**🧩 Microservice Architecture Plan for EMS (Employee Management System)**

Here are the key **independent microservices** you can create:

**1. User Service (Auth + User Info)**

Handles:

* Employee and Manager registration/login
* JWT token generation/validation
* Role-based access control

🔑 **Key Features**:

* User table (ID, Name, Email, Role, Password)
* Token authentication
* Login/logout functionality

**2. Employee Management Service**

Handles:

* Add/update/delete/view employee details
* Department management
* Employee profile

📘 **DB Tables**:

* Employee
* Department

**3. Leave Management Service**

Handles:

* Leave request by employee
* Leave approval/rejection by manager
* Leave history with filters

📘 **DB Tables**:

* LeaveRequest

🔄 Communicates with:

* User Service (to get employee/manager info)
* Notification Service (for leave status update)

**4. Attendance Service**

Handles:

* Employee check-in/check-out
* Attendance logs and summary

📘 **DB Tables**:

* AttendanceLog

🔄 Can include:

* Scheduled tasks to auto-mark absent if not checked in by a certain time

**5. Feedback Service**

Handles:

* Employee daily feedback submission
* Manager replies

📘 **DB Tables**:

* Feedback

**6. Task Management Service**

Handles:

* Assigning tasks to employees
* Task status updates (pending, working, completed)
* Deadline tracking

📘 **DB Tables**:

* Task

**7. Ticketing Service**

Handles:

* Employee support tickets
* Status tracking
* Comments on tickets

📘 **DB Tables**:

* Ticket, TicketComment

**8. Chat Service (Optional for MVP)**

Handles:

* Real-time or pseudo-real-time chat
* Message storage between manager and employees

🧰 Consider using:

* WebSocket with STOMP
* Or fallback to polling with REST APIs

**9. Notification Service (Optional)**

Handles:

* Email notifications
* In-app notifications
* Use Spring @Async or messaging queues like Kafka/RabbitMQ for decoupling

**📦 Common Components**

* **API Gateway** (Spring Cloud Gateway)
  + Route requests to appropriate services
  + Can also do auth filtering
* **Config Server** (Spring Cloud Config)
  + Centralized configs for all services
* **Service Discovery** (Eureka or Consul)
  + Auto-register and discover services
* **Centralized Logging**
  + ELK Stack (Elasticsearch, Logstash, Kibana) or simpler file-based logging

**🔐 Communication between services**

* Use **REST APIs** for simpler start
* Later upgrade to **Feign Clients** or **gRPC** for internal communication
* Use **JWT** for secure communication between services

**🗃️ Database Strategy**

| **Approach** | **Description** |
| --- | --- |
| 🟢 **DB-per-service** (Recommended) | Each microservice has its own DB schema |
| 🔴 Shared DB (avoid) | Coupling risk, hard to scale |

Use MySQL for all for now — switch to PostgreSQL/Mongo later if needed.

**🚀 Development & Deployment Flow**

* Each service is a separate Spring Boot project
* Use **Docker** to containerize services
* Use **Docker Compose** to run all locally
* Optional: Use **Kubernetes** for full deployment control