| **#** | **Service Name** | **Category** | **Responsibility** | **Key Role / Key Entities Owned** |
| --- | --- | --- | --- | --- |
| 1 | config-server | Foundational / Infrastructure | Central configuration management. Handles all externalized configuration files (.yml), database credentials, and API keys for every service. | Single Source of Truth for Settings. |
| 2 | eureka-server | Foundational / Infrastructure | Service discovery registry. Allows microservices to locate each other by name and monitor their health. | Service Phone Book. |
| 3 | api-gateway | Foundational / Infrastructure | Acts as the single entry point for client (mobile/web) requests. Manages JWT validation, rate limiting, and dynamic routing. | Security Guard & Traffic Controller. |
| 4 | auth-service | Identity and User Services | Manages user authentication, registration, and JWT generation/validation. | User Credentials (Hashed Passwords). |
| 5 | customer-service | Identity and User Services | Manages all non-security-related customer data including personal info, saved addresses, and preferences. | CustomerProfile, CustomerAddress. |
| 6 | rider-service | Identity and User Services | Handles all data related to riders/drivers such as license details, vehicle info, availability, and rating. | RiderProfile, Vehicle, RiderRating. |
| 7 | order-service | Core Business & Logistics Services | Executes order-management logic including order creation, tracking, cost calculation, and maintaining order history. | Order, Parcel, OrderHistory. |
| 8 | dispatch-service | Core Business & Logistics Services | Manages real-time logistics including geo-location tracking, order-to-rider matching, and delivery job lifecycle. | DispatchJob, RiderLocation. |
| 9 | payment-service | Core Business & Logistics Services | Handles all financial operations including payment processing, integration with external gateways, and managing refunds and invoices. | Transaction, PaymentMethod. |
| 10 | notification-service | Communication Service | Interfaces with third-party providers to send various alerts and notifications (Push, SMS, Email) reliably. | NotificationLog, DeviceToken. |

**P2P Booking Flow: Sequence of Events**

Here is the step-by-step process for a typical Customer-Rider interaction in the P2P MVP, showing how the microservices interact with one another.

**Phase 1: Authentication & Order Creation**

1. **Client $\rightarrow$ API Gateway:** A Customer attempts to log in using credentials.
2. **API Gateway $\rightarrow$ Auth Service:** Routes the request to the auth-service for validation.
3. **Auth Service:** Validates credentials, generates a **JWT**, and returns it to the Client via the API Gateway.
4. **Client $\rightarrow$ API Gateway:** Customer sends a POST /api/v1/orders request with the JWT and pickup/dropoff details.
5. **API Gateway:** **Validates the JWT** (via a quick check with the auth-service or using the token's public key) and injects the customer\_id into the request headers.
6. **API Gateway $\rightarrow$ Order Service:** Routes the request.
7. **Order Service (Logic):**
   * Creates the new Order and Parcel records (Status: BOOKED).
   * Calls the customer-service and rider-service APIs (if needed) to fetch customer/rider details for logging purposes (e.g., driver's current status).
   * **Sends Event:** Publishes a message (e.g., using a message broker like Kafka/RabbitMQ) or makes an API call to the **Dispatch Service**: "New Job Created for Order ID: 12345."

**Phase 2: Dispatch and Assignment (The Core Logistics)**

1. **Dispatch Service:** Receives the "New Job" request.
2. **Dispatch Service (Logic):**
   * Queries its own real-time data (RiderLocation) and the rider-service API to find all available, nearby riders.
   * Executes the **P2P Matching Algorithm** (e.g., nearest available rider).
   * Assigns the job (Order 12345) to Rider 67890.
   * Updates its own DispatchJob record.
   * **Sends Event to Order Service:** "Order 12345 assigned to Rider 67890."
3. **Order Service:** Updates the Order record's status to PICKUP\_PENDING and fills the driver\_id.
4. **Dispatch Service $\rightarrow$ Notification Service:** Sends two requests:
   * "Notify Customer X: Your rider has been assigned."
   * "Notify Rider Y: You have a new job offer (Order 12345)."
5. **Notification Service:** Looks up device tokens, formats the message, and sends the push notification via Firebase/APNS.

**Phase 3: Delivery and Completion**

1. **Rider App $\rightarrow$ API Gateway $\rightarrow$ Order Service:** The Rider updates the status (e.g., "Picked Up," "Delivered").
2. **Order Service (Logic):** Updates the Order status, calculates the final cost/time, and updates the history.
3. **Order Service $\rightarrow$ Payment Service:** Sends a request: "Process payment for Order 12345 (Amount: $10.50)."
4. **Payment Service:** Interfaces with the payment gateway (e.g., Stripe, PayPal), processes the transaction, and returns the status.
5. **Order Service $\rightarrow$ Notification Service:** Sends: "Notify Customer X: Order 12345 is Delivered."