Data Model and Ownership Strategy

Target Audience: Database Engineers, Backend Developers
Purpose: Define the conceptual data model, ownership boundaries, and consistency guarantees across the distributed system.

---

1. Data Ownership Principles

In this microservices architecture, **Data Ownership** is improved over a monolithic database.

- **Write Authority:** Only ONE service has the authority to WRITE to a specific entity table.
- **Read Access:** Other services must request data via API or subscribe to data replication events. access via direct DB join is strictly prohibited.
- **Referencing:** Cross-service relationships are maintained via "Soft Foreign Keys" (storing the UUID of an entity from another service).

---

2. Core Domain Entities

2.1 Identity Domain (Auth Service)

- **User:** Represents a human or system actor.
  - *Key Attributes:* `id` (UUID), `email`, `password_hash`, `is_active`.
- **Role:** Defines a set of permissions (e.g., "Company Admin", "Driver").
- **Ownership: Auth Service** is the sole writer.
- **External Reference:** All other services store `user_id` to link records (e.g., "Ticket sold by `user_id`").

2.2 Transport Domain (Company Service)

This is the heaviest domain, defining the physical and planned world.

- **Company:** A transport operator (Tenant).
  - *Key Attributes:* `id`, `name`, `tin_number`.
- **Bus:** A physical vehicle.
  - *Key Attributes:* `plate_number`, `capacity`, `features` (AC, WiFi).

    - *Relationship:* Belongs to one Company.
- **Route:** A defined path between two cities.
    - *Structure:* Composed of multiple **Segments** and **Stops**. (e.g., Kigali -> Huye -> Rusizi).
- **Schedule (Trip):** An instance of a Route occurring at a specific time.
    - *Key Attributes:* `departure_time`, `assigned_bus_id`, `assigned_driver_id`, `status` (Scheduled, Departed, Completed, Cancelled).
    - *Role:* This is the central "Product" that is sold.
- **Ownership: Company Service** is the sole writer.

2.3 Sales Domain (Ticketing Service)

- **Ticket:** A contract of carriage.
    - *Key Attributes:* `ticket_id`, `schedule_id` (Ref), `seat_number`, `passenger_details`, `status` (Reserved, Paid, Cancelled), `price`.
    - *Consistency:* Holds a soft reference to `schedule_id` (from Company Service).
- **Ownership: Ticketing Service** is the sole writer.

2.4 Financial Domain (Payment Service)

- **Transaction:** A record of money movement.
    - *Key Attributes:* `amount`, `currency`, `provider_ref` (M-Pesa ID), `status`.
    - *Relationship:* Linked to a `ticket_id`.
- **Ownership: Payment Service** is the sole writer.

---

3. Cross-Service Relationships

3.1 The "Schedule to Ticket" Link

- **Problem:** Tickets (in Ticketing Service) need to know about Schedules (in Company Service).
- **Solution:**
    1. **Creation:** When a Schedule is "Published" in Company Service, an event `ScheduleCreated` is emitted.
    2. **Replication (Optional):** Ticketing Service *may* cache a minimal version of the schedule (ID, Price, Capacity) for performance.
    3. **Validation:** When selling a ticket, Ticketing Service queries Company Service (or its cache) to ensure the Schedule exists and the Bus has capacity.

3.2 The "Ticket to Payment" Link

- **Problem:** Payment needs to know how much to charge for a Ticket.
- **Solution:**
  1. Client sends booking request to Ticketing Service.
  2. Ticketing Service locks seat and returns a `booking_reference`.
  3. Client sends `booking_reference` to Payment Service.
  4. Payment Service validates the amount associated with that reference before processing.

---

4. Consistency Model

The system chooses **Availability** over strict Consistency (CAP Theorem) for read operations, but demands **Strong Consistency** for Write operations within a domain.

- **Seat Inventory: Strong Consistency.** We use Distributed Locking (Redis Redlock) to ensure a seat cannot be sold twice simultaneously.
- **Reporting: Eventual Consistency.** The "Super Admin Dashboard" aggregates data from events. It might be 1-2 seconds behind real-time, which is acceptable for analytics.
- **Offline Sales: Conflict Resolution.** If a POS device sells a seat offline that was sold online 1 minute prior, the Server rejects the POS sync upload. The "Online Server" is always the source of truth.

---

5. Schema Evolution Strategy

- **No Breaking Changes:** Services must not rename columns used by public APIs without a versioned deprecation phase.
- **Additive Changes:** Adding a column is safe.
- **Migrations:** Each service manages its own migration scripts (Alembic for Python). There is no "Master DB Migration" script.