Build FoodFlow: Online Food Delivery App (Zomato / Swiggy Clone)

◇ Problem Statement

Humein ek **Online Food Delivery System** design karna hai
jiska naam hai **Tomato** (Zomato / Swiggy jaisa).

System:

- User ko restaurants dikhata hai
- Cart me items add karne deta hai
- Order place karne deta hai
- Payment + Notification handle karta hai

◇ Design Goal (Why this design?)

- Code **clean & modular** ho
- Easily **extendable** ho (new payment / new order type)
- **Interview-friendly LLD**
- Real-world system se match kare

◇ Core Functionalities

1. User restaurant search kare (location ke basis par)
2. User restaurant select kare
3. Items cart me add kare
4. Order place kare (Now / Scheduled)
5. Payment kare
6. Notification mile

◇ Non-Functional Requirements

- Design should be **scalable**
- Design should be **modifiable**
- Business logic should be **loosely coupled**
- Code should be **easy to extend**

◇ Interview Tips (Important)

1. **Narrow down the scope**

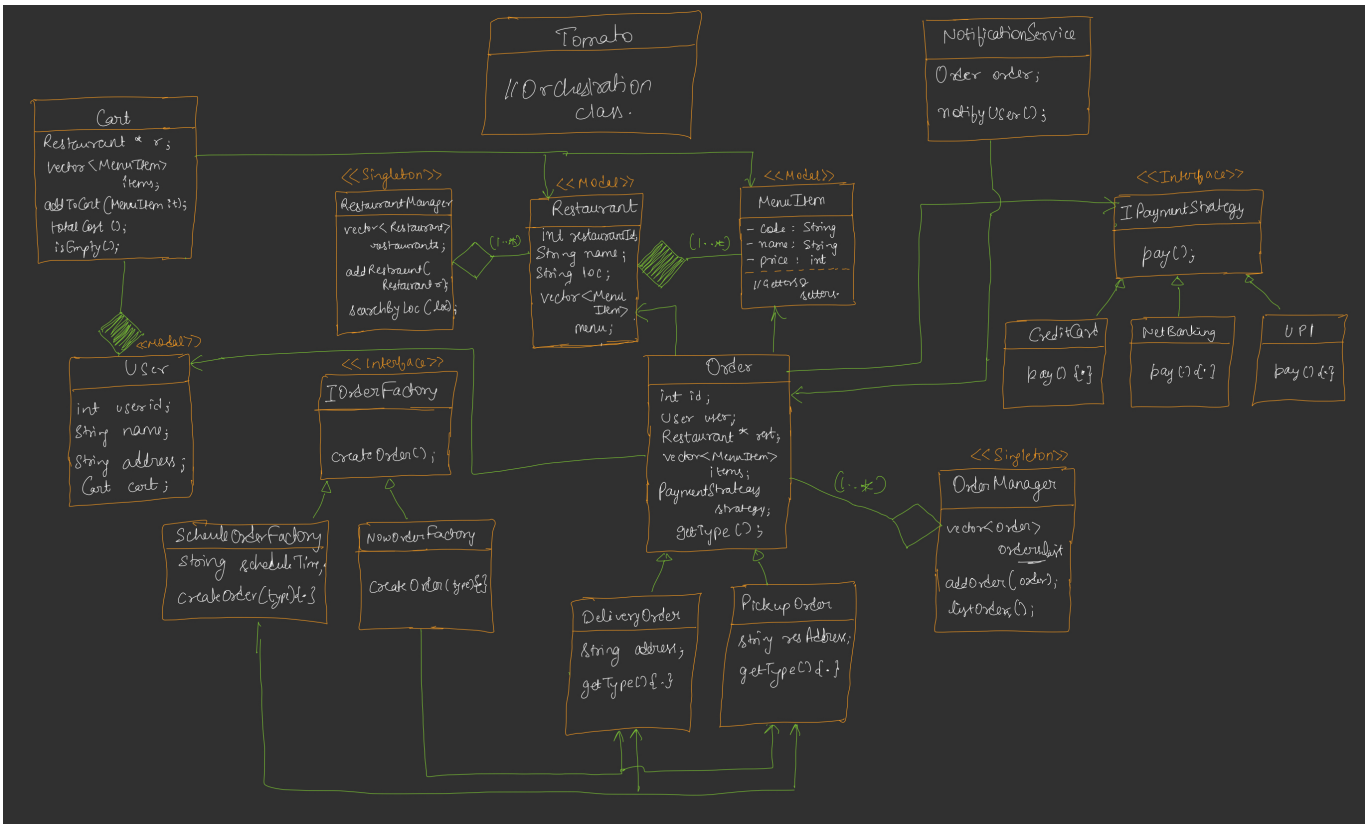
- Ask interviewer:
  "Should I design payment logic or assume it already exists?"

2. **Involve interviewer**
   - Explain your thought process while designing
3. **Start with UML**
   - UML helps to understand:
     - Classes
     - Relationships
     - Responsibilities
4. **Discuss Happy Flow**
   - User → Restaurant → Cart → Order → Payment → Notification

◇ High Level Flow (Samajhne ke liye)

```
User
↓
Cart
↓
Order (Delivery / Pickup)
↓
Payment Strategy
↓
Order Manager
↓
Notification
```

◇ UML Diagram



◇ Project Folder Structure

```
OnlineFoodOrderingSystem/
│
├── Main.cpp              → Happy Flow execution
├── TomatoApp.h           → Facade (main controller)
│
├── models/
│   ├── User
│   ├── Restaurant
│   ├── MenuItem
│   ├── Cart
│   ├── Order (abstract)
│   ├── DeliveryOrder
│   ├── PickupOrder
│
├── managers/
│   ├── RestaurantManager (Singleton)
│   ├── OrderManager      (Singleton)
│
├── strategies/
│   ├── PaymentStrategy (interface)
│   ├── UpiPaymentStrategy
│   ├── CreditCardPaymentStrategy
│
├── factories/
│   ├── OrderFactory (interface)
│   ├── NowOrderFactory
```

```
│   ├── ScheduledOrderFactory
│
├── services/
│   └── NotificationService
│
├── utils/
│   └── TimeUtils
```

◇ Core Classes & Responsibilities (DETAIL)

---

1 User (Model)

Role:

User app ka **customer** hai.

Responsibilities:

- User ki identity represent karna
- User ka cart own karna

Attributes:

- userId
- name
- address
- Cart cart

Design Concept:

Composition

```
User ──owns──► Cart
```

📌 User delete → Cart delete

🗣️ Interview line:

> User aur Cart ke beech composition relationship hai kyunki cart ka lifecycle user pe depend karta hai.

---

2 Restaurant (Model)

Role:

Restaurant seller side entity hai.

Responsibilities:

- Restaurant details hold karna
- Menu manage karna

Attributes:

- restaurantId
- name
- location
- menu (List)

📌 SRP follow hota hai (Restaurant order process nahi karta)

---

3 MenuItem (Model)

Role:

Single food item represent karta hai.

Attributes:

- code
- name
- price
- category (future use)

Design:

Pure **data class** (encapsulation)

---

4 Cart (Model)

Role:

Temporary container for selected items.

Responsibilities:

- addItem()
- getTotalCost()
- isEmpty()

Important Business Rule:

👉 **One Cart = One Restaurant**

📌 Complexity avoid karne ke liye.

5 Order (Abstract Class)

Role:

Order ek **completed transaction** hai.

Why Abstract?

Because order ke types ho sakte hain.

Attributes:

- orderId
- User
- Restaurant
- items
- PaymentStrategy
- total
- scheduledTime

Methods:

- processPayment()
- getType() (pure virtual)

🗣️ Interview line:

> Order class runtime polymorphism enable karti hai.

◇ Order Types (Inheritance)

DeliveryOrder

- extra: userAddress

PickupOrder

- extra: restaurantAddress

```
Order
 ├── DeliveryOrder
 └── PickupOrder
```

📌 IS-A relationship

---

6 PaymentStrategy (Strategy Pattern)

Problem:

Multiple payment options:

- UPI
- Card
- Net Banking

Solution:

Strategy Pattern

```
PaymentStrategy → pay()
```

Implementations:

- UpiPaymentStrategy
- CreditCardPaymentStrategy

🗣️ Interview line:

> Strategy Pattern payment logic ko loosely coupled aur runtime-switchable banata hai.

---

7 OrderFactory (Factory Pattern)

Problem:

Order type runtime par decide hota hai:

- Delivery / Pickup
- Now / Scheduled

Solution:

Factory Pattern

Factories:

- NowOrderFactory
- ScheduledOrderFactory

🗣️ Interview line:

> Factory Pattern object creation logic ko encapsulate karta hai.

---

8 RestaurantManager (Singleton)

Role:

- All restaurants ko manage karta hai

Why Singleton?

- Ek hi restaurant registry chahiye

🗣️ Interview line:

> RestaurantManager singleton hai kyunki system me restaurant ka single source of truth hona chahiye.

9️⃣ OrderManager (Singleton)

Role:

- All placed orders ko track karta hai

🗣️ Interview line:

> OrderManager centralized order lifecycle handle karta hai.

🔟 NotificationService

Role:

- Order success ke baad user ko notify karta hai

Design:

- Stateless
- Static method

📌 Static method ko object se call nahi karna chahiye.

🔷 TomatoApp (Facade Pattern)

Role:

- System ka **single entry point**
- Internals hide karta hai

Provides:

- searchRestaurants()
- addToCart()
- checkoutNow()
- checkoutScheduled()
- payForOrder()

🗣️ Interview line:

> TomatoApp Facade Pattern follow karta hai jo complex subsystem ko simple interface deta hai.

◇ Design Patterns Used

Pattern	Where	Why
Facade	TomatoApp	Simplify usage
Factory	OrderFactory	Encapsulate creation
Strategy	PaymentStrategy	Flexible payment
Singleton	Managers	Single source
Inheritance	Order types	Code reuse
Composition	User → Cart	Ownership

◇ End-to-End Happy Flow

1. User app open karta hai
2. Restaurant search karta hai
3. Restaurant select karta hai
4. Items cart me add karta hai
5. Checkout karta hai
6. Factory order create karti hai
7. Strategy payment process karti hai
8. OrderManager order save karta hai
9. NotificationService notify karti hai

◇ Why This Design Is Interview-Ready

- SOLID principles follow
- No God class
- Easily extendable
- Real-world mapping

🧠 Interview Gold Opening Line

> “I have designed a food delivery system using Facade, Factory, Strategy and Singleton patterns. The system is modular, scalable and close to real-world applications like Zomato.”