LapakKirim

LapakKirim is a comprehensive delivery service application that offers users multiple shipping provider options, including **JNE, J&T Express, and ID Express**. The platform enables customers to seamlessly compare shipping rates, track deliveries in real time, and select the most suitable courier service based on their needs. It is designed for efficiency, reliability, and user convenience, ensuring a smooth logistics experience from order placement to final delivery.

The system architecture consists of multiple layers and components to ensure smooth operation, scalability, and security.

1. Architectural Overview (Layered Architecture)

The system follows a **4-Tier Architecture**:

1. **Client Layer (Frontend)**
2. **Application Layer (Backend & API)**
3. **Data Layer (Database & Storage)**
4. **Infrastructure Layer (Cloud & Networking)**

2. Detailed System Architecture Diagram (Components & Flow)

1. Client Layer (Frontend)

Users: Customers, Couriers, Admins
Technology: React.js (Web), Flutter (Mobile App)
Functionality:

- Customers can **place orders, make payments, and track deliveries**.
- Couriers can **accept delivery requests and update status**.
- Admins can **manage users, orders, and couriers**.

2. Application Layer (Backend & API)

Technology: Node.js (Express.js) / Django (Python)
Core Services:

1. **User Management Service**
    - Handles authentication (JWT, OAuth 2.0)
    - Manages user roles (Customer, Courier, Admin)
2. **Order Management Service**

- Manages order placement and tracking
- Assigns couriers based on availability

3. **Payment Service**
    - Integrates with Stripe, PayPal, or local payment gateways
    - Handles refunds and transaction statuses
4. **Delivery Tracking Service**
    - Uses GPS and Google Maps API for live tracking
    - Updates order delivery status
5. **Notification Service**
    - Sends email, SMS, and push notifications
    - WebSockets for real-time updates
6. **Admin Dashboard Service**
    - Manages analytics, reports, and system logs

3. Data Layer (Database & Storage)

Technology: MySQL / PostgreSQL (Relational Database) + Redis (Cache)
Tables & Relationships:

- Users (Customer, Courier, Admin)
- Orders (Links to Customers)
- Payments (Links to Orders)
- Deliveries (Links to Orders & Couriers)
- Tracking (Real-time order tracking)

Storage:

- AWS S3 / Google Cloud Storage (for images, invoices)
- Redis (for caching real-time tracking updates)

4. Infrastructure Layer (Cloud & Networking)

Hosting: AWS / Google Cloud / DigitalOcean
Key Components:

1. **Load Balancer** - Distributes traffic across servers
2. **API Gateway** - Manages secure API calls

3. **CI/CD Pipeline** - Automates deployment (Jenkins, GitHub Actions)
4. **Monitoring Tools** - Prometheus, Grafana for performance tracking
5. **Security Layer** - Firewalls, DDoS Protection, SSL/TLS Encryption

3. Workflow of the System

1. **User Registration & Login**
    - Customers sign up via email/phone and create an account.
    - Authentication is managed via JWT or OAuth.
2. **Placing an Order**
    - Customers enter pickup & delivery locations.
    - The system calculates the delivery fee.
    - The customer makes a payment.
3. **Courier Assignment & Tracking**
    - An available courier is assigned to the order.
    - The courier updates the delivery status.
    - The customer receives real-time tracking.
4. **Delivery Completion**
    - Once delivered, the system updates the order status.
    - Payment is processed and the courier gets paid.
5. **Admin Management**
    - Admins monitor and manage orders, users, and reports.

4. Technologies Stack

Layer	**Technology Choices**
Frontend	React.js, Flutter
Backend	Node.js (Express), Django
Database	MySQL, PostgreSQL, Redis (Cache)
Storage	AWS S3, Google Cloud Storage
Payment Gateway	Stripe, PayPal, Midtrans

Layer	Technology Choices
Real-time Tracking	Google Maps API, WebSockets
DevOps	Docker, Kubernetes, Jenkins
Security	JWT Auth, SSL/TLS, Firewalls

5. Security Considerations

- **Authentication & Authorization:** JWT-based authentication, OAuth2 for third-party login.
- **Data Encryption:** AES encryption for sensitive data, HTTPS/TLS for secure API calls.
- **Input Validation:** Protect against SQL Injection & XSS attacks.
- **Logging & Monitoring:** Real-time logging using ELK Stack (Elasticsearch, Logstash, Kibana).

6. Scalability & Performance

- **Horizontal Scaling:** Load balancer distributes traffic to multiple backend servers.
- **Caching Strategy:** Redis/Memcached to store frequently accessed data.
- **Microservices Architecture:** Each service is independent and scalable.

7. Identify the Core Features

A **Delivery of Goods** application typically involves:

- **User Management** (Customers, Admins, Couriers)
- **Order Management** (Order creation, tracking, status updates)
- **Payment Processing**
- **Delivery Assignment & Tracking**
- **Notifications & Reviews**

8. Use Case Diagram

Actors:

- **Customer:** Places orders, tracks deliveries, makes payments
- **Admin:** Manages users, assigns couriers, monitors system
- **Courier:** Accepts deliveries, updates delivery status

Example Use Cases:

- Register/Login

- Place Order
- Assign Delivery
- Track Order
- Make Payment
- Update Order Status

9. Class Diagram

Key entities in the system:

1. **User (id, name, email, role, phone_number, password_hash)**
2. **Customer (id, user_id, address, payment_info)**
3. **Courier (id, user_id, vehicle_type, availability_status)**
4. **Order (id, customer_id, pickup_address, delivery_address, status, total_price)**
5. **Delivery (id, order_id, courier_id, estimated_time, delivery_status, tracking_code)**
6. **Payment (id, order_id, payment_method, transaction_status, timestamp)**

```
class User:
   def __init__(self, id: int, name: str, email: str, phone_number: str, password_hash: str, role: str):
      self.id = id
      self.name = name
      self.email = email
      self.phone_number = phone_number
      self.password_hash = password_hash
      self.role = role  # ('customer', 'courier', 'admin')

class Customer(User):
   def __init__(self, id: int, name: str, email: str, phone_number: str, password_hash: str, address: str,
payment_info: str):
      super().__init__(id, name, email, phone_number, password_hash, "customer")
      self.address = address
      self.payment_info = payment_info

class Courier(User):
   def __init__(self, id: int, name: str, email: str, phone_number: str, password_hash: str,
vehicle_type: str, availability_status: str):
      super().__init__(id, name, email, phone_number, password_hash, "courier")
      self.vehicle_type = vehicle_type
      self.availability_status = availability_status  # ('available', 'busy')

class Order:
   def __init__(self, id: int, customer: Customer, pickup_address: str, delivery_address: str, status: str,
total_price: float):
```

```
        self.id = id
        self.customer = customer
        self.pickup_address = pickup_address
        self.delivery_address = delivery_address
        self.status = status  # ('pending', 'confirmed', 'on_delivery', 'completed', 'cancelled')
        self.total_price = total_price

class Delivery:
    def __init__(self, id: int, order: Order, courier: Courier, estimated_time: datetime, delivery_status:
str, tracking_code: str):
        self.id = id
        self.order = order
        self.courier = courier
        self.estimated_time = estimated_time
        self.delivery_status = delivery_status  # ('assigned', 'in_transit', 'delivered')
        self.tracking_code = tracking_code

class Payment:
    def __init__(self, id: int, order: Order, payment_method: str, transaction_status: str, timestamp:
datetime):
        self.id = id
        self.order = order
        self.payment_method = payment_method  # ('credit_card', 'paypal', 'bank_transfer')
        self.transaction_status = transaction_status  # ('pending', 'successful', 'failed')
        self.timestamp = timestamp
```

10. Sequence Diagram

Order Placement Flow

1. Customer places an order
2. System calculates delivery fee
3. Customer confirms and pays
4. System assigns a courier
5. Courier picks up and updates delivery status
6. Customer tracks order in real-time

5. Database Schema Design

```
sql
CopyEdit
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
```

```
    name VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    phone_number VARCHAR(20),
    password_hash VARCHAR(255),
    role ENUM('customer', 'courier', 'admin')
);

CREATE TABLE customers (
    id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT UNIQUE,
    address TEXT,
    payment_info VARCHAR(255),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);

CREATE TABLE couriers (
    id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT UNIQUE,
    vehicle_type VARCHAR(50),
    availability_status ENUM('available', 'busy'),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);

CREATE TABLE orders (
    id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    pickup_address TEXT,
    delivery_address TEXT,
    status ENUM('pending', 'confirmed', 'on_delivery', 'completed',
'cancelled'),
    total_price DECIMAL(10,2),
    FOREIGN KEY (customer_id) REFERENCES customers(id) ON DELETE CASCADE
);

CREATE TABLE deliveries (
    id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT UNIQUE,
    courier_id INT,
    estimated_time DATETIME,
    delivery_status ENUM('assigned', 'in_transit', 'delivered'),
    tracking_code VARCHAR(50) UNIQUE,
    FOREIGN KEY (order_id) REFERENCES orders(id) ON DELETE CASCADE,
    FOREIGN KEY (courier_id) REFERENCES couriers(id) ON DELETE SET NULL
);

CREATE TABLE payments (
    id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT UNIQUE,
    payment_method ENUM('credit_card', 'paypal', 'bank_transfer'),
    transaction_status ENUM('pending', 'successful', 'failed'),
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (order_id) REFERENCES orders(id) ON DELETE CASCADE
);
```