_id: ObjectId,

adminId: ObjectId, // Ref to Users

targetType: "User" | "Listing" | "Review",

targetId: ObjectId,

actionType: "Verify" | "Ban" | "Flag" | "Resolve",

notes: String,

timestamp: ISODate

}

**💬 Chat System (MongoDB Style)**

**🔹 10. ChatRooms Collection**

json

CopyEdit

{

\_id: ObjectId,

type: "BuyerSeller" | "OwnerProvider",

participants: {

buyerId: ObjectId,

sellerId: ObjectId,

ownerId: ObjectId,

providerId: ObjectId

},

listingId: ObjectId, // Optional

appointmentId: ObjectId, // Optional

createdAt: ISODate

}

**🔹 11. ChatMessages Collection**

json

CopyEdit

{

\_id: ObjectId,

chatRoomId: ObjectId, // Ref to ChatRooms

senderId: ObjectId, // Ref to Users

message: String,

sentAt: ISODate,

isRead: Boolean

}

**📌 Notes:**

* MongoDB favors **embedding** when documents are used together frequently (e.g., services inside ServiceProviders).
* For **messages**, it's best to use separate documents in a ChatMessages collection to avoid growing documents too large.
* MongoDB doesn't enforce strict schema like SQL, so validations are usually done at the application level or using schema validation rules via **Mongoose** or MongoDB’s **schema validator**.

Would you like a Mongoose model version of this too?

4o

**You said:**

yes please

**ChatGPT said:**

Perfect! Here's the **Mongoose schema setup** (Node.js + MongoDB ODM) for your application, including **Users, Vehicles, Listings, Services, Appointments, Documents, Reviews, Admin Actions, and Chat**.

I'll break it down per entity and use best practices like timestamps, refs, and validation where necessary.

**📦 Mongoose Models**

**🔹 1. User Model**

js

CopyEdit

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

name: String,

email: { type: String, required: true, unique: true },

phoneNumber: String,

passwordHash: { type: String, required: true },

role: { type: String, enum: ['CarOwner', 'Buyer', 'ServiceProvider', 'Admin'], required: true },

profileImage: String,

verified: { type: Boolean, default: false }

}, { timestamps: true });

module.exports = mongoose.model('User', userSchema);

**🔹 2. Vehicle Model**

js

CopyEdit

const vehicleSchema = new mongoose.Schema({

ownerId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },

registrationNumber: { type: String, unique: true, required: true },

make: String,

model: String,

year: Number,

fuelType: String,

transmission: String,

kilometersDriven: Number,

location: String,

description: String,

status: { type: String, enum: ['Available', 'Sold'], default: 'Available' },

images: [String]

}, { timestamps: true });

module.exports = mongoose.model('Vehicle', vehicleSchema);

**🔹 3. Listing Model**

js

CopyEdit

const listingSchema = new mongoose.Schema({

vehicleId: { type: mongoose.Schema.Types.ObjectId, ref: 'Vehicle', required: true },

listedBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },

price: Number,

isActive: { type: Boolean, default: true }

}, { timestamps: { createdAt: 'listingDate' } });

module.exports = mongoose.model('Listing', listingSchema);

**🔹 4. ServiceProvider Model**

js

CopyEdit

const serviceProviderSchema = new mongoose.Schema({

userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },

businessName: String,

address: String,

verified: { type: Boolean, default: false },

rating: { type: Number, default: 0 },

services: [{

serviceName: String,

description: String,

price: Number

}]

});

module.exports = mongoose.model('ServiceProvider', serviceProviderSchema);

**🔹 5. Appointment Model**

js

CopyEdit

const appointmentSchema = new mongoose.Schema({

vehicleId: { type: mongoose.Schema.Types.ObjectId, ref: 'Vehicle' },

providerId: { type: mongoose.Schema.Types.ObjectId, ref: 'ServiceProvider' },

userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },

serviceId: mongoose.Schema.Types.ObjectId, // Optional, from embedded service

scheduledDateTime: Date,

status: { type: String, enum: ['Booked', 'Completed', 'Cancelled'], default: 'Booked' }

});

module.exports = mongoose.model('Appointment', appointmentSchema);

**🔹 6. ServiceHistory Model**

js

CopyEdit

const serviceHistorySchema = new mongoose.Schema({

vehicleId: { type: mongoose.Schema.Types.ObjectId, ref: 'Vehicle' },

serviceDate: Date,

description: String,

document: {

documentType: { type: String, enum: ['RC', 'PUC', 'Insurance'] },

fileURL: String,

uploadedAt: { type: Date, default: Date.now }

}

});

module.exports = mongoose.model('ServiceHistory', serviceHistorySchema);

**🔹 7. Document Model (Optional if not embedded)**

js

CopyEdit

const documentSchema = new mongoose.Schema({

ownerId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },

documentType: { type: String, enum: ['RC', 'PUC', 'Insurance'] },

fileURL: String,

uploadDate: { type: Date, default: Date.now }

});

module.exports = mongoose.model('Document', documentSchema);

**🔹 8. Review Model**

js

CopyEdit

const reviewSchema = new mongoose.Schema({

reviewerId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },

targetId: mongoose.Schema.Types.ObjectId, // Can be a User or Provider

roleReviewed: { type: String, enum: ['Seller', 'ServiceProvider'] },

rating: { type: Number, min: 1, max: 5 },

comments: String

}, { timestamps: true });

module.exports = mongoose.model('Review', reviewSchema);

**🔹 9. AdminAction Model**

js

CopyEdit

const adminActionSchema = new mongoose.Schema({

adminId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },

targetType: { type: String, enum: ['User', 'Listing', 'Review'] },

targetId: mongoose.Schema.Types.ObjectId,

actionType: { type: String, enum: ['Verify', 'Ban', 'Flag', 'Resolve'] },

notes: String

}, { timestamps: true });

module.exports = mongoose.model('AdminAction', adminActionSchema);

**💬 Chat System**

**🔹 10. ChatRoom Model**

js

CopyEdit

const chatRoomSchema = new mongoose.Schema({

type: { type: String, enum: ['BuyerSeller', 'OwnerProvider'], required: true },

buyerId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },

sellerId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },

ownerId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },

providerId: { type: mongoose.Schema.Types.ObjectId, ref: 'ServiceProvider' },

listingId: { type: mongoose.Schema.Types.ObjectId, ref: 'Listing' },

appointmentId: { type: mongoose.Schema.Types.ObjectId, ref: 'Appointment' }

}, { timestamps: true });

module.exports = mongoose.model('ChatRoom', chatRoomSchema);

**🔹 11. ChatMessage Model**

js

CopyEdit

const chatMessageSchema = new mongoose.Schema({

chatRoomId: { type: mongoose.Schema.Types.ObjectId, ref: 'ChatRoom', required: true },

senderId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },

message: { type: String, required: true },

isRead: { type: Boolean, default: false }

}, { timestamps: { createdAt: 'sentAt' } });

module.exports = mongoose.model('ChatMessage', chatMessageSchema);