Capstone Project – Online Food Delivery System

A company like **Swiggy/Zomato** wants to design a database in MongoDB to manage **customers, restaurants, menu items, and orders**.

---

1. Database & Collections Setup

```
use foodDeliveryDB

// Customers
db.customers.insertMany([
  { _id: 1, name: "Rahul Sharma", email: "rahul@example.com", city: "Bangalore" },
  { _id: 2, name: "Priya Singh", email: "priya@example.com", city: "Delhi" },
  { _id: 3, name: "Aman Kumar", email: "aman@example.com", city: "Hyderabad" }
]);

// Restaurants
db.restaurants.insertMany([
  { _id: 101, name: "Spicy Treats", city: "Bangalore", rating: 4.5 },
  { _id: 102, name: "Delhi Biryani House", city: "Delhi", rating: 4.2 },
  { _id: 103, name: "Hyderabad Grill", city: "Hyderabad", rating: 4.7 }
]);

// Menu Items (each linked to restaurant)
db.menu.insertMany([
  { _id: 201, restaurant_id: 101, name: "Paneer Tikka", price: 250 },
  { _id: 202, restaurant_id: 101, name: "Veg Biryani", price: 180 },
  { _id: 203, restaurant_id: 102, name: "Chicken Biryani", price: 300 },
  { _id: 204, restaurant_id: 103, name: "Mutton Biryani", price: 400 },
  { _id: 205, restaurant_id: 103, name: "Butter Naan", price: 50 }
]);

// Orders (linked to customer + menu items array)
db.orders.insertMany([
  {
    _id: 301,
    customer_id: 1,
    items: [ { menu_id: 201, qty: 2 }, { menu_id: 202, qty: 1 } ],
    order_date: ISODate("2025-01-05"),
    status: "Delivered"
  },
  {
    _id: 302,
    customer_id: 2,
    items: [ { menu_id: 203, qty: 1 } ],
    order_date: ISODate("2025-01-06"),
    status: "Delivered"
  },
  {
    _id: 303,
    customer_id: 3,
    items: [ { menu_id: 204, qty: 1 }, { menu_id: 205, qty: 3 } ],
    order_date: ISODate("2025-01-07"),
    status: "Pending"
  }
]);
```

---

2. Capstone Exercises

A. CRUD Operations

1. Insert a new customer in Mumbai.
2. Find all restaurants in Hyderabad.
3. Update the rating of `"Spicy Treats"` to 4.8.
4. Delete a menu item `"Butter Naan"`.

B. Indexing

5. Create a unique index on `customers.email`.
6. Create a compound index on `restaurants.city` and `rating`.
7. Verify indexes with `getIndexes()`.
8. Write a query that benefits from the compound index.
9. Write a query that falls back to **COLLSCAN** (e.g., filter only by `rating`).

C. Aggregation Framework

10. Find the total orders placed by each customer.
11. Calculate the total revenue per restaurant.
12. Find the top 2 most expensive dishes across all restaurants.
13. Find the average price of dishes per restaurant.
14. Count how many pending orders exist per city.
15. Show the highest-rated restaurant in each city.

D. `$lookup` (Joins)

16. Show all orders with **customer name and city** attached.
17. Show all orders with **restaurant name and menu item details** attached.
18. For each customer, list all the dishes they ordered with quantity.
19. Find customers who ordered from `"Hyderabad Grill"`.
20. Show a detailed bill for order `301` (dish name, qty, price, total).

E. Advanced Analytics

21. Find customers who spent more than ₹500 in total.
22. Find the top-spending customer in Bangalore.
23. Find restaurants that earned more than ₹500 revenue.
24. Show the daily revenue (total money collected per day).
25. Show the most popular dish (ordered the most times).

▯ This project covers:

- **CRUD basics** (insert, update, delete, read)
- **Indexing** (unique + compound + COLLSCAN vs IXSCAN demo)
- **Aggregation** (grouping, sorting, averages, sums, top-N)
- **Joins (`$lookup`)** for customer-order-restaurant relationships
- **Business analytics use cases**
