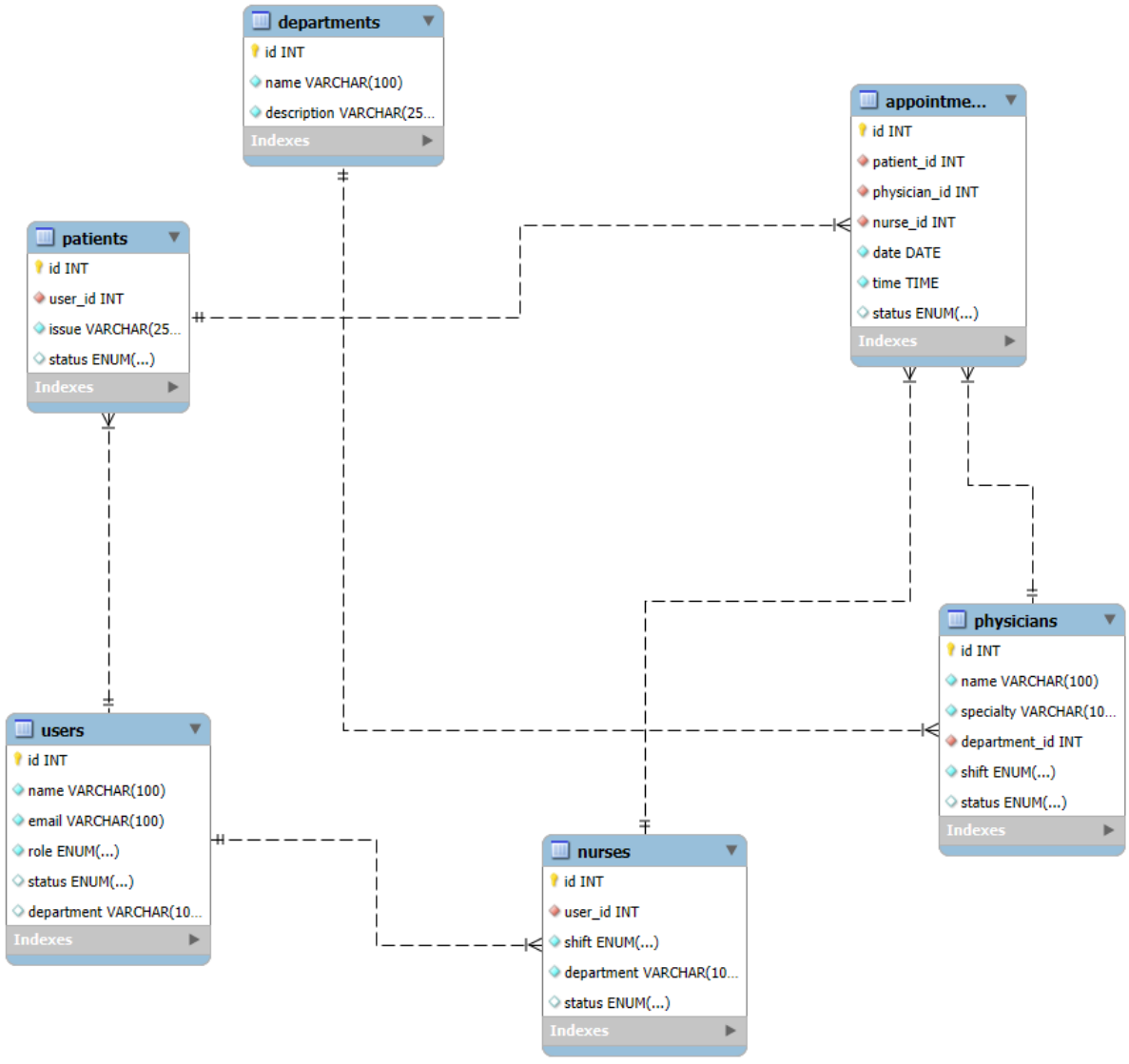
Hospital Management System

(Group 2)

- **Microservices:**
  1) Appointment Service
  2) Physician Service
  3) Admin Service
  4) UIService
  5) Eureka Server
  6) APIGateway

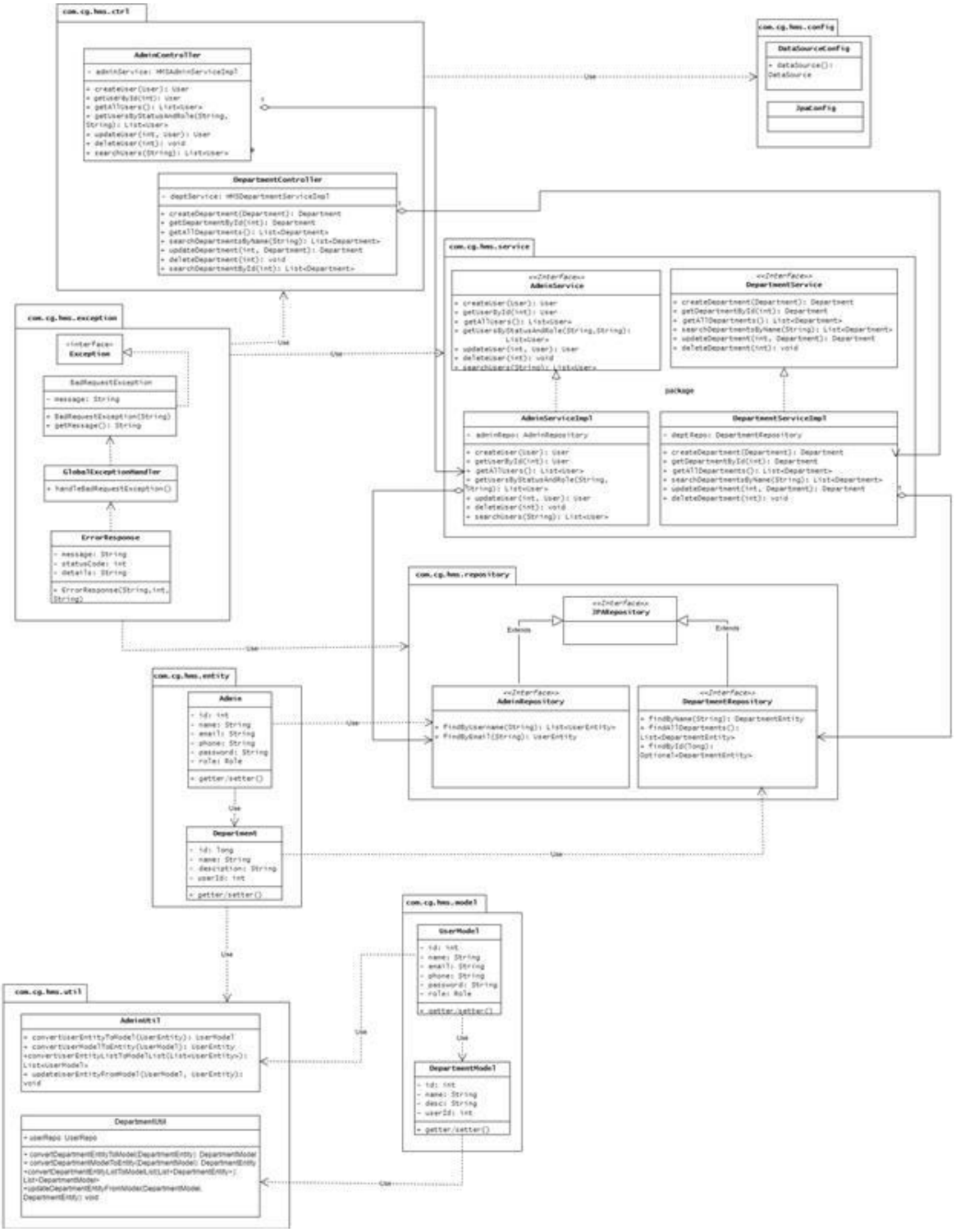
- **Use Case Diagram: –**

- **Entity-Relationship Diagram**

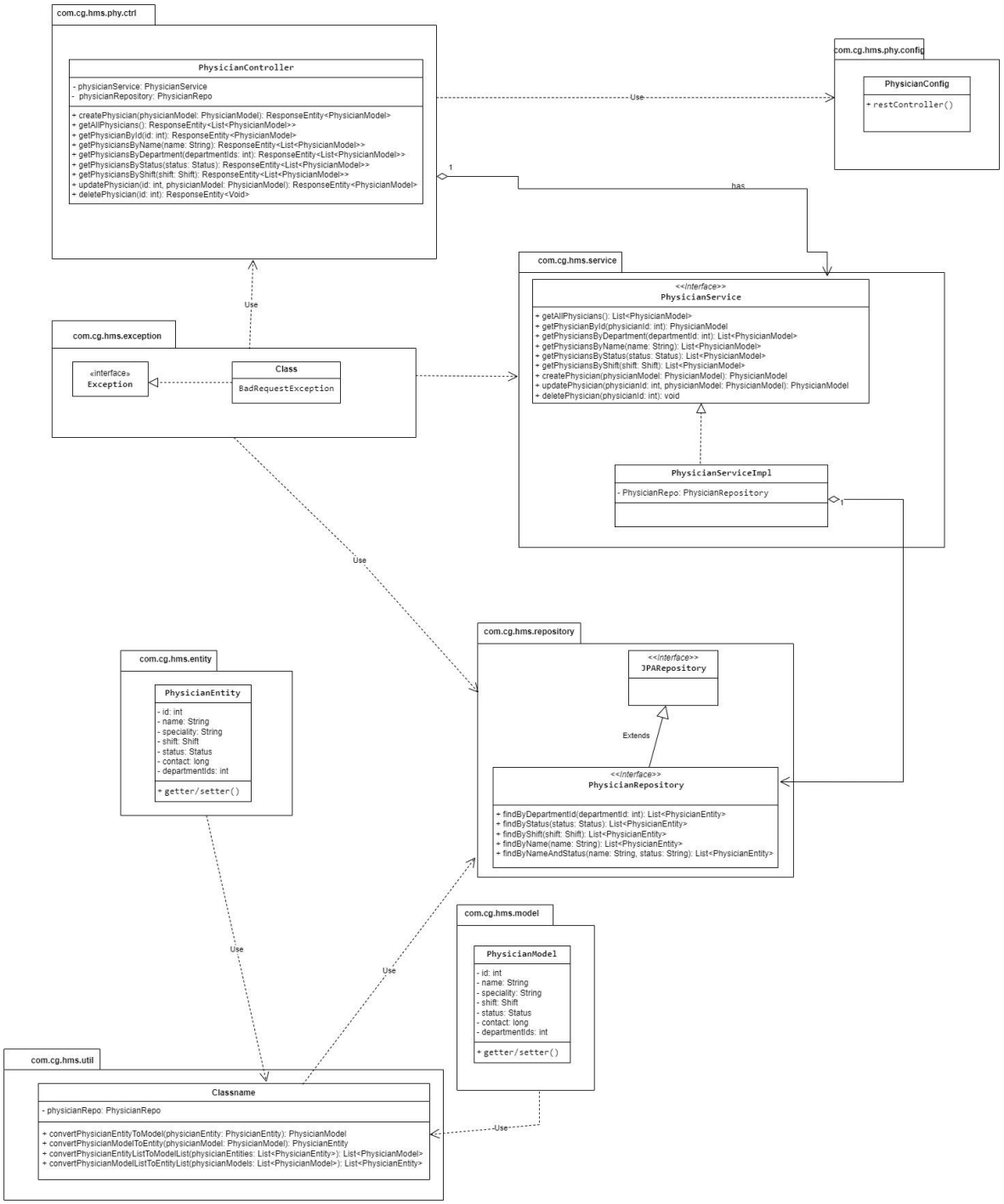


- # Class Diagram –

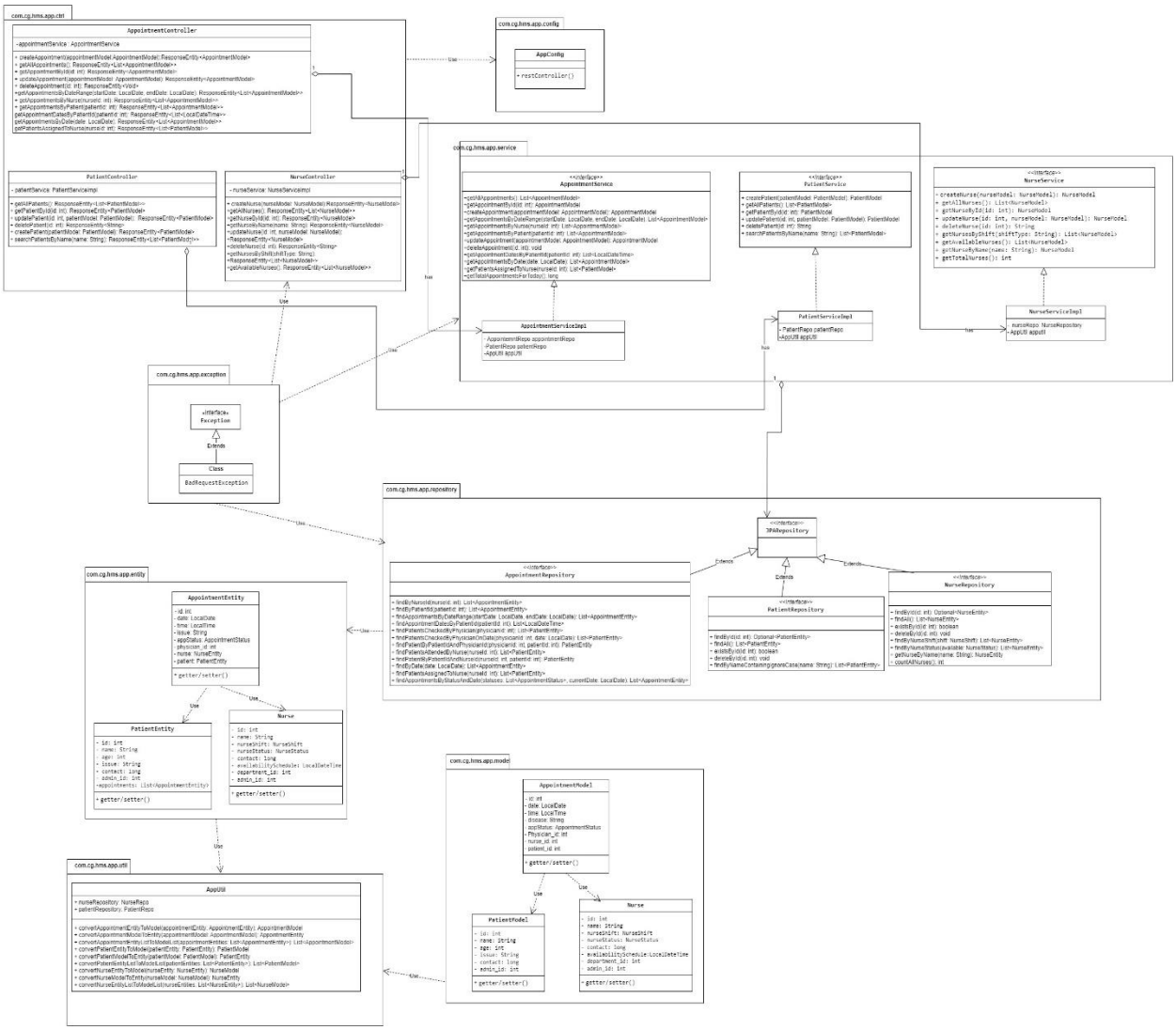
1. Admin Service



2. Physician Service



3. Appointment Service –



To organize the entities into the four microservices based on the database schema

---

1. Physician Microservice

Entities:

- Physician

Responsibilities:

- **Assign specialty**: Each physician has a specialty that determines the type of medical cases they handle.
- **Manage shift**: Physicians are assigned to specific shifts (e.g., Morning, Afternoon, Night).
- **Assign department**: Physicians are linked to a department (e.g., Emergency, Neurology).
- **Manage status**: The status can be updated (Available ,Unavailable).

2. Appointment Microservice

Entities:

- Appointment
- Nurse
- Patient

Responsibilities:

- S**chedule Appointment**: Create a new appointment, linking the patient, physician, and nurse to the appointment with a set date and time.
- **Cancel Appointment**: Cancel an appointment and update the appointment status accordingly.
- **Track Appointment Status**: Track and update the status of an appointment, including "Scheduled," "Completed," "Cancelled".
- **Assign Shift**: Nurses are assigned to specific shifts (Morning, Afternoon, or Night).
- **Track Nurse Availability**: Ensure that nurses are available during scheduled appointments and can be linked to appointments accordingly.

- **Create Patient Record**: Admin creates and stores patient records. These records are linked to the AdminService for authentication and user data.
- **Assign to Appointments**: Patients are assigned to appointments based on their medical condition and physician availability.

---

3. **Admin Microservice –**

Entities:

- User
- Department

Responsibilities –

- Admin creates new users in the system (physicians, nurses, patients, or other admin users).
- **Update User**: Admin can update user details (e.g., name, email, role, department).
- **Delete User**: Admin can deactivate or delete users from the system.
- **Assign Roles**: Admin assigns specific roles (e.g., ADMIN) to users.
- **Manage User Status**: Admin can set user status to 'Available' or 'Unavailable' depending on their activity.
- **Create Department**: Admin can create new departments within the hospital (e.g., Cardiology, Neurology).
- **Assign Department to Users**: Admin assigns users (like physicians, nurses) to specific departments.
- **Update Department**: Admin can update department details (e.g., name, head of the department).
- **Delete Department**: Admin can remove or deactivate departments, ensuring users are reassigned if needed.

- **Database for all four services -**

Database/ Database IDE used – MySQL Workbench

1. Admin Microservice

```
CREATE DATABASE IF NOT EXISTS adminservice;
USE adminservice;
CREATE TABLE IF NOT EXISTS users (
 id INT AUTO_INCREMENT PRIMARY KEY,
 name VARCHAR(255) NOT NULL,
 email VARCHAR(255) NOT NULL UNIQUE,
 role ENUM('admin', 'physician', 'nurse', 'patient') NOT NULL,
 status ENUM('active', 'inactive') DEFAULT 'active',
 department VARCHAR(100)
);
-- Departments Table
CREATE TABLE IF NOT EXISTS departments (
id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(100) NOT NULL
);
```

2. Physician Microservice

```
CREATE DATABASE IF NOT EXISTS physicianservice;
USE physicianservice;

--physicians table

CREATE TABLE IF NOT EXISTS physicians (
   id INT AUTO_INCREMENT PRIMARY KEY,
   user_id INT NOT NULL,  -- Reference to users in Admin Service
   specialty VARCHAR(100) NOT NULL,
   shift ENUM('Morning', 'Afternoon', 'Night') NOT NULL,
   department VARCHAR(100) NOT NULL,
```

```
    status ENUM('active', 'inactive') DEFAULT 'active',
    FOREIGN KEY (user_id) REFERENCES adminservice.users(id)
);
```

3. Appointment Microservice

```
CREATE DATABASE IF NOT EXISTS appointmentservice;
USE appointmentservice;

-- Patients Table

CREATE TABLE IF NOT EXISTS patients (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    issue VARCHAR(255) NOT NULL,
    status ENUM('active', 'inactive') DEFAULT 'active',
    FOREIGN KEY (user_id) REFERENCES adminservice.users(id)
);

-- Nurses Table

CREATE TABLE IF NOT EXISTS nurses (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,  -- Reference to users in Admin Service
    shift ENUM('Morning', 'Afternoon', 'Night') NOT NULL,
    department VARCHAR(100) NOT NULL,
    status ENUM('active', 'inactive') DEFAULT 'active',
    FOREIGN KEY (user_id) REFERENCES adminservice.users(id)
);

-- Appointments Table

CREATE TABLE IF NOT EXISTS appointments (
    id INT AUTO_INCREMENT PRIMARY KEY,
    patient_id INT NOT NULL,
    physician_id INT NOT NULL
    nurse_id INT NOT NULL,
    date DATE NOT NULL,
    time TIME NOT NULL,
    status ENUM('Scheduled', 'Completed', 'Cancelled', 'Rescheduled') DEFAULT
'Scheduled',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    FOREIGN KEY (patient_id) REFERENCES appointmentservice.patients(id),
    FOREIGN KEY (physician_id) REFERENCES physicianservice.physicians(id),
    FOREIGN KEY (nurse_id) REFERENCES appointmentservice.nurses(id)
);
```

- **Key Considerations:**

1. Data Integrity and Consistency

- Foreign Keys: Use foreign keys to maintain relationships (e.g., patient_id, physician_id).
- Transaction Management: Ensure consistency across services using event-driven architecture or sagas.
- Data Validation: Validate incoming data to avoid invalid entries.

2. Service Intercommunication

- RESTful APIs: Each service exposes APIs for communication.
- API Gateway: Use an API Gateway for routing and cross-cutting concerns like authentication.
- Error Handling: Implement clear error messages and HTTP status codes.

3. Authentication and Authorization

- JWT Authentication: Use JWT for secure, stateless authentication across services.
- Role-Based Access Control (RBAC): Ensure proper roles (Admin, Physician, Nurse, Patient) have access to only the relevant data.

4. Service Dependency and Decoupling

- Loose Coupling: Each service should function independently.
- Service Discovery: Implement service discovery for dynamic service location.

5. Data Storage and Management

- Database per Service: Each service should have its own database for autonomy.
- Schema Evolution: Manage schema changes and migrations independently for each service.

➢ Endpoints for all three services-

1. Admin Microservice
2. Physician Microservice
3. Appointment Microservice

1. Admin Service :

HTTP Method	Endpoint	Description
POST	/admin/user/register	Create a new admin user
POST	/admin/user/login	Login an admin user
PUT	/admin/user/update/{id}	Update details of a specific admin user
DELETE	/admin/user/{id}	Delete a specific admin user
GET	/admin/user/	Get all admin users
GET	/admin/user/{id}	Get details of a specific admin user
GET	/admin/user/username/{username}	Get admin users by username
GET	/admin/user/email/{email}	Get an admin user by email
GET	/admin/appointments/{appointmentId}/physician	Get the physician assigned for an appointment
GET	/admin/appointments/{physicianId}	Get appointments for a physician
GET	/admin/patients/{physicianId}	Get patients assigned to a physician
POST	/admin/appointment/create	Create a new appointment
POST	/admin/department/	Create a new department
PUT	/admin/department/{id}	Update an existing department
DELETE	/admin/department/{id}	Delete a specific department

GET	/admin/departments/	Get a list of all departments
GET	/admin/department/{id}	Get details of a specific department
GET	/admin/department/name/{name}	Get a department by its name

2. Physician Service

HTTP Method	Endpoint	Description
POST	/physician/	Create a new physician
GET	/physician/	Get all physicians
GET	/physician/{id}	Get details of a specific physician
GET	/physician/byName/{name}	Get physicians by their name
GET	/physician/byDepartment/{departmentIds}	Get physicians by department ID
GET	/physician/byStatus/{status}	Get physicians by their status (Available/Unavailable)
GET	/physician/byShift/{shift}	Get physicians by their shift (Morning, Afternoon, Night)
PUT	/physician/{id}	Update details of a specific physician
DELETE	/physician/{id}	Delete a specific physician

3. Appointment Service

Here is a combined API table for the **Appointment**, **Nurse**, and **Patient** services. Since they are all part of the same service, I've grouped the APIs logically based on their respective functionalities.

HTTP Method	Endpoint	Description
AppointmentAPIs		
POST	/app/appointments/	Create a new appointment

GET	/app/appointments/	Get all appointments
GET	/app/appointments/{id}	Get appointment by ID
PUT	/app/appointments/	Update an existing appointment
DELETE	/app/appointments/{id}	Delete appointment by ID
GET	/app/appointments/date-range	Get appointments within a date range
GET	/app/appointments/nurse/{nurseId}	Get appointments by Nurse ID
GET	/app/appointments/patient/{patientId}	Get appointments by Patient ID
GET	/app/appointments/patient/{patientId}/appointments-dates	Get appointment dates by Patient ID
GET	/app/appointments/date/{date}	Get appointments by specific date
GET	/app/appointments/nurse/{nurseId}/patients	Get patients assigned to a specific nurse
Nurse APIs		
POST	/app/nurses/	Create a new nurse
GET	/app/nurses/	Get all nurses
GET	/app/nurses/{id}	Get nurse by ID
GET	/app/nurses/name/{name}	Get nurse by name
PUT	/app/nurses/{id}	Update an existing nurse
DELETE	/app/nurses/{id}	Delete a nurse by ID

GET	/app/nurses/by-shift	Get nurses by shift type
GET	/app/nurses/available	Get available nurses
GET	/app/nurses/count	Get total nurses count
Patient APIs		
POST	/app/patients/	Create a new patient
GET	/app/patients/	Get all patients
GET	/app/patients/{id}	Get patient by ID
PUT	/app/patients/{id}	Update a patient by ID
DELETE	/app/patients/{id}	Delete patient by ID
GET	/app/patients/search	Search patients by name