**Full-stack e-Commerce website**

**1. Database First (MongoDB)**

* Design your **ER diagram** and create collections accordingly.
* Set up **relationships (references or embedded documents)** based on your use case.
* Ensure indexing for performance.

**2. Backend Development (Spring Boot)**

* Create **models/entities** matching MongoDB collections.
* Implement **repositories and services** for database operations.
* Set up **JWT authentication (login, register, secure routes).**
* Develop REST **APIs per collection** (e.g., User, Product, Order).
* Test APIs using **Postman** before integrating frontend.

**3. Frontend Development (React)**

* Build **Context and Reducer** for state management.
* Start with **authentication (login/register UI)** and integrate the backend.
* Implement **product listing, cart, and checkout** screens.
* Fetch data **per collection** after completing its backend API.

### User ****Collection Design****

## **🔹 1. When to Use Embedded vs. Referenced Data?**

📌 **Rule of Thumb:**

* **Embed data** when it is small, frequently accessed, and belongs naturally to the parent document (e.g., addresses in User).
* **Reference data** when it is large, shared across multiple documents, or changes frequently (e.g., wishlist, cart, orders, products).

✅ **What to Embed?**

* addresses (since they are tied to a user and usually don't change frequently).
* seller\_info (since it only belongs to sellers and isn’t a separate entity).
* permissions for Admin (small and won’t grow excessively).

❌ **What to Reference?** (@DBRef or just store ObjectIds)

* wishlist, cart, orders should reference their respective collections.
* Storing just the **ObjectIds** instead of using @DBRef is preferred for **better query performance**.

## **🔹 2. Optimized** User **Collection Schema**

### ✅ ****Best Practice JSON Document****

{

"\_id": "user123",

"name": "John Doe",

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

"phone": "+919876543210",

"role": "Consumer", // Can be "Consumer", "Seller", "Admin"

"addresses": [

{

"type": "Home",

"street": "123 Street Name",

"city": "New York",

"state": "NY",

"zip": "10001",

"country": "USA",

"status": "default"

}

],

"wishlist": ["prod123", "prod789"],

"cart": ["prod111", "prod222"],

"orders": ["order001", "order002"],

"seller\_info": null, // Will contain data only for Sellers

"permissions": null, // Only for Admin role

"createdAt": "2025-03-11T10:00:00Z",

"updatedAt": "2025-03-11T10:00:00Z"

}

### ✅ ****Best Practice for a Seller User****

{

"\_id": "user456",

"name": "Alice Smith",

"email": "alice.smith@example.com",

"phone": "+911234567890",

"role": "Seller",

"addresses": [

{

"type": "Business",

"street": "456 Business St",

"city": "Los Angeles",

"state": "CA",

"zip": "90001",

"country": "USA"

}

],

"wishlist": [],

"cart": [],

"orders": [],

"seller\_info": {

"store\_name": "Alice Fashion Hub",

"gst\_number": "GST456XYZ",

"products": ["prod111", "prod222"]

},

"permissions": null,

"createdAt": "2025-03-11T10:00:00Z",

"updatedAt": "2025-03-11T10:00:00Z"

}

### ✅ ****Best Practice for an Admin User****

{

"\_id": "user789",

"name": "Rajesh Kumar",

"email": "rajesh.kumar@ecommerce.com",

"phone": "+919876543210",

"role": "Admin",

"addresses": [], // Admins may not have addresses

"wishlist": [],

"cart": [],

"orders": [],

"seller\_info": null,

"permissions": ["manage\_users", "manage\_orders", "manage\_products", "view\_reports"],

"createdAt": "2025-03-11T10:00:00Z",

"updatedAt": "2025-03-11T10:00:00Z"

}

## **🔹 3. Why This is the Best Practice?**

### ✅ ****1. No**** @DBRef ****for Wishlist, Cart, Orders****

Instead of @DBRef, we store ObjectIds as strings (wishlist, cart, orders), making queries much **faster** and avoiding **extra joins**.

### ✅ ****2. Use Conditional Fields Based on Role****

* seller\_info is **only present** when the role is "Seller".
* permissions only exist for "Admin".
* This reduces unnecessary fields and keeps the document **lightweight**.

### ✅ ****3. Addresses are Embedded****

* Addresses **don’t change often**, so we embed them to avoid unnecessary queries.

### ✅ ****4. Timestamps for Auditing (****createdAt****,**** updatedAt****)****

* Helps in tracking when a user was created or last updated.

## **🔹 4. What About Indexing?**

### ✅ ****Important Indexes for Performance****

1️⃣ **Index on email (Unique)**

* Users are often queried by email during login/registration.

@Indexed(unique = true)

private String email;

2️⃣ **Index on role**

* Useful when filtering users (e.g., fetching all sellers).

@Indexed

private String role;

3️⃣ **Index on createdAt**

* Useful for sorting users by signup date.

@Indexed

private String createdAt;

## **🔹 5. Final Optimized User Model in Spring Boot**

import lombok.Data;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.index.Indexed;

import org.springframework.data.mongodb.core.mapping.Document;

import java.util.List;

@Data

@Document(collection = "users")

public class User {

@Id

private String id;

private String name;

@Indexed(unique = true)

private String email;

private String phone;

@Indexed

private String role; // Consumer, Seller, Admin

private List<Address> addresses;

private List<String> wishlist; // Store ObjectIds of products

private List<String> cart; // Store ObjectIds of products

private List<String> orders; // Store ObjectIds of orders

private SellerInfo sellerInfo; // Only for Sellers

private List<String> permissions; // Only for Admins

@Indexed

private String createdAt;

private String updatedAt;

}

## **🔹 6. Final Thoughts**

✅ **No @DBRef** → We store ObjectIds directly (faster queries).  
✅ **Embed small, static data** (like addresses).  
✅ **Keep dynamic fields role-specific** (like seller\_info, permissions).  
✅ **Use proper indexes** (on email, role, createdAt).  
✅ **Maintain timestamps** (createdAt, updatedAt).

### 🎯 ****TL;DR: Best Practices Summary****

* 📌 **Embed:** addresses, seller\_info, permissions
* 📌 **Reference using ObjectIds (no @DBRef):** wishlist, cart, orders
* 📌 **Indexes:** email (unique), role, createdAt
* 📌 **Role-based fields:** Avoid unnecessary fields for each user type