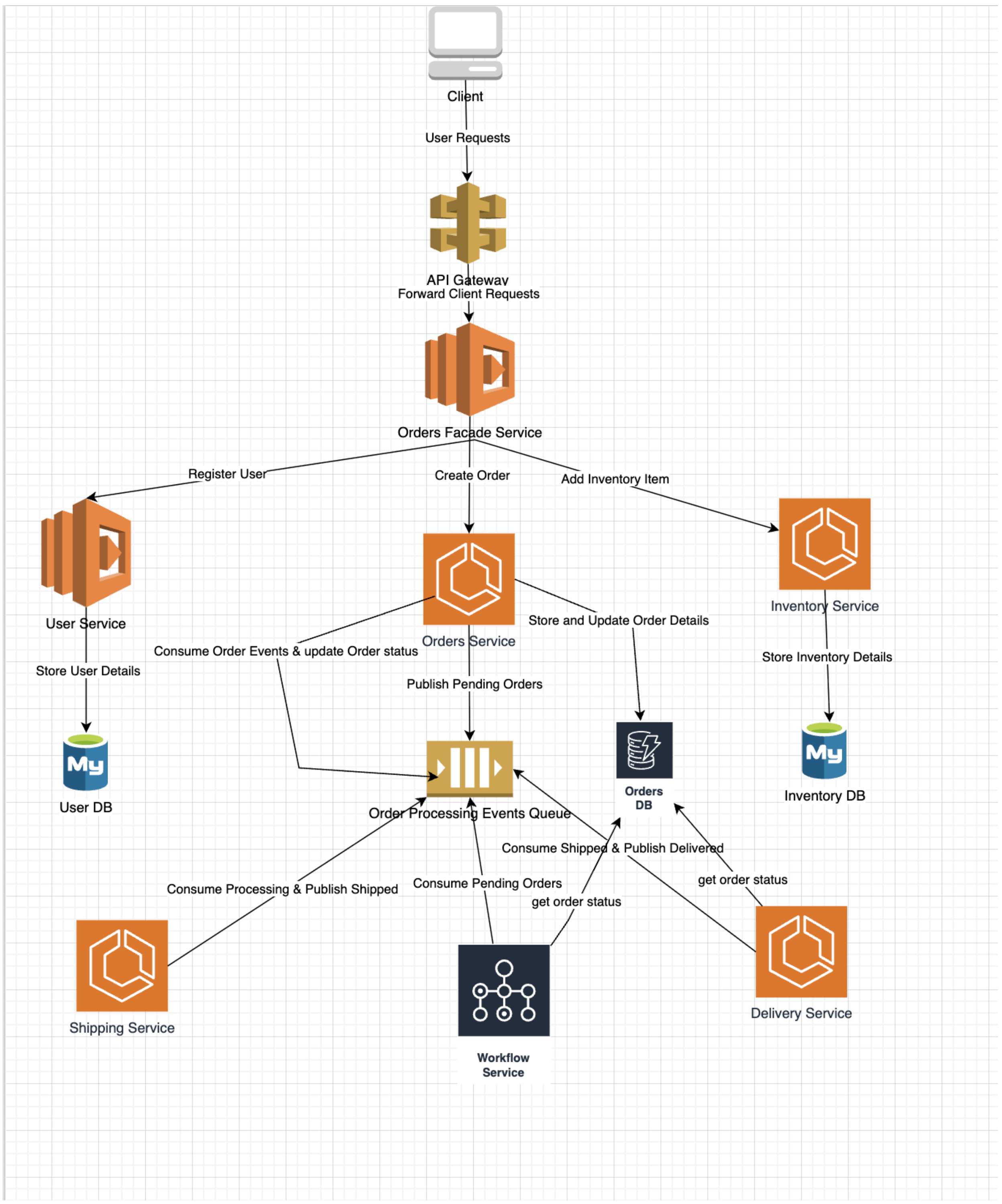
Overview

The **Order Processing System** is a backend microservices-driven application simulating an e-commerce platform. It supports user registration, product inventory, and complete order lifecycle management using event-driven architecture.


Client
User Requests
API Gateway
Forward Client Requests
Orders Facade Service
Register User
Create Order
Add Inventory Item
User Service
Orders Service
Inventory Service
Store and Update Order Details
Consume Order Events & update Order status
Store User Details
Publish Pending Orders
Store Inventory Details
User DB
Orders DB
Inventory DB
Order Processing Events Queue
Consume Shipped & Publish Delivered
get order status
Consume Processing & Publish Shipped
Consume Pending Orders
get order status
Shipping Service
Workflow Service
Delivery Service

Component	Description
API Gateway	Entry point for client requests. Forwards all API calls to the Orders Facade Service.
Orders Facade Service	Unified gateway for handling user, order, and inventory operations. Applies the **Facade Pattern**.
User Service	Registers and stores user data in a dedicated user DB.
Inventory Service	Manages inventory items, product details, and stores them in Inventory DB.
Orders Service	Handles core order creation, persistence, and status transitions. Stores orders in Orders DB.
Workflow Service	Background scheduler that picks `PENDING` orders every 5 minutes and transitions them to `PROCESSING`.
Shipping Service	Listens for `ORDER_PROCESSING` events and moves orders to `SHIPPED`.
Delivery Service	Listens for `ORDER_SHIPPED` events and moves orders to `DELIVERED`.
KafkaBroker (Simulated)	In-memory event queue using pub-sub model to simulate asynchronous communication between services.

🧭 3. User Flows

🧾 1. User Registration

- The user sends a registration request to the system via the **Orders Facade Service**.
- The request is forwarded to the **UserService**, which creates and stores the user in the **UserDB**.

📦 2. Add Inventory

- An admin or system client posts inventory item details.
- The **InventoryService** adds this item to the **InventoryDB**.

🛒 3. Place an Order

- A user places an order via the Facade.
- The **OrderService**:
    - Validates the user exists.
    - Validates inventory and reserves stock.
    - Stores the order with status `PENDING`.
    - Publishes an event: `ORDER_PLACED` to Kafka.

🔁 4. Process Order (via Workflow)

- The **WorkflowService** listens to the `ORDER_PLACED` topic.

- It buffers messages and, every minute, polls and checks:
    - If the order is **still in `PENDING` state** (using `OrderRepository`).
    - If yes, updates it to `PROCESSING` and publishes `ORDER_PROCESSING`.

🚚 5. Ship the Order

- **ShippingService** listens to `ORDER_PROCESSING`.
- It checks the order from DB:
    - If the order is in `PROCESSING`, transitions it to `SHIPPED`.
    - Publishes `ORDER_SHIPPED`.

📬 6. Deliver the Order

- **DeliveryService** listens to `ORDER_SHIPPED`.
- It checks if the order is in `SHIPPED`:
    - If so, it marks it `DELIVERED`.
    - Publishes `ORDER_DELIVERED`.

🔄 7. Cancel an Order

- A user may cancel an order **only if it's still in `PENDING`**.
- The cancel request is routed via Facade to `OrderService`.
- If valid:

    - The order is updated to `CANCELLED`.
    - No further events are published for it.
    - Any further workflow/shipping messages for this order will be skipped due to state checks.

🧼 8. Idempotency & Fault Handling

- Each service (Workflow, Shipping, Delivery) **reads from the DB** before acting.
- This ensures:
    - No state corruption.
    - Stale or invalid transitions (e.g. trying to ship a `CANCELLED` order) are safely skipped.

📑 API Contracts – Order Processing System

👤 User APIs

➕ Register a User

http

CopyEdit

```
POST /api/users

Content-Type: application/json
```

Request Body:

json

CopyEdit

```
{
  "userId": "user11",
  "name": "Sriram",
  "email": "sriram@example.com"
}
```

Response:

json

CopyEdit

```
{
  "message": "User registered successfully",
  "userId": "user11"
}
```

📦 Inventory APIs

➕ Add Inventory Item

h

CopyEdit

```
POST /api/inventory
```

```
Content-Type: application/json
```

Request Body:

json

CopyEdit

```
{
  "itemId": "item1",
  "itemName": "Laptop",
  "availableQty": 10
}
```

Response:

json

CopyEdit

```
{
  "message": "Inventory item added",
  "itemId": "item1"
}
```

📝 Order APIs

🛒 Place a New Order

http

CopyEdit

```
POST /api/orders
Content-Type: application/json
```

Request Body:

json

CopyEdit

```
{
  "order": {
    "orderId": "order-1",
    "userId": "user11",
    "items": [
      {
        "itemId": "item1",
        "quantity": 3
      }
    ],
    "status": "PENDING"
  }
}
```

Response:

json

CopyEdit

```
{
  "message": "Order placed successfully",
  "orderId": "order-1"
}
```

🔍 Get All Orders (Optionally Filter by Status)

http

CopyEdit

```
GET /api/orders?status=PROCESSING
```

Response:

json

CopyEdit

```
[
  {
    "orderId": "order-1",
    "userId": "user11",
    "status": "PROCESSING",
    "items": [...]
  }
]
```

🔍 Get Orders by User

http

CopyEdit

```
GET /api/orders/user/{userId}
```

Example:

http

CopyEdit

```
GET /api/orders/user/user11
```

Response:

json

CopyEdit

```
[
  {
    "orderId": "order-1",
    "status": "DELIVERED",
    ...
  }
]
```

❌ Cancel an Order (Only if in PENDING)

http

CopyEdit

```
POST /api/orders/{orderId}/cancel
```

Example:

http

CopyEdit

```
POST /api/orders/order-1/cancel
```

Response:

json

CopyEdit

```
{
  "message": "Order cancelled successfully",
  "orderId": "order-1",
  "status": "CANCELLED"
}
```

🚫 Error Response (Standard Format)

json

CopyEdit

```
{
  "error": "Order cannot be cancelled. Current status: SHIPPED",

  "status": 400

}
```