Use Case: Logistics & Shipment Tracking API

📌 Overview

Develop a **RESTful API for a Logistics & Shipment Tracking System**.

The system allows:

- Customers to create shipments and track deliveries
- Delivery Agents to update shipment status
- Admins to manage hubs, vehicles, and monitor performance

This system is similar to platforms like FedEx, Delhivery, or DHL backend systems.

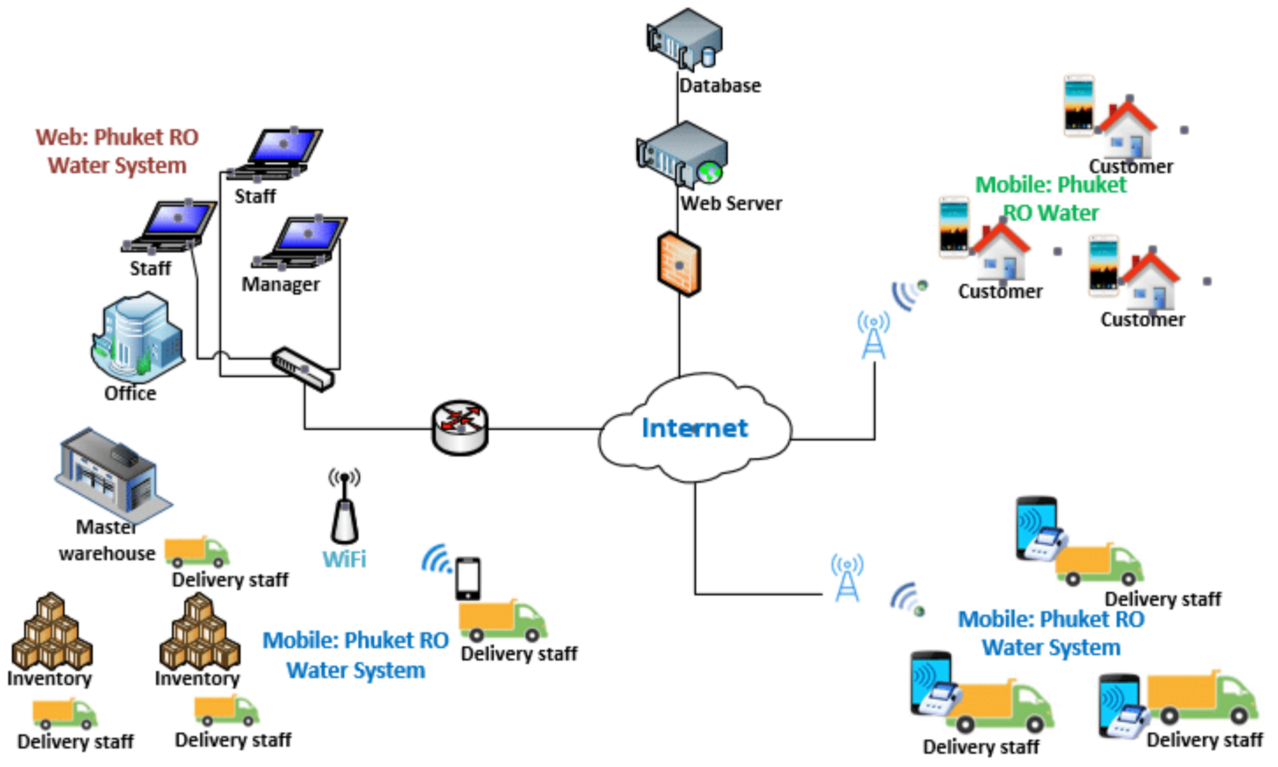
Actors

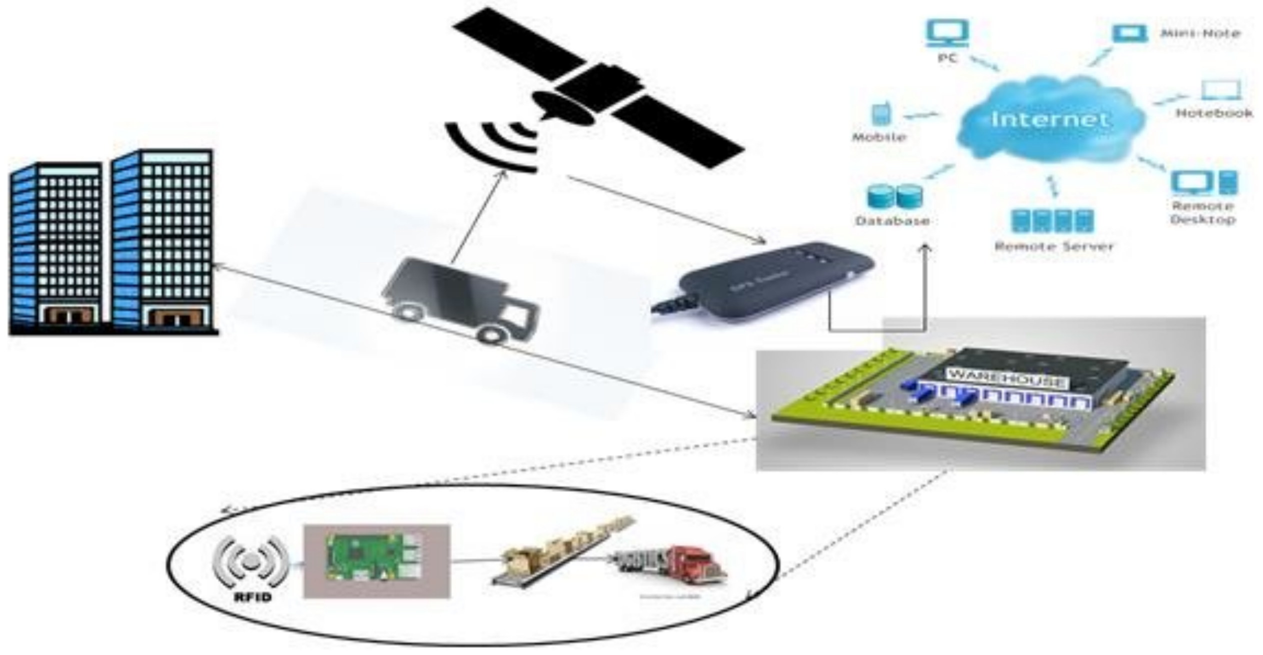
- Customer
- Delivery Agent
- Admin

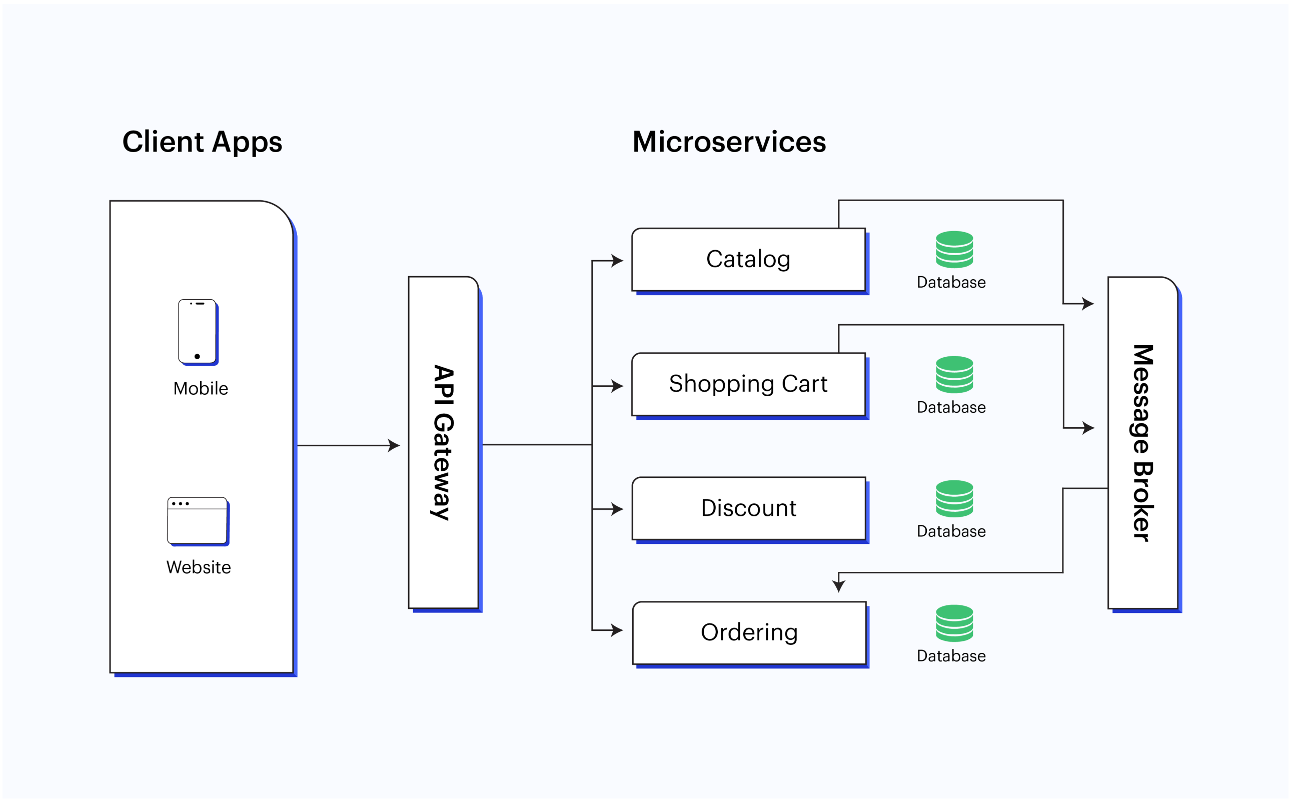
Technology Stack

- FastAPI
- SQLAlchemy
- PostgreSQL
- JWT Authentication
- Docker
- Redis (for tracking cache & real-time status updates)

System Architecture


Database
Web Server
Web: Phuket RO Water System
Staff
Staff
Manager
Office
Internet
Mobile: Phuket RO Water
Customer
Customer
Customer
Master warehouse
Delivery staff
WiFi
Mobile: Phuket RO Water System
Delivery staff
Inventory
Inventory
Delivery staff
Delivery staff
Delivery staff
Mobile: Phuket RO Water System
Delivery staff
Delivery staff


PC
Mini-Note
Mobile
Internet
Notebook
Database
Remote Server
Remote Desktop
WAREHOUSE
RFID



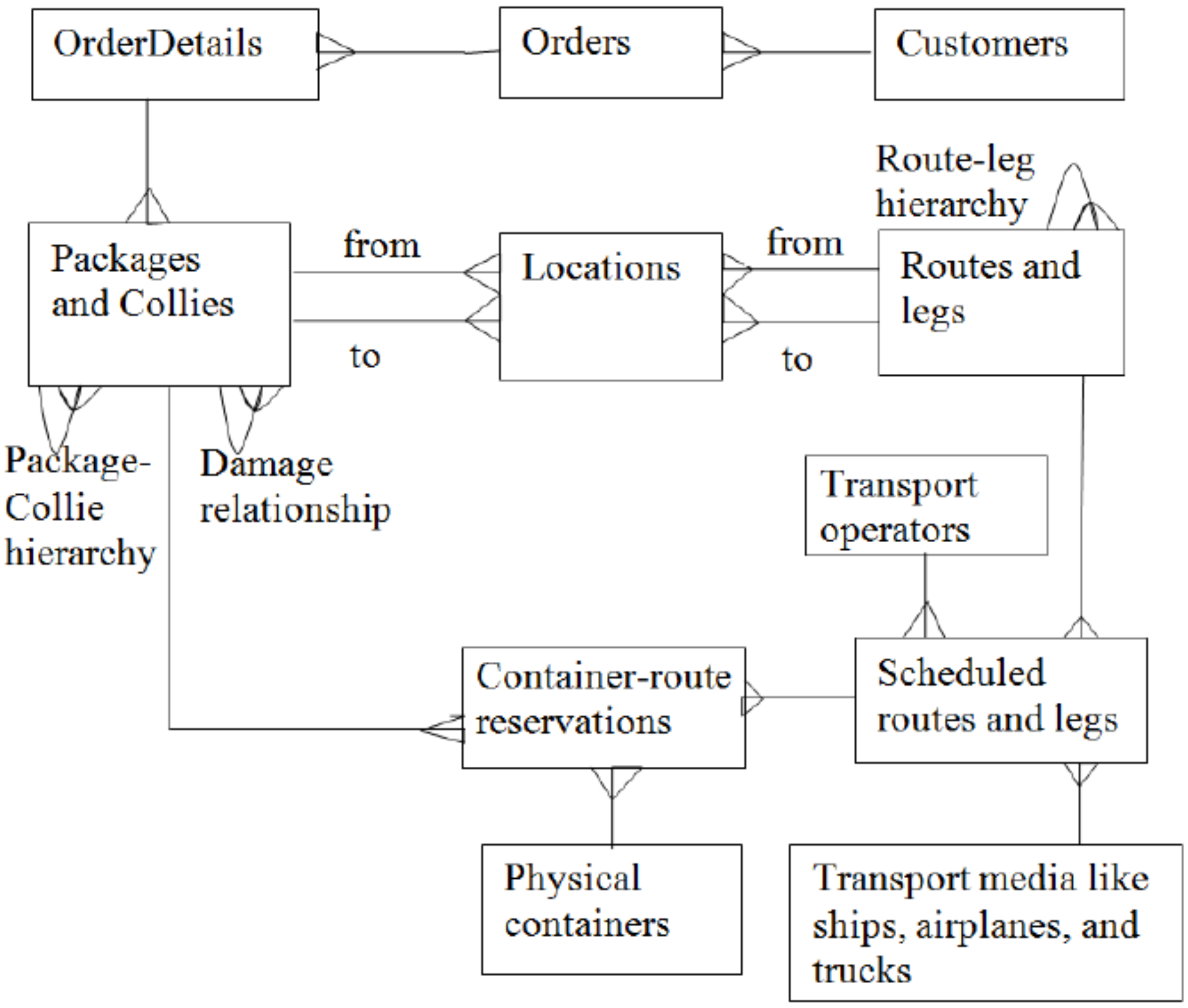
Project Structure

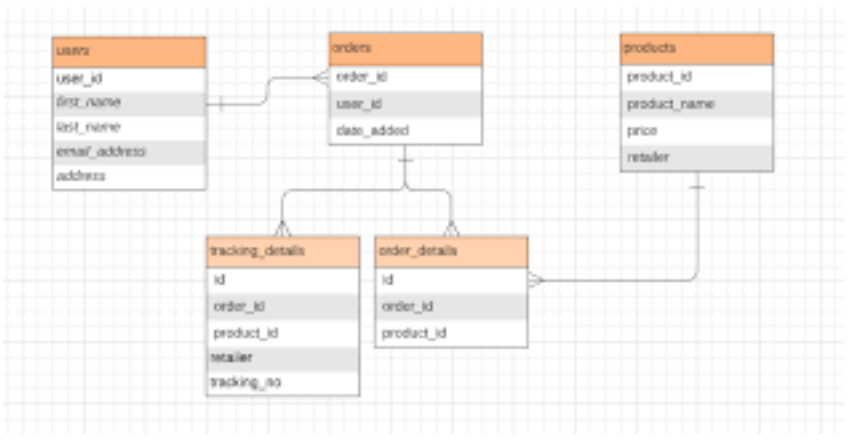
```
logistics-api/
│
├── app/
│   ├── main.py
│
│   ├── core/                        # Core infrastructure
│   │   ├── config.py                 # Environment
settings
│   │   ├── database.py               # Engine,
SessionLocal, Base
│   │   ├── security.py               # JWT, password
hashing
│   │   ├── dependencies.py           # get_db,
get_current_user, role checks
│
│   ├── models/                       # SQLAlchemy ORM
models
│   │   ├── base.py
│   │   ├── user.py                   # Admin, Dispatcher,
Driver, Customer
│   │   ├── shipment.py
│   │   ├── tracking.py
│   │   ├── hub.py
│
│   ├── schemas/                         # Pydantic request/
response models
│   │   ├── auth_schema.py
│   │   ├── user_schema.py
│   │   ├── shipment_schema.py
│   │   ├── tracking_schema.py
│   │   ├── hub_schema.py
│
│   ├── repositories/                    # Data access layer
(DB only)
│   │   ├── user_repository.py
│   │   ├── shipment_repository.py
│   │   ├── tracking_repository.py
│   │   ├── hub_repository.py
│
│   ├── services/                        # Business logic
layer
│   │   ├── auth_service.py
│   │   ├── user_service.py
│   │   ├── shipment_service.py
│   │   ├── tracking_service.py
│   │   ├── hub_service.py
│
│   ├── api/                             # API layer
(Controllers)
│   │   ├── router.py                    # Central router
inclusion
│   │   ├── routes/
│   │   │   ├── auth.py
│   │   │   ├── shipments.py
│   │   │   ├── tracking.py
│   │   │   ├── hubs.py
│   │   │   ├── admin.py
│
│   ├── middleware/                      # Middleware
components
│   │   ├── cors.py
│   │   ├── logging_middleware.py
│   │   ├── rate_limiter.py              # Optional (API
protection)
│
│   ├── exceptions/                      # Centralized error
handling
```

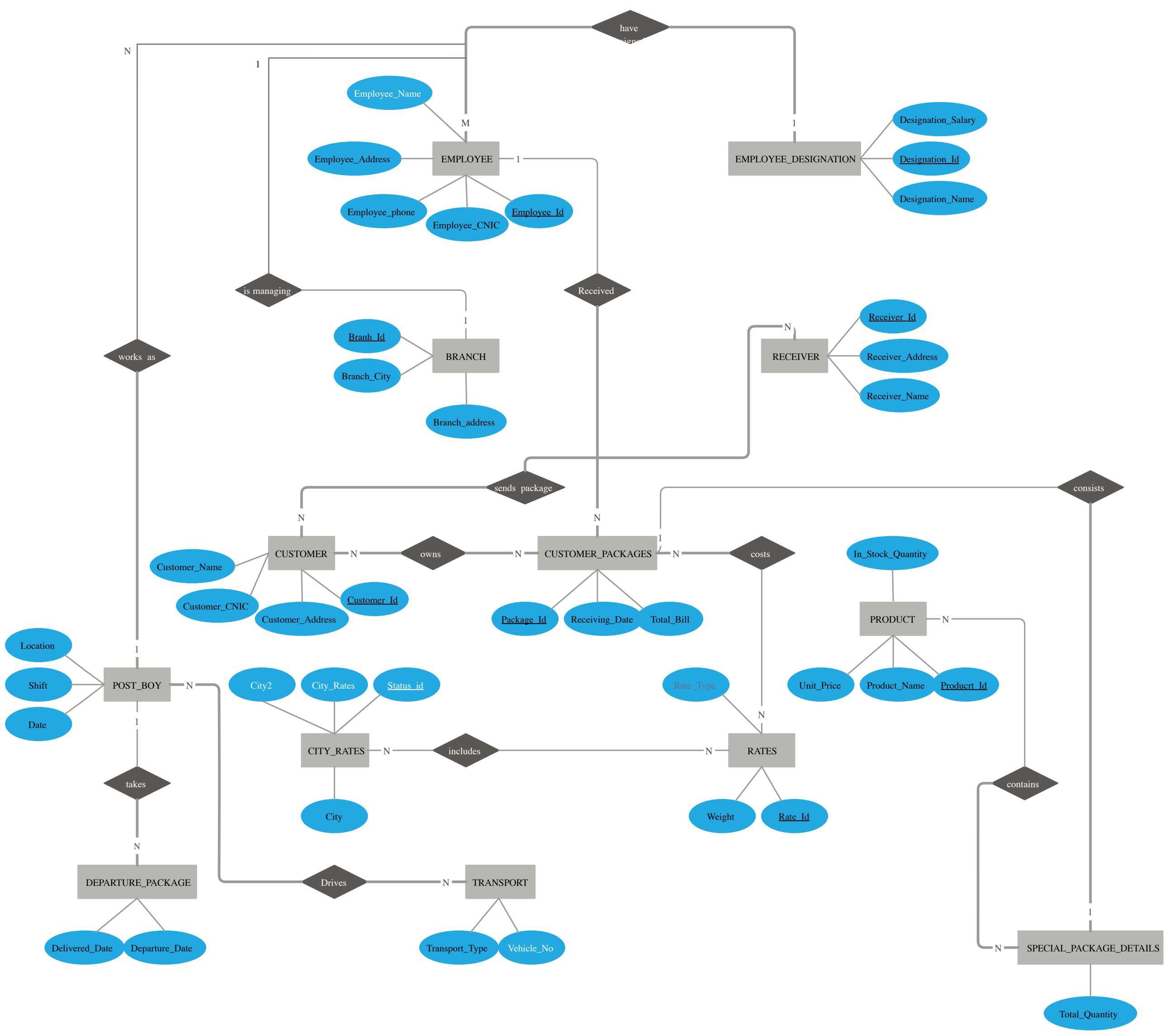
```
│       │       ├── custom_exceptions.py
│       │       ├── exception_handlers.py
│
│       ├── utils/                                # Utility helpers
│       │       ├── constants.py
│       │       ├── validators.py
│
├── alembic/                                      # DB migrations
├── alembic.ini
│
├── tests/                                        # Unit & integration
tests
│       ├── test_auth.py
│       ├── test_shipments.py
│       ├── test_tracking.py
│       ├── test_hubs.py
│       ├── test_admin.py
│
├── Dockerfile
├── docker-compose.yml
├── requirements.txt
├── .env
└── README.md
```

Database Design

ER Diagram


OrderDetails
Orders
Customers
Route-leg hierarchy
Packages and Collies
from
Locations
from
Routes and legs
to
to
Package-Collie hierarchy
Damage relationship
Transport operators
Container-route reservations
Scheduled routes and legs
Physical containers
Transport media like ships, airplanes, and trucks


users
user_id
first_name
last_name
email_address
address
orders
order_id
user_id
date_added
products
product_id
product_name
price
retailer
tracking_details
id
order_id
product_id
retailer
tracking_no
order_details
id
order_id
product_id



Tables

1️⃣ users

Column	Type
id	UUID
email	String
password_hash	String
role	Enum(customer, agent, admin)

2️⃣ shipments

Column	Type
id	UUID

tracking_number	String
customer_id	FK(users)
source_address	Text
destination_address	Text
status	created/in_transit/out_for_delivery/delivered
created_at	Timestamp

3️⃣ tracking_updates

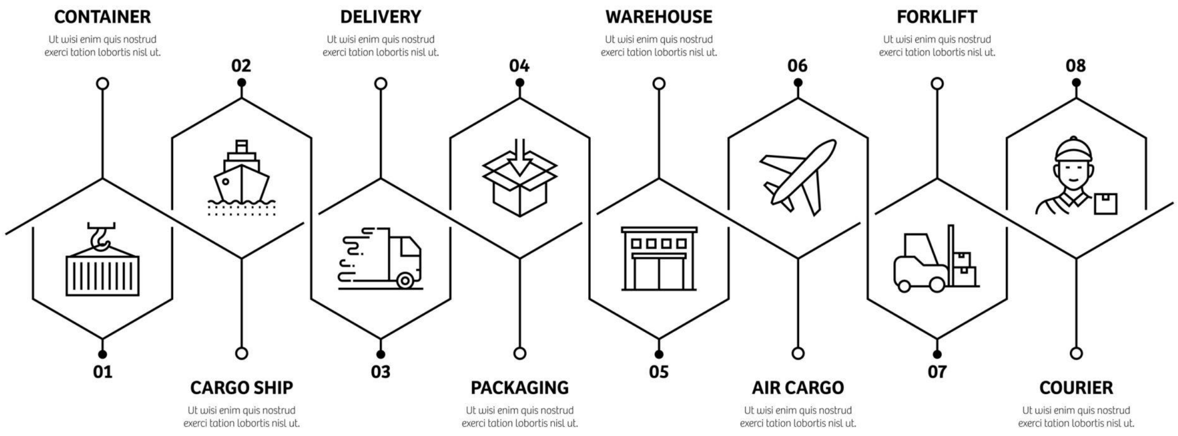
Column	Type
id	UUID
shipment_id	FK(shipments)
location	String
status	String
updated_at	Timestamp

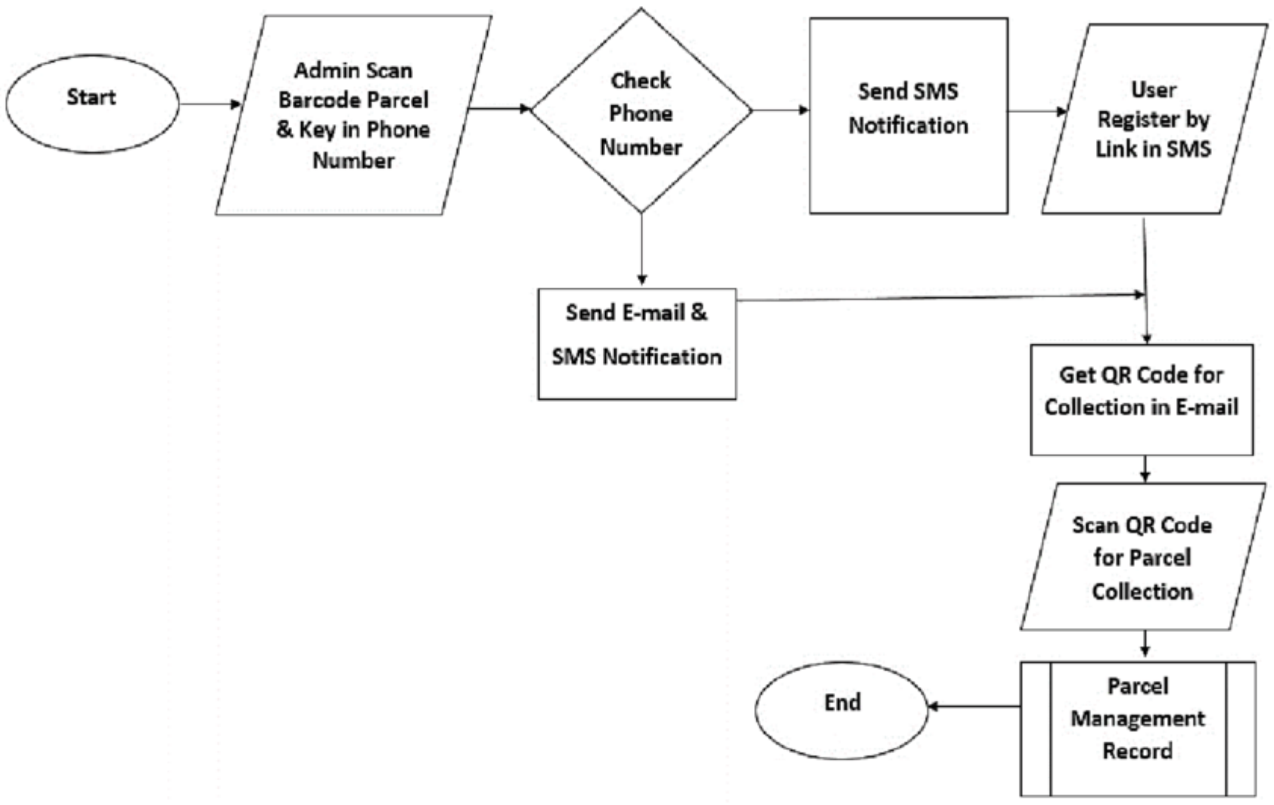
4️⃣ hubs

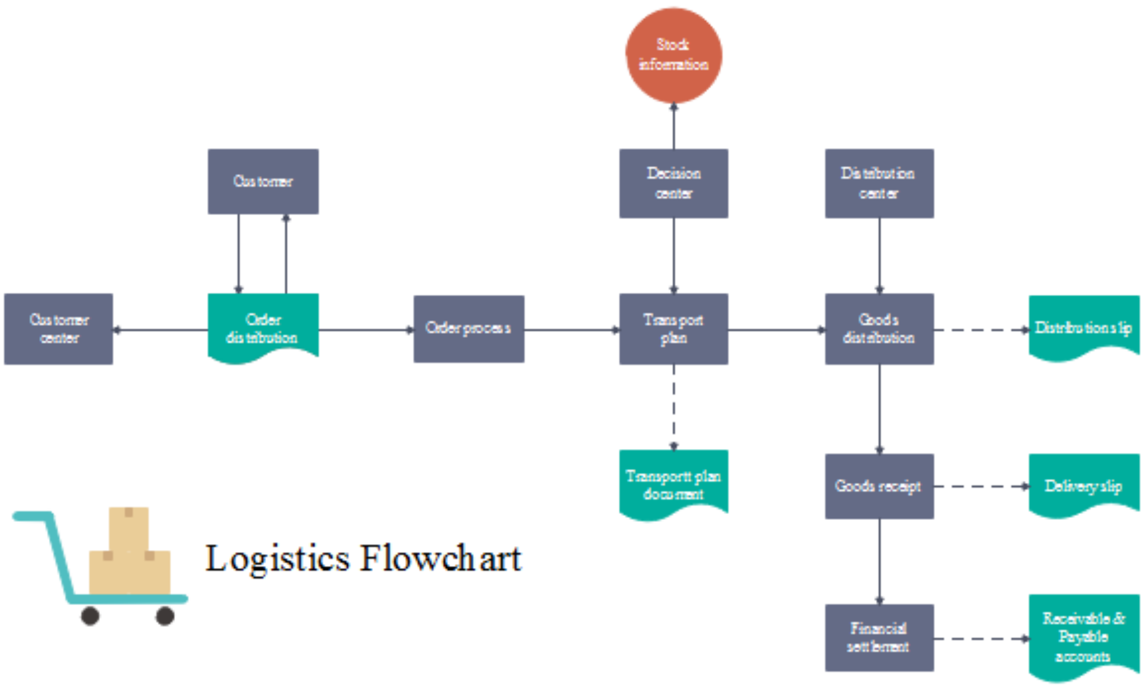
Column	Type
id	UUID
hub_name	String
city	String

Sprint 1 – Shipment Creation & Tracking APIs

Workflow


CONTAINER
Ut wisi enim quis nostrud exerci tation lobortis nisl ut.
01
02
CARGO SHIP
Ut wisi enim quis nostrud exerci tation lobortis nisl ut.
DELIVERY
Ut wisi enim quis nostrud exerci tation lobortis nisl ut.
03
04
PACKAGING
Ut wisi enim quis nostrud exerci tation lobortis nisl ut.
WAREHOUSE
Ut wisi enim quis nostrud exerci tation lobortis nisl ut.
05
06
AIR CARGO
Ut wisi enim quis nostrud exerci tation lobortis nisl ut.
FORKLIFT
Ut wisi enim quis nostrud exerci tation lobortis nisl ut.
07
08
COURIER
Ut wisi enim quis nostrud exerci tation lobortis nisl ut.


Start
Admin Scan Barcode Parcel & Key in Phone Number
Check Phone Number
Send SMS Notification
User Register by Link in SMS
Send E-mail & SMS Notification
Get QR Code for Collection in E-mail
Scan QR Code for Parcel Collection
Parcel Management Record
End



Core Endpoints

POST `/auth/register`

Register customer or agent.

POST `/shipments`

Create new shipment

```
{
  "source_address": "Chennai",
  "destination_address": "Bangalore"
}
```

Response

```
{
  "tracking_number": "TRK123456",
  "status": "created"
}
```

GET `/shipments/{tracking_number}`

Track shipment.

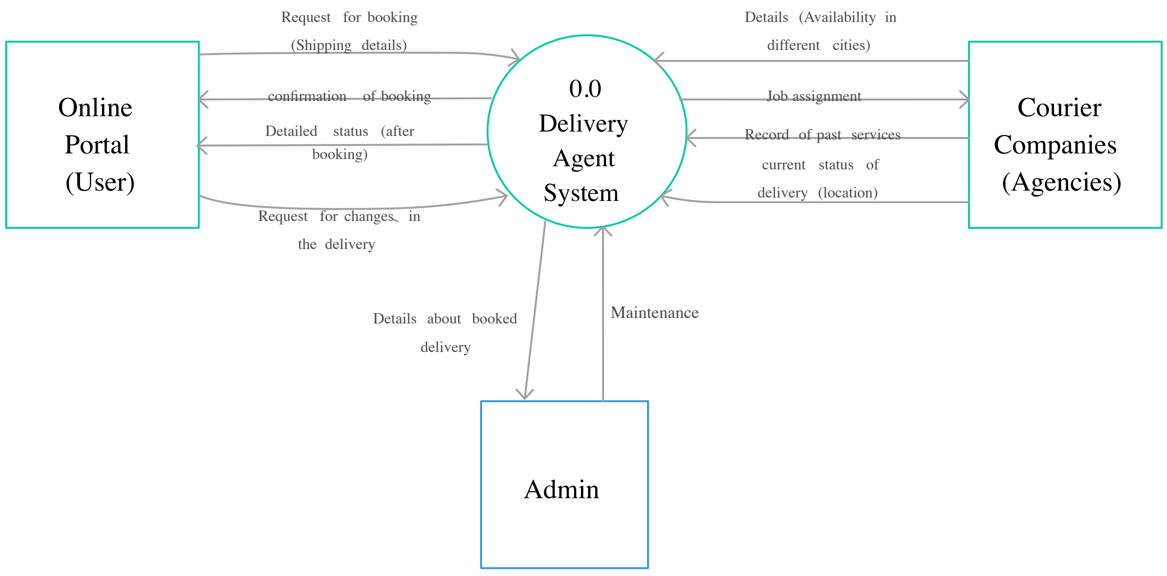
Response:

```
{
  "tracking_number": "TRK123456",
  "status": "in_transit",
  "current_location": "Salem Hub"
}
```

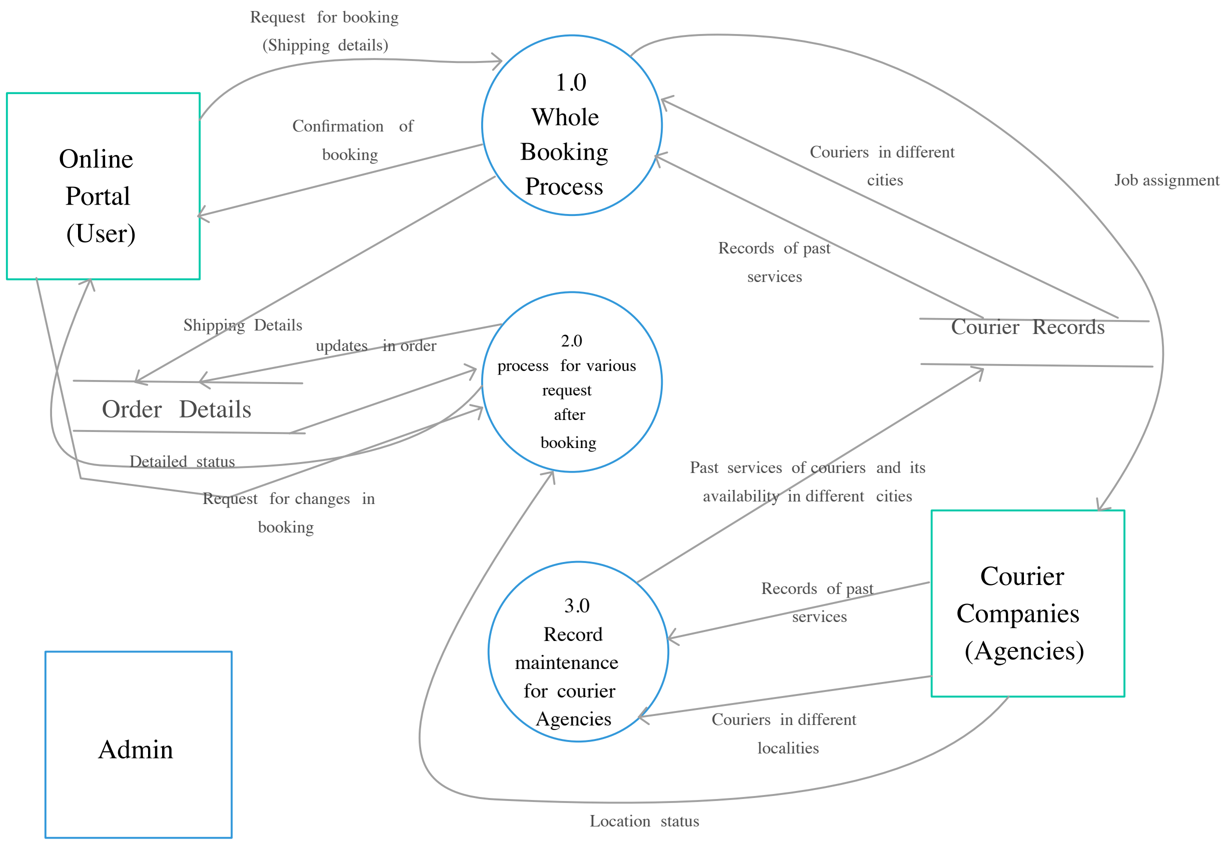
Sprint 2 – Role-Based Updates & Agent Flow

Delivery Workflow

Context level Diagram


Online Portal (User)
Request for booking (Shipping details)
confirmation of booking
Detailed status (after booking)
Request for changes in the delivery
0.0 Delivery Agent System
Details (Availability in different cities)
Job assignment
Record of past services
current status of delivery (location)
Courier Companies (Agencies)
Details about booked delivery
Maintenance
Admin

Level 0 Diagram


Request for booking (Shipping details)
Online Portal (User)
Confirmation of booking
1.0 Whole Booking Process
Couriers in different cities
Job assignment
Records of past services
Courier Records
Shipping Details
updates in order
Order Details
2.0 process for various request after booking
Detailed status
Request for changes in booking
Past services of couriers and its availability in different cities
3.0 Record maintenance for courier Agencies
Records of past services
Courier Companies (Agencies)
Couriers in different localities
Location status
Admin

1

Order Booking

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

2

Product Packaging

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

3

Delivery Route

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

4

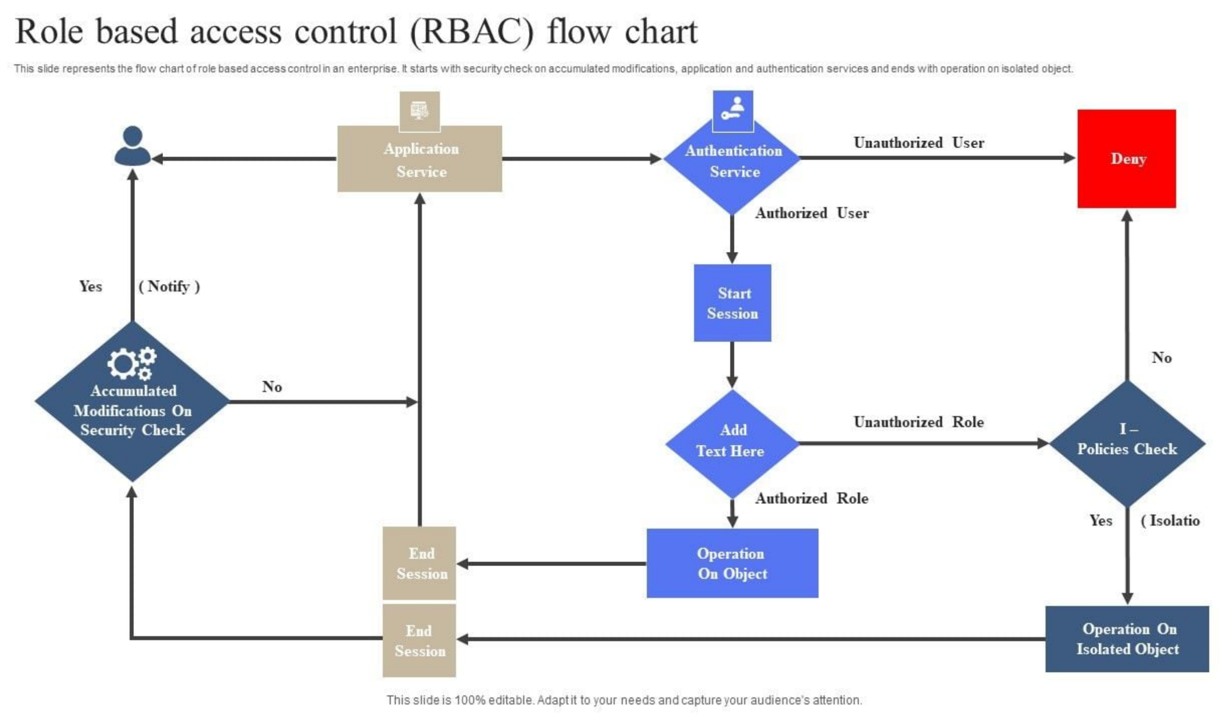
Courier Delivery

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

5

Door to Door Belivery

Lorem ipsum dolor sit amet, consectetur adipiscing elit.


Role based access control (RBAC) flow chart
This slide represents the flow chart of role based access control in an enterprise. It starts with security check on accumulated modifications, application and authentication services and ends with operation on isolated object.
Application Service
Authentication Service
Unauthorized User
Deny
Authorized User
Yes
(Notify)
Start Session
Accumulated Modifications On Security Check
No
No
Add Text Here
Unauthorized Role
I – Policies Check
Authorized Role
Yes
(Isolatio
End Session
Operation On Object
End Session
Operation On Isolated Object
This slide is 100% editable. Adapt it to your needs and capture your audience's attention.

4

Agent Endpoints

PUT `/shipments/{id}/status`

Agent updates shipment status.

```
{
  "status": "out_for_delivery",
  "location": "Bangalore Hub"
}
```

POST `/tracking/{shipment_id}`

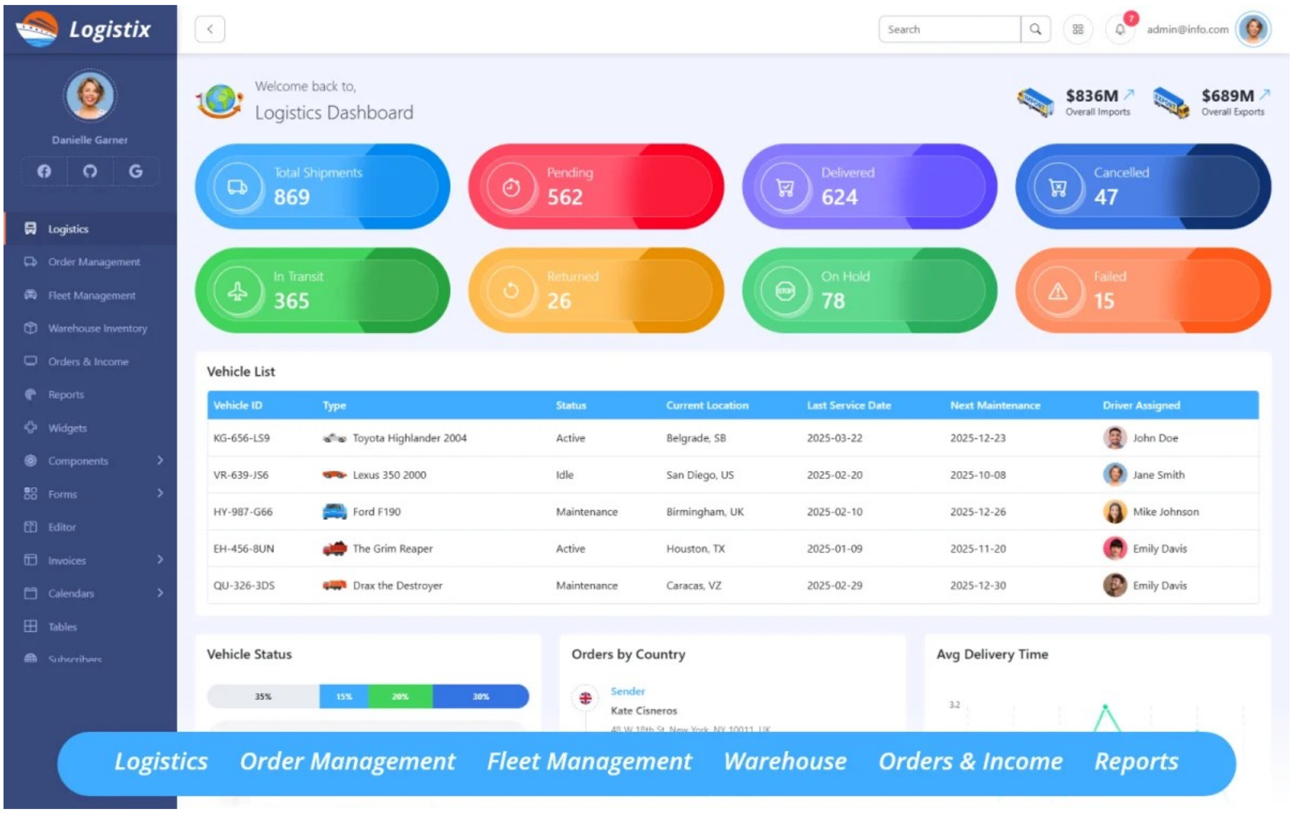
Add tracking update.

Customer APIs

Method	Endpoint	Description
GET	/shipments	View all shipments
GET	/shipments/{id}	Track shipment
DELETE	/shipments/{id}	Cancel shipment (if not dispatched)

Sprint 3 – Admin & Hub Management

Admin Workflow



Welcome back to,
Logistics Dashboard

$836M Overall Imports
$689M Overall Exports

Total Shipments 869
Pending 562
Delivered 624
Cancelled 47
In Transit 365
Returned 26
On Hold 78
Failed 15

Vehicle List

Vehicle ID	Type	Status	Current Location	Last Service Date	Next Maintenance	Driver Assigned
KG-656-LS9	Toyota Highlander 2004	Active	Belgrade, SB	2025-03-22	2025-12-23	John Doe
VR-639-JS6	Lexus 350 2000	Idle	San Diego, US	2025-02-20	2025-10-08	Jane Smith
HY-987-G66	Ford F190	Maintenance	Birmingham, UK	2025-02-10	2025-12-26	Mike Johnson
EH-456-8UN	The Grim Reaper	Active	Houston, TX	2025-01-09	2025-11-20	Emily Davis
QU-326-3DS	Drax the Destroyer	Maintenance	Caracas, VZ	2025-02-29	2025-12-30	Emily Davis

Vehicle Status

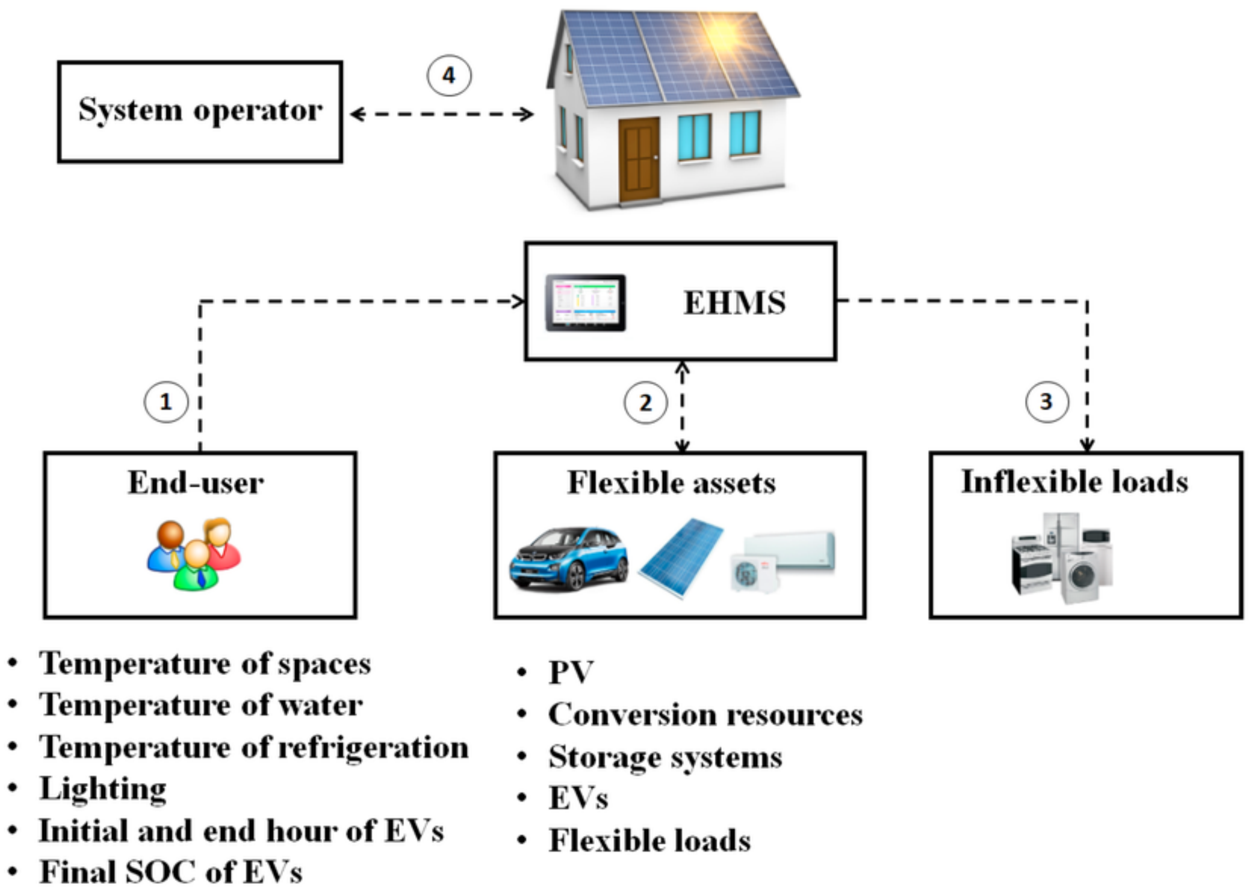
35% 15% 20% 30%

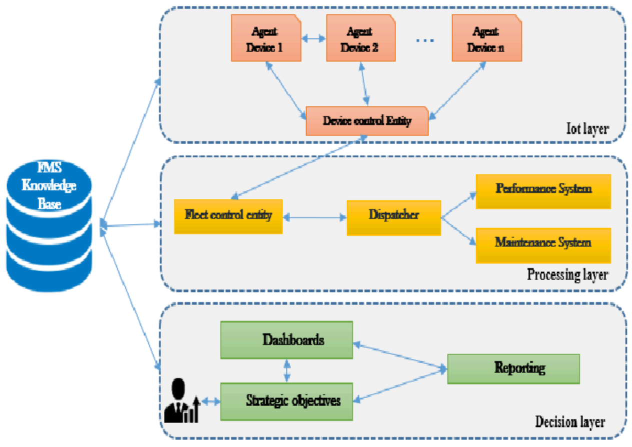
Orders by Country

Sender
Kate Cisneros

Avg Delivery Time

3.2





4

Admin APIs

POST /admin/hubs

Create hub.

GET /admin/reports

```
{
  "total_shipments_today": 350,
  "delivered": 300,
  "in_transit": 50
}
```

GET `/admin/users`

DELETE `/admin/users/{id}`

Additional PUT & DELETE APIs

Shipment Management

Method	Endpoint	Role
PUT	/shipments/{id}	Customer
DELETE	/shipments/{id}	Customer
PUT	/shipments/{id}/assign-agent	Admin

Hub Management

Method	Endpoint	Role
PUT	/admin/hubs/{id}	Admin
DELETE	/admin/hubs/{id}	Admin

Integration Test Cases

Test Shipment Creation

```
def test_create_shipment(auth_token):
    response = client.post(
        "/shipments",
        headers={"Authorization": f"Bearer {auth_token}"},
        json={
            "source_address": "Chennai",
            "destination_address": "Bangalore"
        }
    )
    assert response.status_code == 201
```

Test Status Update

```
def test_update_status(agent_token):
    response = client.put(
        "/shipments/uuid/status",
        headers={"Authorization": f"Bearer {agent_token}"},
        json={"status": "in_transit", "location": "Salem"}
    )
    assert response.status_code == 200
```

Test Cancel Shipment

```
def test_cancel_shipment(auth_token):
    response = client.delete(
        "/shipments/uuid",
        headers={"Authorization": f"Bearer {auth_token}"}
    )
    assert response.status_code == 200
```

Microservice Split

1️⃣ Auth Service

2️⃣ Shipment Service

3️⃣ Tracking Service

4️⃣ Hub Service

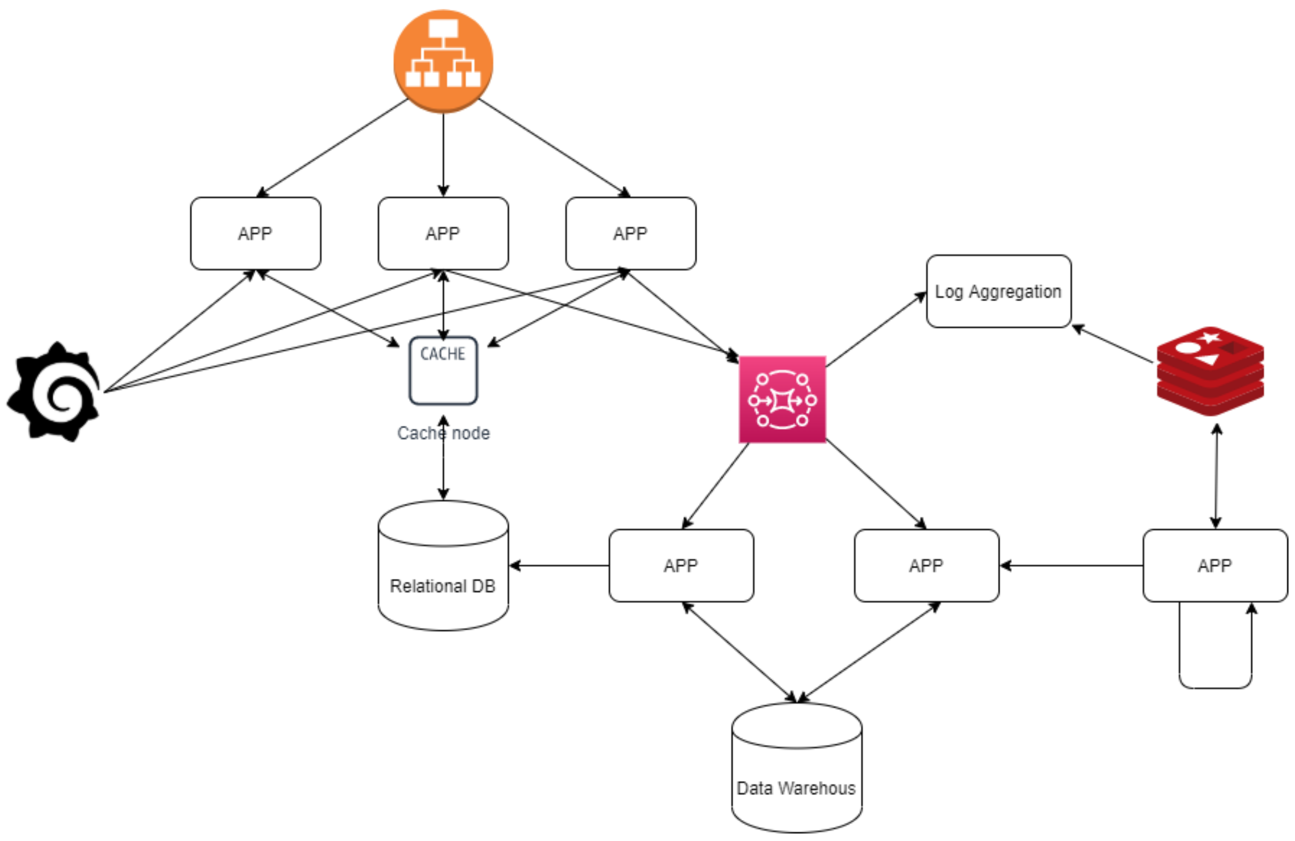
5️⃣ Reporting Service

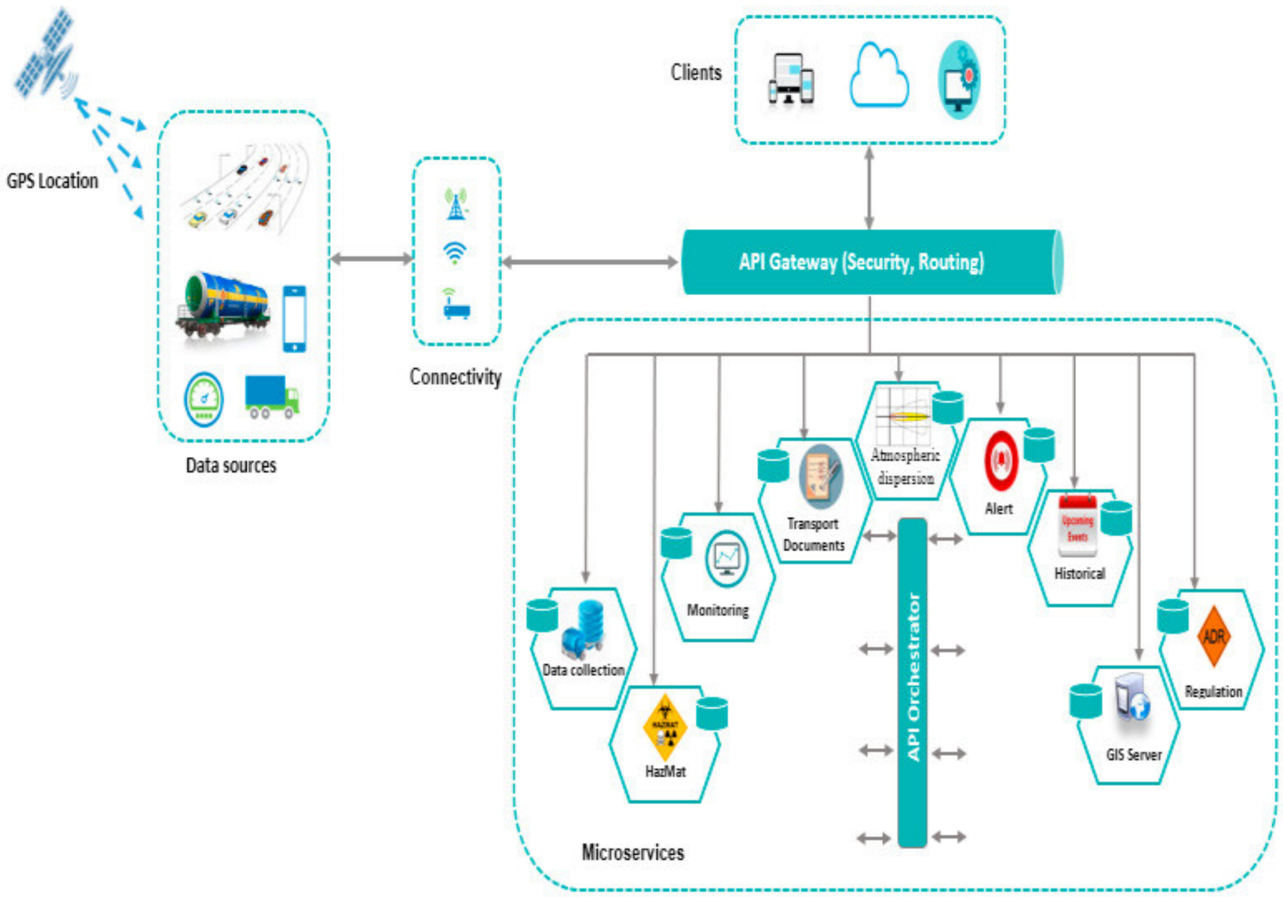
6️⃣ Kafka Cluster

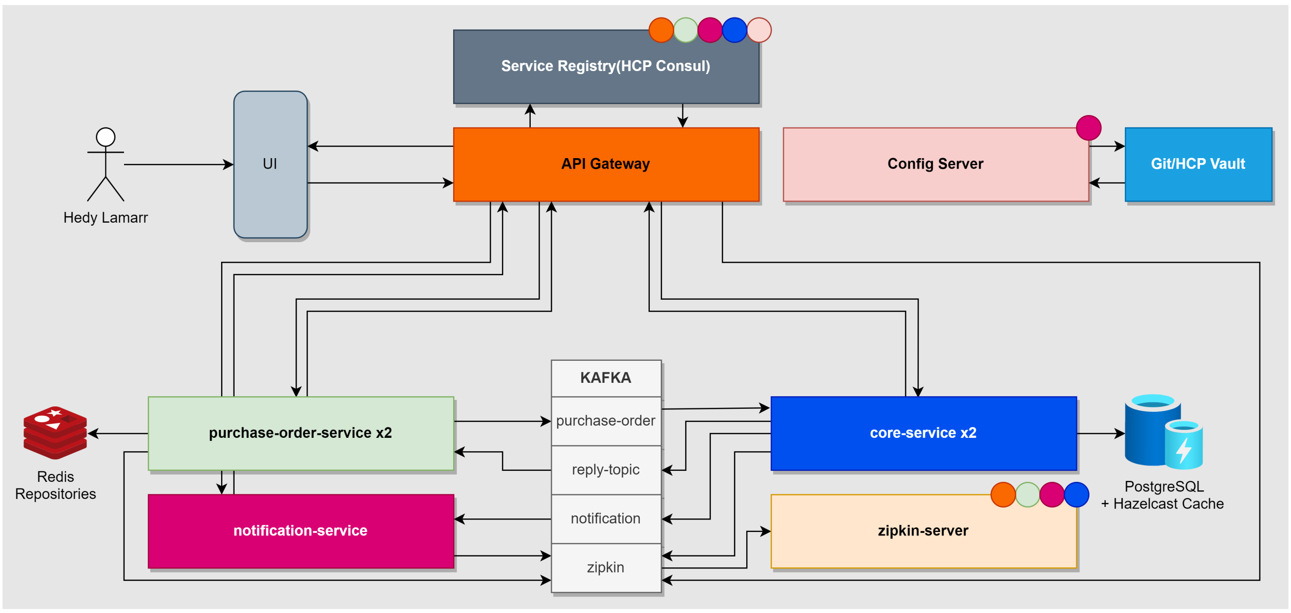
7️⃣ Redis

8️⃣ PostgreSQL

Enterprise Architecture Diagram


APP
APP
APP
Log Aggregation
CACHE
Cache node
Relational DB
APP
APP
APP
Data Warehous


Clients
GPS Location
Data sources
Connectivity
API Gateway (Security, Routing)
Atmospheric dispersion
Alert
Transport Documents
Historical
Monitoring
Data collection
HazMat
API Orchestrator
Regulation
GIS Server
Microservices



4

Communication Pattern (Pure Event-Driven)

Type	Technology
Client → Service	REST
Service → Service	Kafka
Real-time tracking	Redis
Persistence	PostgreSQL
Deployment	Docker

Kafka Topic Design

Topic	Producer	Consumers
user.created	Auth Service	Reporting
shipment.created	Shipment Service	Hub, Reporting, Notification
shipment.assigned	Hub Service	Reporting
shipment.status.updated	Shipment Service	Tracking, Reporting
shipment.delivered	Shipment Service	Reporting

Service Responsibilities (Event-Driven)

1️⃣ Auth Service

Responsibilities

- Register
- Login
- JWT creation
- Publish user events

Publishes

```
user.created
```

No service calls Auth via REST.
JWT validation is local using shared secret/public key.

2️⃣ Shipment Service

Responsibilities

- Create shipment
- Update shipment
- Assign agent
- Change shipment status

Publishes

```
shipment.created
shipment.status.updated
shipment.delivered
```

No direct communication with Hub or Tracking.

3️⃣ Hub Service

Responsibilities

- Manage hubs
- Assign shipments to hubs

Consumes

```
shipment.created
```

Publishes

```
shipment.assigned
```

4️⃣ Tracking Service

Responsibilities

- Store tracking history
- Maintain Redis cache for latest shipment status

Consumes

```
shipment.status.updated
shipment.delivered
```

Workflow

- Save tracking record in DB
- Update Redis cache
- Expose REST endpoint for customer tracking

5️⃣ Reporting Service

Responsibilities

- Analytics
- Performance metrics
- Daily reports
- Hub metrics

Consumes

```
user.created
shipment.created
shipment.assigned
shipment.status.updated
shipment.delivered
```

Maintains aggregated reporting DB.

Enterprise Docker Setup

Folder Structure

```
enterprise-logistics/
│
├── services/
│   ├── auth-service/
│   ├── shipment-service/
│   ├── hub-service/
│   ├── tracking-service/
│   ├── reporting-service/
│
├── docker-compose.yml
└── .env
```

Enterprise docker-compose.yml