**Case Study: Simple Transportation Microservices System**

**1. Overview**

You want to build a transportation platform consisting of multiple microservices communicating with each other.  
This system will include:

* Vehicle Service
* Route Service
* Driver Service
* Booking Service

Each microservice will register with Eureka Server for discovery.  
API Gateway will route external requests to respective services.  
Circuit breaker pattern will ensure resilience in service calls.

**2. Microservices & Responsibilities**

| **Service Name** | **Responsibility** |
| --- | --- |
| **Vehicle Service** | Manage vehicles: add, update, get details of vehicles |
| **Route Service** | Manage routes: define routes, stops, distance, duration |
| **Driver Service** | Manage drivers: personal info, license, availability |
| **Booking Service** | Manage trip bookings: book a vehicle & driver for a route |

**3. Entities and their details**

**Vehicle Service Entities**

* **Vehicle**
  + vehicleId (Long, PK)
  + vehicleNumber (String, unique)
  + type (Enum: BUS, CAR, VAN, TRUCK)
  + capacity (Integer)
  + status (Enum: AVAILABLE, ON\_TRIP, MAINTENANCE)
  + createdAt (Timestamp)
  + updatedAt (Timestamp)

**Route Service Entities**

* **Route**
  + routeId (Long, PK)
  + source (String)
  + destination (String)
  + distanceKm (Double)
  + estimatedDurationMin (Integer)
  + stops (List<String>) — sequence of stops along the route
  + createdAt (Timestamp)
  + updatedAt (Timestamp)

**Driver Service Entities**

* **Driver**
  + driverId (Long, PK)
  + name (String)
  + licenseNumber (String, unique)
  + phoneNumber (String)
  + email (String)
  + availabilityStatus (Enum: AVAILABLE, ON\_TRIP, OFF\_DUTY)
  + vehicleId (Long) — FK to Vehicle service (for assigned vehicle)
  + createdAt (Timestamp)
  + updatedAt (Timestamp)

**Booking Service Entities**

* **Booking**
  + bookingId (Long, PK)
  + customerName (String)
  + contactNumber (String)
  + routeId (Long) — FK to Route service
  + vehicleId (Long) — FK to Vehicle service
  + driverId (Long) — FK to Driver service
  + bookingStatus (Enum: PENDING, CONFIRMED, COMPLETED, CANCELLED)
  + bookingTime (Timestamp)
  + startTime (Timestamp)
  + endTime (Timestamp)
  + createdAt (Timestamp)
  + updatedAt (Timestamp)

**4. Architecture flow & interaction**

* All microservices register with **Eureka Server** for discovery.
* External clients call the **API Gateway** (e.g., Spring Cloud Gateway) which routes requests to appropriate services.
* The **Booking Service** coordinates between Vehicle, Route, and Driver services when creating a booking:
  + Checks availability of vehicles and drivers via service calls
  + Selects an available vehicle and driver
  + Books the trip and updates statuses accordingly
* Communication between services happens using **REST API calls** via service discovery.
* Use **Circuit Breaker** (e.g., Resilience4j or Netflix Hystrix) on inter-service calls in Booking Service to handle failures gracefully (fallbacks, retries).

**5. Circuit Breaker Implementation Notes**

* In **Booking Service**, wrap calls to Vehicle, Driver, and Route services with circuit breakers.
* On failure (timeout, connection error), fallback logic can:
  + Return cached data or empty responses
  + Notify user about unavailability
* Circuit breaker will prevent cascading failures when a downstream service is down.

**Summary diagram (conceptual)**

Client

|

API Gateway

|

|--------> Vehicle Service (Vehicle Entity)

|

|--------> Driver Service (Driver Entity)

|

|--------> Route Service (Route Entity)

|

|--------> Booking Service (Booking Entity)

|

(Calls other services for data & coordination with circuit breaker)

**Public Sector Case Study: Municipal Services Management System**

**Scenario**

A municipal government wants a scalable system to manage different public services like:

* **Citizen Service Requests** (e.g., complaints, requests for repair)
* **Department Service Catalog** (e.g., types of services provided by city departments)
* **Employee Management** (municipal staff assigned to requests)
* **Notifications** (sending status updates to citizens)

The microservices architecture will be:

| **Service Name** | **Responsibilities** | **Main Entities** |
| --- | --- | --- |
| **Citizen Service Service** | Manage citizen requests (CRUD), assign employees | Request, RequestStatus, Citizen |
| **Department Service** | Manage city departments and their service types | Department, ServiceType |
| **Employee Service** | Manage municipal employees, departments mapping | Employee, Department |
| **Notification Service** | Send notifications (SMS/email) on request updates | Notification, Request |

**Architecture Overview**

* **Eureka Server**: For service discovery, all services register here.
* **API Gateway (Spring Cloud Gateway)**: Routes incoming requests to appropriate microservices.
* **Circuit Breaker (Resilience4j)**: Prevent cascading failures, fallback methods on downstream service failures.
* **Config Server** (optional but recommended) for centralized configuration.

**Entity Relationships & Multi-Table Design**

**1. Citizen Service Service**

* **Request**
  + requestId (PK)
  + title
  + description
  + status (FK to RequestStatus)
  + citizenId (FK to Citizen)
  + assignedEmployeeId (FK to Employee - from Employee Service via API)
  + departmentId (FK to Department - from Department Service via API)
  + createdDate
  + updatedDate
* **RequestStatus**
  + statusId (PK)
  + statusName (e.g., New, In Progress, Resolved, Closed)
* **Citizen**
  + citizenId (PK)
  + name
  + email
  + phoneNumber

**2. Department Service**

* **Department**
  + departmentId (PK)
  + departmentName
  + location
* **ServiceType**
  + serviceTypeId (PK)
  + serviceName (e.g., Waste Management, Road Repair)
  + departmentId (FK to Department)

**3. Employee Service**

* **Employee**
  + employeeId (PK)
  + name
  + email
  + phoneNumber
  + departmentId (FK to Department)

**4. Notification Service**

* **Notification**
  + notificationId (PK)
  + requestId (FK to Request)
  + message
  + status (Sent, Failed)
  + sentDate

**Inter-service Communication**

* Services communicate via Feign client
* Example:
  + Citizen Service fetches employee info by calling Employee Service API.
  + Citizen Service fetches department info by calling Department Service API.
* API Gateway routes all incoming requests and acts as the single entry point.
* Eureka Server handles service registry and discovery for all services.

**Circuit Breaker Implementation (Using Resilience4j or Spring Cloud)**

* Each service client will wrap calls with a circuit breaker.
* If downstream service is down or slow, fallback methods will provide default responses or cached data.
* Example:
  + Citizen Service calls Employee Service — If Employee Service fails, fallback returns "Employee data not available."

**Sample Entity Diagram (High-Level)**

Citizen (1) <-- (Many) Request (Many) --> (1) RequestStatus

Request --> assignedEmployeeId (calls Employee Service)

Request --> departmentId (calls Department Service)

Department (1) <-- (Many) ServiceType

Employee (Many) --> Department (1)

Notification --> Request (1)

**Summary**

| **Service** | **Entities / Tables** | **Relationships** | **Notes** |
| --- | --- | --- | --- |
| Citizen Service | Request, RequestStatus, Citizen | Request → Citizen (Many-to-One), RequestStatus (Many-to-One) | Calls Employee & Department via REST |
| Department Service | Department, ServiceType | ServiceType → Department (Many-to-One) | Manages city departments |
| Employee Service | Employee, Department | Employee → Department (Many-to-One) | Manages municipal employees |
| Notification Service | Notification | Notification → Request (Many-to-One) | Sends status updates |