Software Requirements Specification (SRS)

Project Title: Food Delivery Platform (Microservices)

# 1. Introduction

1.1 Purpose  
The purpose of this project is to build a mini Food Delivery Platform using microservices. This SRS provides requirements, data models, API contracts, and interaction diagrams to help you implement the project step-by-step.

1.2 Scope  
The system allows users to browse restaurants and menus, place orders, and track delivery. Each core capability is implemented as an independent service that communicates over REST.

# 2. System Overview

Core microservices:

• User Service: Manage users (customers, delivery partners, restaurants).  
• Restaurant Service: Manage restaurants and menu items.  
• Order Service: Create and manage orders and order lifecycle.  
• Delivery Service: Assign delivery partners and track delivery.  
• Notification Service: Send notifications on order updates.  
• API Gateway: Single entry point for clients; routes requests to services.  
• Service Registry & Config: Centralized discovery and configuration.

# 3. Functional Requirements

User Service:  
• Register and update users.  
• Retrieve user profile.  
• List delivery partners (for assignment).

Restaurant Service:  
• Add/update restaurants and menu items.  
• Check item availability.  
• Retrieve menus by restaurant.

Order Service:  
• Place new order with items and address.  
• Validate availability with Restaurant Service.  
• Manage order states (Pending → Accepted → Preparing → OutForDelivery → Delivered → Cancelled).  
• Request delivery assignment.

Delivery Service:  
• Assign delivery partner to an order.  
• Update delivery status and location.  
• Provide delivery history.

Notification Service:  
• Send order confirmations and status updates.  
• Deliver notifications via console/email (simplified).

# 4. Non-Functional Requirements

• Each service must have its own database.  
• Use REST + Feign for synchronous inter-service calls.  
• Use a service registry for discovery and a gateway for routing.  
• Basic resilience: timeouts and retries for remote calls.  
• Containerizable (Docker) for deployment.

# 5. Technologies

Spring Boot, Spring Cloud (Feign, Gateway, Eureka), Spring Data JPA, MySQL/Postgres, Docker, Resilience4j (optional).

# 6. Data Models (examples)

User:  
• id: Long  
• name: String  
• email: String  
• role: String (CUSTOMER/RESTAURANT/DELIVERY)  
• address: String

Restaurant:  
• id: Long  
• name: String  
• address: String  
• rating: Float

MenuItem:  
• id: Long  
• restaurantId: Long  
• name: String  
• price: BigDecimal  
• available: boolean

Order:  
• id: Long  
• userId: Long  
• restaurantId: Long  
• totalAmount: BigDecimal  
• status: String  
• createdAt: Timestamp

OrderItem:  
• id: Long  
• orderId: Long  
• menuItemId: Long  
• quantity: int  
• price: BigDecimal

DeliveryAssignment:  
• id: Long  
• orderId: Long  
• deliveryPartnerId: Long  
• status: String  
• assignedAt: Timestamp

# 7. Sample APIs

User Service:  
POST /api/users  
GET /api/users/{id}

Restaurant Service:  
GET /api/restaurants  
GET /api/restaurants/{id}/menu  
POST /api/restaurants/{id}/menu

Order Service:  
POST /api/orders (place order)  
GET /api/orders/{id}  
GET /api/users/{id}/orders

Delivery Service:  
POST /api/delivery/assign  
PUT /api/delivery/{id}/status

Notification Service:  
POST /api/notifications

# 8. Example: Place Order Request

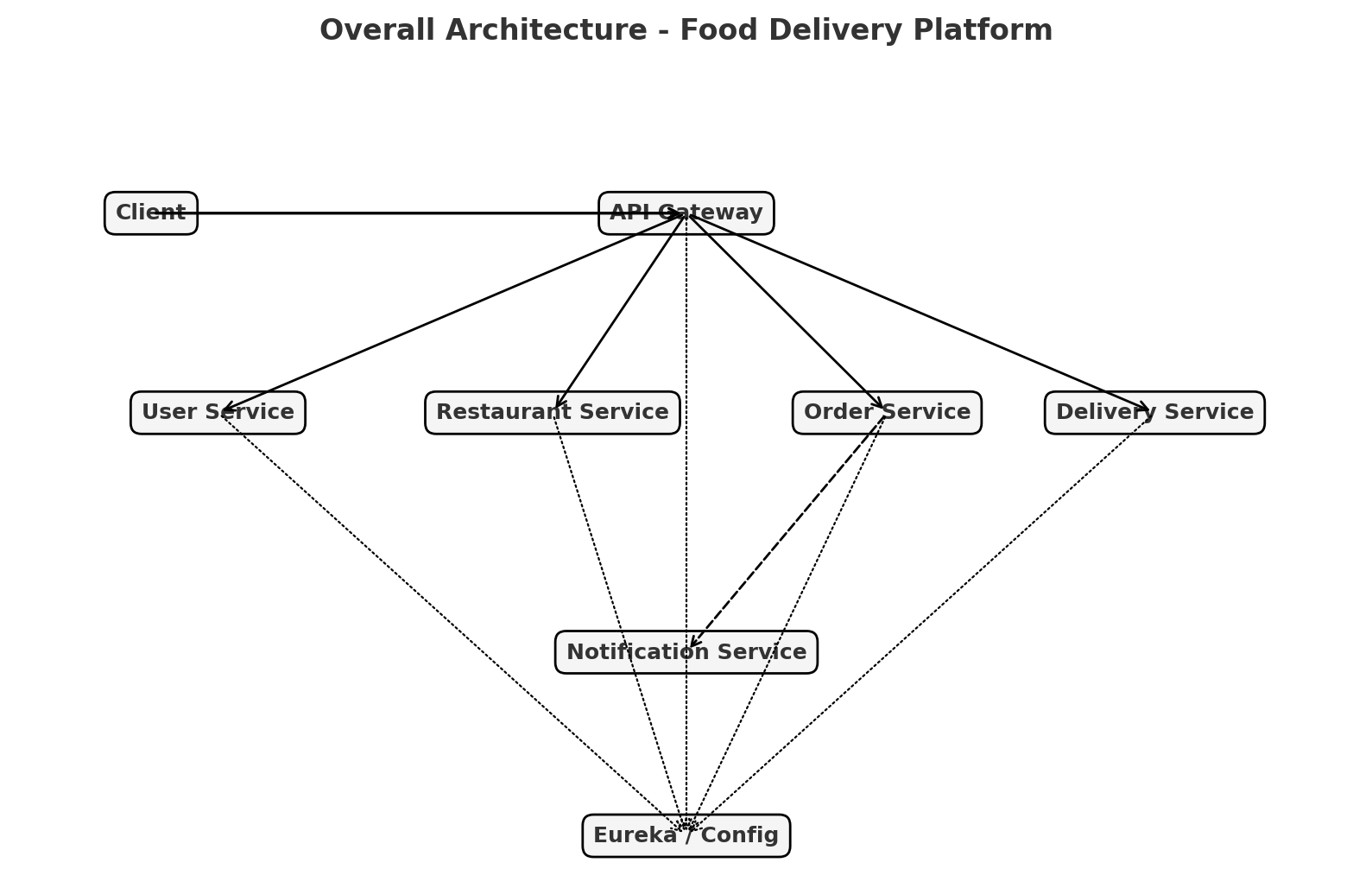
POST /api/orders  
  
Request JSON:  
{  
 "userId": 1,  
 "restaurantId": 10,  
 "items": [ { "menuItemId": 100, "quantity": 2 }, { "menuItemId": 101, "quantity": 1 } ],  
 "deliveryAddress": "123 Main St, City"  
}

# 9. Interaction Sequences

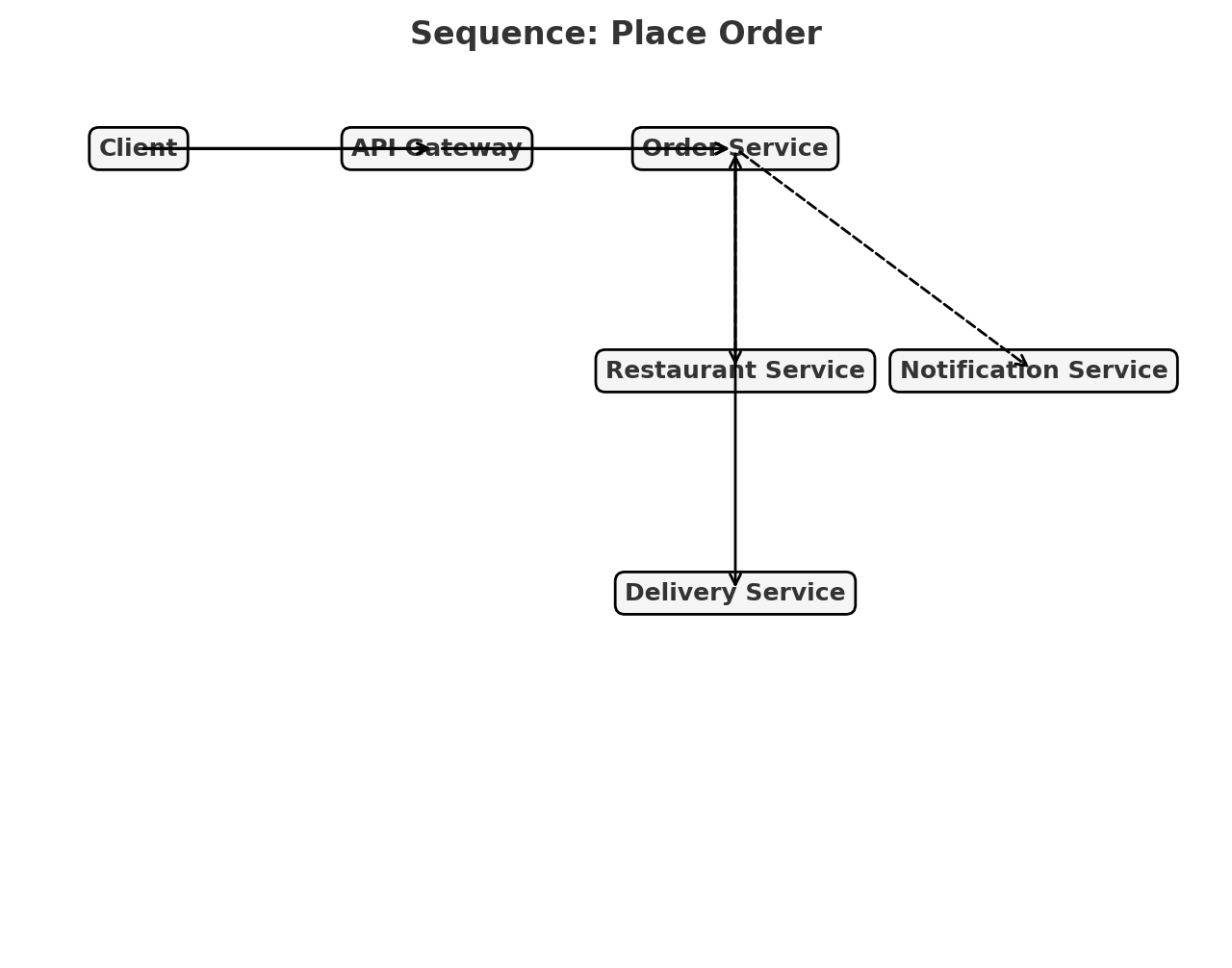
Included diagrams illustrate: overall architecture, place-order sequence, and delivery status updates.

# 10. Diagrams

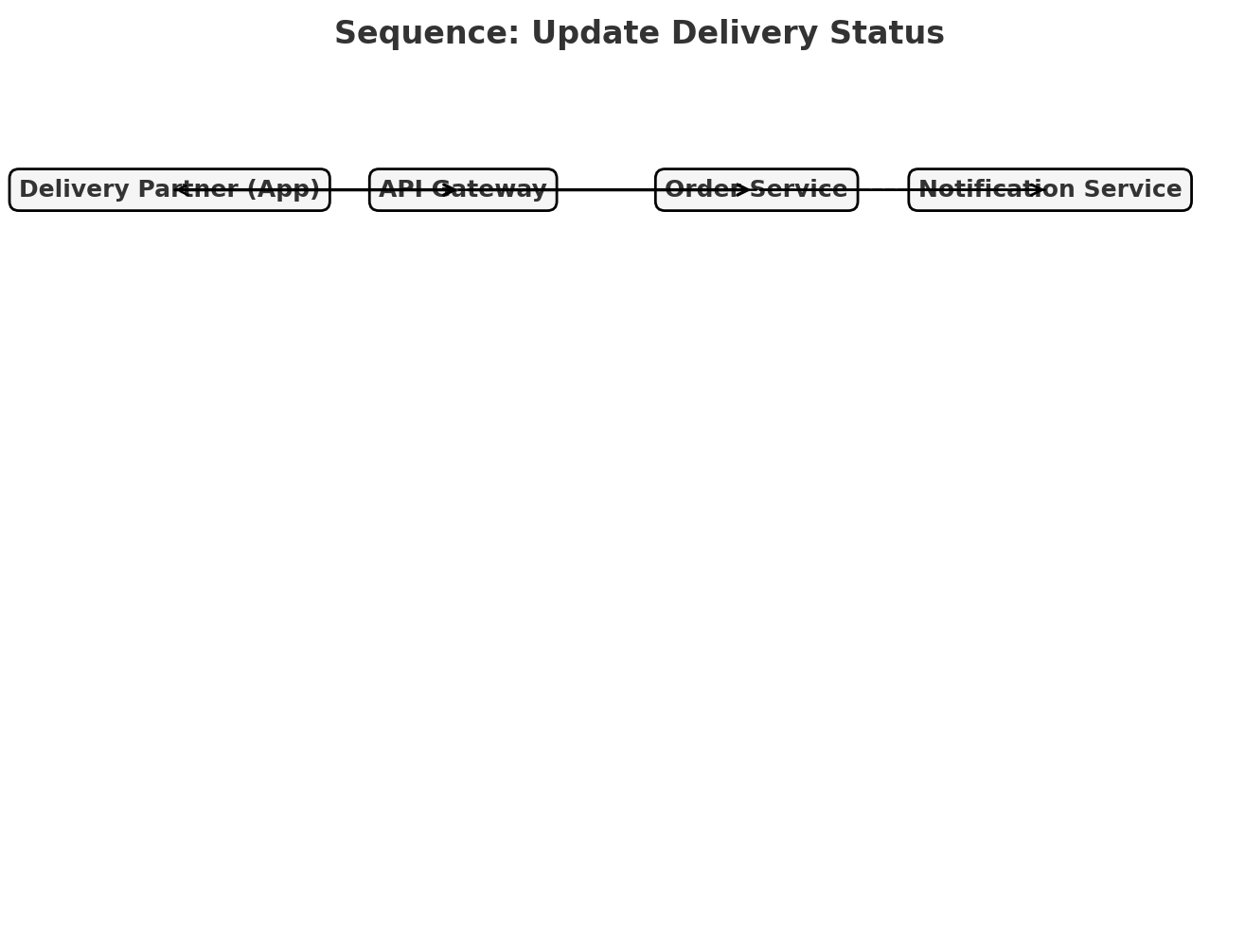
Overall Architecture:



Place Order Sequence:



Update Delivery Status Sequence:



# 11. Deployment & Testing Notes

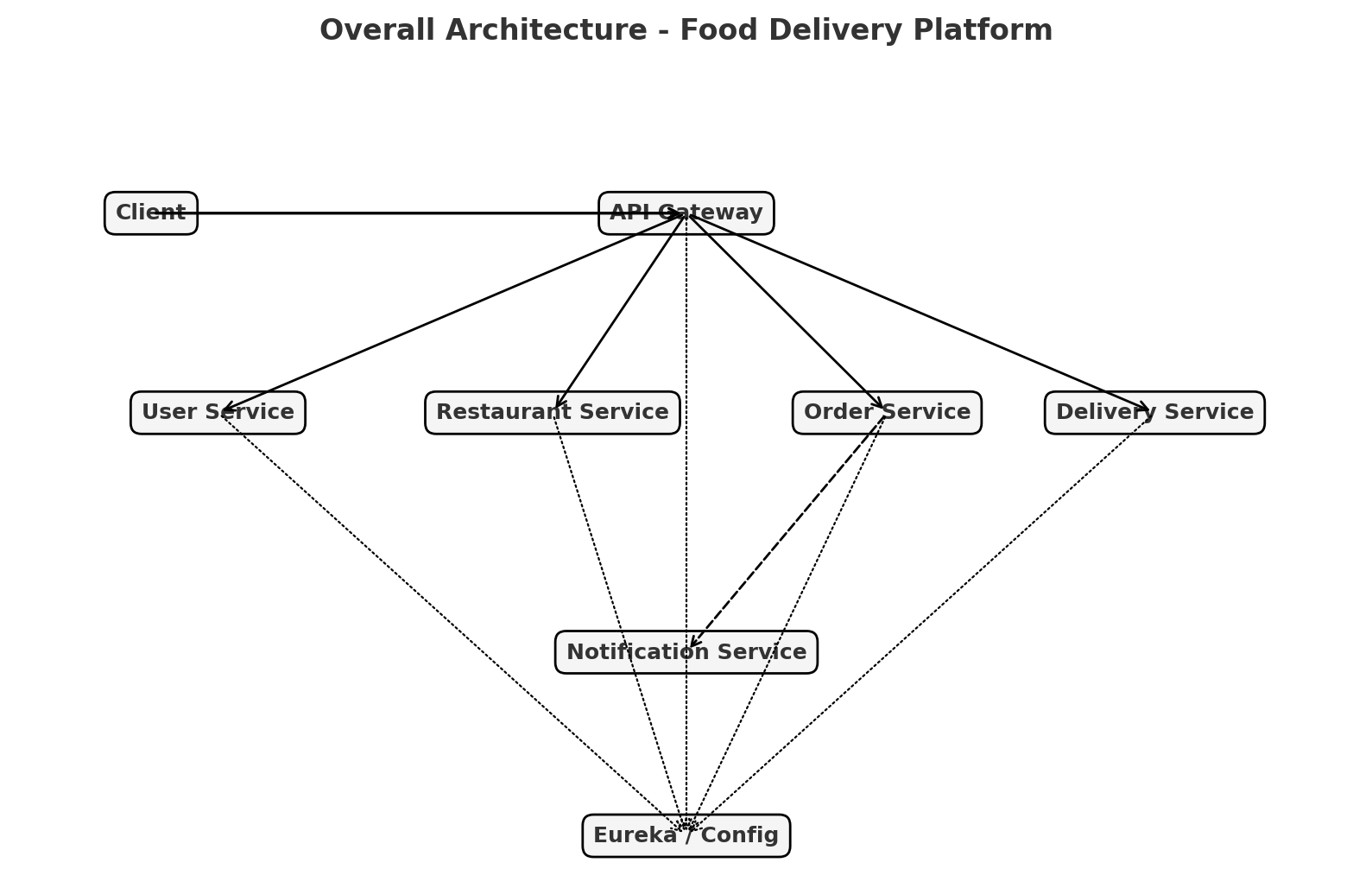
• Start Eureka server and Config server first.  
• Run services (User, Restaurant, Order, Delivery, Notification) with their own DBs.  
• Use Postman to call API Gateway endpoints.  
• Basic unit & integration tests recommended per service.

# 12. Next Steps (suggested)

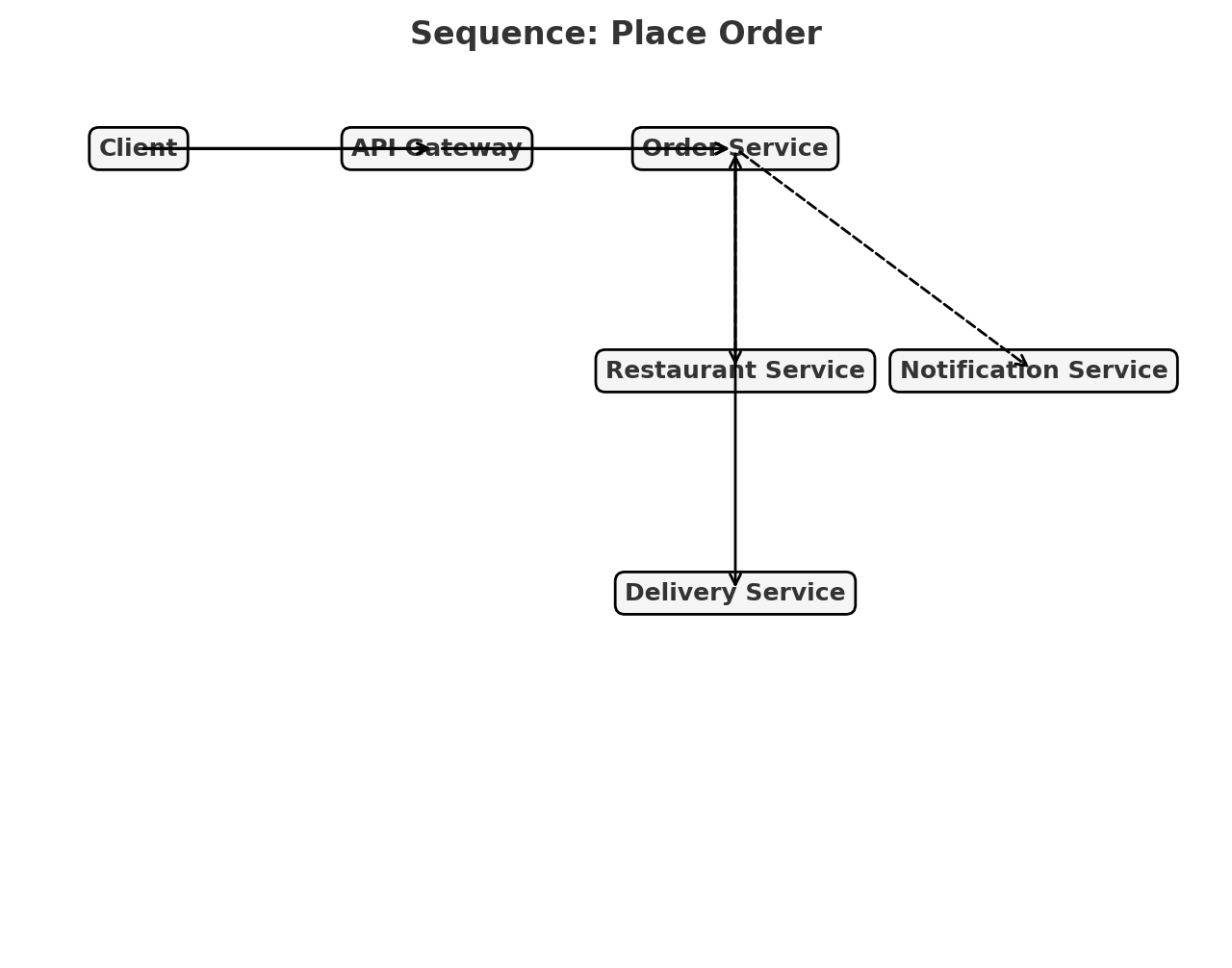
• Implement services one by one starting with User and Restaurant.  
• Implement Order service and test end-to-end via API Gateway.  
• Add resilience and monitoring later.

# System Diagrams

## Overall Architecture



## Place Order Flow



## Update Status Flow

