To build a **food ordering system**, you’ll need to define several **core objects (entities)** and their **relationships**. Based on common food ordering app architectures (like UberEats, PickMe Food, or Domino’s), here is a practical breakdown:

**🍔 Core Entities (Objects)**

**1. User**

Represents customers and potentially admin roles.

{

"\_id": "ObjectId",

"name": "John Doe",

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

"password": "hashed-password",

"phone": "+94771234567",

"address": "123 Main St, Colombo",

"role": "user", // or "admin"

"createdAt": "Date",

"updatedAt": "Date"

}

**2. Restaurant**

Restaurant owners, menus, location, etc.

{

"\_id": "ObjectId",

"name": "Pizza Hut",

"email": "contact@pizzahut.lk",

"phone": "+94112233444",

"address": "Colombo 03",

"logoUrl": "/images/pizzahut.png",

"categories": ["Pizza", "Fast Food"],

"isOpen": true,

"rating": 4.5

}

**3. MenuItem (FoodItem)**

Represents dishes available from restaurants.

{

"\_id": "ObjectId",

"restaurantId": "ObjectId",

"name": "Cheese Pizza",

"description": "Freshly baked pizza with mozzarella",

"price": 1500,

"imageUrl": "/images/cheese-pizza.png",

"category": "Pizza",

"availability": true

}

**4. Cart**

Temporary container before confirming the order.

{

"\_id": "ObjectId",

"userId": "ObjectId",

"items": [

{

"menuItemId": "ObjectId",

"quantity": 2

}

],

"totalPrice": 3000,

"updatedAt": "Date"

}

**5. Order**

Represents confirmed and paid orders.

{

"\_id": "ObjectId",

"userId": "ObjectId",

"restaurantId": "ObjectId",

"items": [

{

"menuItemId": "ObjectId",

"name": "Cheese Pizza",

"quantity": 2,

"price": 1500

}

],

"totalAmount": 3000,

"status": "pending", // paid, preparing, delivered, cancelled

"paymentMethod": "Card", // or Cash, UPI, etc.

"deliveryAddress": "Custom address here",

"orderedAt": "Date",

"deliveredAt": "Date (optional)"

}

**6. Payment**

For transaction details (optional if handled by third-party gateways).

{

"\_id": "ObjectId",

"orderId": "ObjectId",

"userId": "ObjectId",

"amount": 3000,

"method": "Card",

"status": "Success", // or Failed

"transactionId": "XYZ123456",

"paidAt": "Date"

}

**7. Review / Rating**

Optional feedback system.

{

"\_id": "ObjectId",

"userId": "ObjectId",

"restaurantId": "ObjectId",

"menuItemId": "ObjectId",

"rating": 4,

"comment": "Delicious and fast delivery!",

"createdAt": "Date"

}

**🧠 Relationships Between Entities**

| **Entity** | **Related To** |
| --- | --- |
| User | Cart, Orders, Reviews |
| Restaurant | MenuItems, Orders, Reviews |
| MenuItem | Restaurant, Orders, Reviews |
| Cart | User, MenuItems |
| Order | User, Restaurant, MenuItems |
| Payment | User, Order |

**🛢️ Recommended Database**

**✅ MongoDB (NoSQL) – Great for fast development and flexibility**

* Use **collections** for users, restaurants, menuItems, orders, payments, reviews
* You can embed arrays (like items inside orders)
* Scales well and easy to connect with Node.js (via Mongoose)

**Alternative: Relational (SQL)**

Use **PostgreSQL** or **MySQL** if:

* You need strict relationships
* Need joins for analytics
* Have multiple apps sharing a database

**🔧 Example MongoDB Collections**

users

restaurants

menuItems

orders

payments

reviews

**📦 Optional Collections**

* deliveryDrivers
* notifications
* coupons
* favorites

**🔐 Basic Backend Stack Suggestion**

* **Node.js + Express** (backend)
* **MongoDB + Mongoose**
* **JWT** for auth
* **Bcrypt** for password hashing
* **Cloudinary** or Firebase for food images

Let me know if you want:

* MongoDB schema code (Mongoose models)
* Admin panel database layout
* Architecture for REST API or GraphQL backend
* Frontend structure (React/Next.js)

Would you like me to generate **all model files in Mongoose** for this system?