Project Overview

The *Smart Delivery Route Optimization System* is designed to optimize logistics operations for a delivery company. Its primary goal is to manage orders, assign routes to delivery vehicles, monitor traffic conditions, and keep customers informed of their delivery status—all while being scalable, secure, and resilient.

The system uses a **microservices architecture**, where each service is responsible for a specific function. These services communicate through RESTful APIs, allowing for independent deployment, testing, and scaling. Let's dive into each service and the overall workflow.

Components and Workflow

1. Order Management Service (order-ms)

- **Responsibilities**:
    - Manages delivery orders, including creating, updating, retrieving, and deleting orders.
    - Tracks order details such as customer ID, addresses, priority, package details, and deadlines.
- **Workflow**:
    - When a new order is placed, the service assigns it a unique orderId and stores relevant details in the database.
    - The service responds with the orderId and confirmation of the order status.
    - This data is later used by other services, like Route Optimization, to plan and prioritize deliveries.

2. Route Optimization Service (route-opt-ms)

- **Responsibilities**:
    - Calculates optimized routes for delivery vehicles.
    - Handles route adjustments based on new orders, real-time traffic data, and vehicle availability.
    - Uses a priority queue for high-priority orders and the Traveling Salesman Problem (TSP) approximation to optimize multi-stop routes.
- **Workflow**:
    - When an order is created or updated in order-ms, the Route Optimization Service fetches the order details and current traffic conditions.
    - It uses DSA techniques to find the most efficient route, incorporating intermediate stops if the vehicle has multiple deliveries.
    - The optimized route is returned with an Estimated Time of Arrival (ETA) for each stop.
    - If real-time traffic data suggests delays, the service recalculates the route and updates the ETA.

3. Traffic Service (traffic-ms)

- **Responsibilities**:
    - Provides real-time traffic information, updating the status of specific roads (e.g., congested, closed).
    - Supplies data to the Route Optimization Service to assist with routing decisions.
- **Workflow**:
    - At regular intervals, the Traffic Service fetches data on road conditions from an external API or database.
    - It sends updates on traffic status and disruptions to the Route Optimization Service.
    - These updates help the Route Optimization Service to dynamically adjust routes based on current conditions.

4. Fleet Management Service (fleet-ms)

- **Responsibilities**:
    - Tracks vehicles' locations, capacity, and operational status.
    - Manages vehicle assignments based on route plans, capacity constraints, and order requirements.
- **Workflow**:

- When a new route is generated by the Route Optimization Service, the Fleet Management Service is notified to assign a vehicle.
- It verifies that the vehicle has enough capacity for the order and is operational.
- The service tracks the vehicle's real-time location and communicates with Route Optimization to provide availability updates.
- This service also manages maintenance and fuel levels, notifying the system when a vehicle becomes unavailable.

5. Notification Service (notify-ms)

- **Responsibilities**:
  - Notifies customers and logistics operators about order status and ETAs.
  - Sends real-time alerts for any delivery delays or ETA changes.
- **Workflow**:
  - When an order is dispatched, the Notification Service sends the customer a confirmation with the estimated arrival time.
  - If the ETA changes (due to traffic or rerouting), it sends an updated notification.
  - Notifications are stored with timestamps for reference, and messages can be customized based on the type of notification (e.g., confirmation, delay alert).

System Workflow (End-to-End)

1. **Order Placement**:
   - A customer places an order via an external UI or API call, providing details like delivery address, priority level, and package information.
   - The Order Management Service stores the order details and returns an orderId to confirm the order.
2. **Route Calculation**:
   - The Route Optimization Service receives the new order details.
   - It fetches traffic data from the Traffic Service and uses DSA to calculate the optimal delivery route, accounting for priority level and deadlines.
   - If other stops are assigned to the same vehicle, the service includes them in the route and calculates the optimal sequence (solving TSP approximately).
   - Once the route is calculated, the service sends it to the Fleet Management Service for vehicle assignment.
3. **Vehicle Assignment**:
   - The Fleet Management Service checks available vehicles based on capacity, location, and maintenance status.
   - Once a suitable vehicle is identified, it's assigned to the route, and the vehicle's location is tracked in real time.
   - The service also provides real-time updates to the Route Optimization Service if the vehicle becomes unavailable or is delayed.
4. **Traffic Monitoring and Dynamic Rerouting**:
   - The Traffic Service continually monitors and updates traffic conditions.
   - If a traffic delay affects an ongoing route, the Route Optimization Service recalculates the route and updates the ETA.
   - The updated route and ETA are sent to both the Fleet Management and Notification Services.
5. **Notification Delivery**:
   - The Notification Service sends the initial ETA to the customer when the order is dispatched.
   - If the route or ETA is updated, a new notification is sent to keep the customer informed.

Security, Monitoring, and Deployment

- **Security**: OAuth2 authentication is implemented to secure APIs, allowing only authorized users to access customer data and order management features.
- **Monitoring**: Prometheus is used for monitoring service health and performance metrics, with Grafana providing visual dashboards. This setup helps track service uptime, API response times, and alert for any issues.
- **Deployment**: Each microservice is containerized using Docker, making it portable and easy to deploy. Kubernetes is used to orchestrate containers, ensuring scalability and load balancing.

Developer Checklist

1. **Set up microservices with REST APIs**: Create endpoints for each service following REST principles, ensuring each service is stateless and independent.
2. **Implement Database Models**: Design database schemas for storing orders, traffic data, fleet information, and notifications. Use a SQL database.
3. **Optimize Routing**: Use shortest path and TSP algorithms in Route Optimization Service for efficiency.
4. **Integrate Traffic Data**: Use either mock data or a real traffic API to provide updates.
5. **Build OAuth2 Authentication**: Secure services with OAuth2 to restrict access to order and customer data.
6. **Enable Monitoring**: Integrate Prometheus and Grafana to monitor system health and performance.
7. **Deploy and Test**: Containerize services with Docker, then deploy and test on Kubernetes or a similar orchestration platform.

This overview covers the main system components and flow, providing developers with a structured approach to build each microservice and integrate them into a working logistics optimization solution.