Supplier Order Matching System

Problem Statement

In a supply chain, multiple suppliers may offer the same product with different prices, stock availability, and delivery times. The **Supplier Order Matching System** helps businesses find the best supplier for a given order based on predefined criteria (e.g., price, stock availability, location, and delivery time).

Key Features: Supplier registration, product catalog, and optimal supplier selection logic.

Key Requirements

1. Functional Requirements

Supplier Management:

- Suppliers should be able to register and provide details like name, location, product catalog, stock levels, and pricing.
- API to fetch available suppliers for a given product.

Product Catalog Management:

- Suppliers can add, update, and delete products.
- API to retrieve a supplier's product list with stock levels and prices.

Order Placement & Matching:

- Customers (or businesses) can place orders for a product without selecting a specific supplier.
- The system automatically selects the best supplier based on:
    - **Lowest Price** (cheapest available option)
    - **Stock Availability** (supplier with enough stock)
    - **Fastest Delivery Time** (closest supplier or quickest dispatch)
    - **Supplier Rating** (if past performance data is available)
- The customer gets an order confirmation with assigned supplier details.

Order Fulfillment & Updates:

- Suppliers receive order notifications.
- Suppliers confirm or reject orders based on their stock levels.
- Customers receive status updates (Pending, Confirmed, Shipped, Delivered, Canceled).

Order History & Reporting:

- Customers and suppliers can view order history.
- Admins can track supplier performance based on fulfillment rates.

2. Non-Functional Requirements

- **Scalability:** The system should support multiple suppliers and high order volumes.
- **Security:** Use JWT-based authentication for APIs.
- **Performance:** Optimize database queries and use caching for fast supplier lookups.
- **Cloud-Agnostic Deployment:** Deployable on AWS, GCP, or Azure.

System Architecture

Tech Stack:

- **Backend:** Java Spring Boot / Python Django REST / Node Express JS Framework
- **Database:** PostgreSQL (for relational data) or MongoDB (for flexible schema)
- **Cache:** Redis (optional - for supplier availability caching)
- **Queue System:** RabbitMQ or Kafka (for order notifications)
- **API Authentication:** JWT-based auth

Data Model

Order Matching Criteria:

1. Check suppliers with **available stock** for the requested product.
2. Rank suppliers based on:
    - **Lowest price** → Prefer cheapest supplier.
    - **Fastest delivery** → If multiple suppliers have the same price, choose the fastest.
    - **Best supplier rating** → If tie, use supplier rating as tiebreaker.
3. Assign the best supplier and notify them.

Supplier Table:

id	name	location	rating	created_at
1	ABC Ltd	NY, USA	4.5	2024-02-13
2	XYZ Corp	CA, USA	4.2	2024-02-13

Product Table:

id	name	category	created_at
1	Laptop	Electronics	2024-02-13

Supplier_Product Table (Stock & Pricing):

id	supplier_id	product_id	price	stock	estimated_delivery_days
1	1	1	1200.00	50	3
2	2	1	1150.00	30	5

Order Table:

id	customer_id	product_id	quantity	status	supplier_id	total_price	created_at
101	201	1	2	Confirmed	2	2300.00	2024-02-13

API Endpoints

1. Supplier Management

- `POST /api/suppliers/` → Register a new supplier
- `GET /api/suppliers/{id}/` → Get supplier details
- `GET /api/suppliers/{id}/products/` → Get supplier's product catalog

2. Product & Pricing Management

- `POST /api/products/` → Add a new product
- `PUT /api/products/{id}/` → Update product details
- `DELETE /api/products/{id}/` → Remove a product

3. Order Placement & Matching

- `POST /api/orders/` → Place an order (automatically matches the best supplier)
- `GET /api/orders/{id}/` → Get order details
- `PUT /api/orders/{id}/status/` → Update order status

4. Order Matching & Supplier Selection

- `GET /api/match-supplier/{product_id}/?quantity=2`
    - Returns the best supplier for the given product and quantity

Sample Request / Response:

1️⃣ Register a Supplier

Endpoint:

```
POST /api/suppliers
```

Request Body (JSON)

```
{
  "name": "ABC Supplies",
  "location": "New York, USA",
  "rating": 4.8,
  "products": [
    { "productId": 101, "price": 50, "stock": 100, "deliveryTime": 2 },
    { "productId": 102, "price": 30, "stock": 50, "deliveryTime": 3 }
  ]
}
```

Response (201 Created)

```
{
  "message": "Supplier registered successfully",
  "supplierId": 1
}
```

2️⃣ Get Supplier Details

Endpoint:

```
GET /api/suppliers/{supplierId}
```

Sample Request:

```
GET /api/suppliers/1
```

Response (200 OK)

```
{
  "supplierId": 1,
  "name": "ABC Supplies",
  "location": "New York, USA",
  "rating": 4.8,
  "products": [
    { "productId": 101, "price": 50, "stock": 100, "deliveryTime": 2 },
    { "productId": 102, "price": 30, "stock": 50, "deliveryTime": 3 }
  ]
}
```

3️⃣ Place a New Order

Endpoint:

```
POST /api/orders
```

Request Body (JSON)

```
{
  "customerName": "John Doe",
  "productId": 101,
  "quantity": 20
}
```

Response (201 Created)

```
{
  "message": "Order placed successfully",
  "orderId": 10,
  "assignedSupplier": {
    "supplierId": 1,
    "name": "ABC Supplies",
    "location": "New York, USA",
    "price": 50,
    "estimatedDeliveryTime": 2
  },
  "status": "Pending"
}
```

4️⃣ Match Best Supplier for an Order

Endpoint:

```
GET /api/match-supplier/{productId}?quantity=20
```

Sample Request:

```
GET /api/match-supplier/101?quantity=20
```

Response (200 OK)

```
{
  "productId": 101,
  "bestSupplier": {
    "supplierId": 1,
    "name": "ABC Supplies",
    "location": "New York, USA",
    "price": 50,
    "stock": 100,
    "estimatedDeliveryTime": 2
  }
}
```

5️⃣ Update Order Status

Endpoint:

```
PUT /api/orders/{orderId}/status
```

Request Body (JSON)

```
{
  "status": "Shipped"
}
```

Response (200 OK)

```
{
  "message": "Order status updated successfully",
  "orderId": 10,
  "newStatus": "Shipped"
}
```

6️⃣ Get All Orders for a Supplier

Endpoint:

```
GET /api/orders?supplierId={supplierId}
```

Sample Request:

```
GET /api/orders?supplierId=1
```

Response (200 OK)

```
[
  {
    "orderId": 10,
    "customerName": "John Doe",
    "productId": 101,
    "quantity": 20,
    "status": "Shipped",
    "assignedSupplier": {
      "supplierId": 1,
      "name": "ABC Supplies",
      "location": "New York, USA"
    }
  }
]
```

7️⃣ Add a New Product

Endpoint:

```
POST /api/products/
```

Request Body (JSON)

```
{
  "supplierId": 1,
  "productName": "Iron Sheets",
  "price": 75.0,
  "stock": 200,
  "deliveryTime": 4
}
```

Response (201 Created)

```
{
  "message": "Product added successfully",
  "productId": 103,
  "supplierId": 1,
  "productName": "Iron Sheets",
  "price": 75.0,
  "stock": 200,
  "deliveryTime": 4
}
```

Error Handling: Supplier / Product / Order Not Found
Response (404 Not Found)

```
{
  "error": "Supplier|Product|Order not found",
  "message": "No supplier|pr found with ID 999"
}
```

Workflow

1. **Supplier Registration:** Suppliers register and list their products with pricing and stock details.
2. **Customer Places an Order:** The customer requests a product, and the system finds the best supplier.
3. **Order Matching Algorithm:**
   - Filters suppliers with available stock.
   - Ranks suppliers based on **price, delivery time, and rating**.
   - Assigns the **best supplier** to fulfill the order.
4. **Supplier Gets Notification:** The selected supplier receives an order request.
5. **Supplier Confirms/Rejects Order:** If confirmed, order status updates to **Confirmed**.
6. **Order Status Updates:** As the supplier processes the order, status changes (Shipped, Delivered).
7. **Customer Gets Notifications:** The system sends updates on the order progress.

Enhancements & Future Scope

- **AI-based Supplier Matching:**
  - Use machine learning to predict supplier reliability based on past data.
- **Dynamic Pricing Strategy:**
  - Implement bidding-based supplier selection.
- **Real-time Tracking:**
  - Integrate with logistics APIs for shipment tracking.
- **Automated Dispute Resolution:**
  - Handle cases where suppliers fail to fulfill orders.
