**🏥 Case Study: Microservices-Based Architecture for a Hospital Information System (HIS)**

**1. Overview**

This case study outlines the architecture and design of a **Hospital Information System (HIS)** built using **Spring Boot Microservices**, integrating secure, scalable, and modular services to manage **patients**, **doctors**, and **clinical operations**. The system employs **Spring Cloud Config**, **Eureka Service Discovery**, **JWT-based authentication**, and a layered architecture approach using **Spring Data JPA**, **Swagger**, and best practices for distributed systems.

**2. Architectural Goals**

* Decouple hospital functional domains into independently deployable microservices.
* Ensure secure, stateless communication using **JWT**.
* Centralize configuration and enable dynamic service discovery.
* Maintain domain-driven boundaries through isolated persistence layers.
* Provide API discoverability and contract validation through **Swagger/OpenAPI**.

**3. Service Decomposition**

The HIS is decomposed into the following microservices:

| **Microservice** | **Responsibility** |
| --- | --- |
| Patient Service | Manages patient registration, demographics, medical history. |
| Doctor Service | Maintains doctor credentials, availability, and specializations. |
| Appointment Service | Handles doctor-patient scheduling and appointment tracking. |
| Authentication Service | Provides login, role-based authentication, and JWT issuance. |
| API Gateway | Central entry point for all clients with routing and security. |
| Config Server | Externalized configuration for all services. |
| Eureka Discovery | Service registration and discovery. |

Each service is designed to follow **bounded context** principles and owns its respective data store, implementing the **Database-per-Service** pattern.

**🧍‍♂️ Patient Properties**

These describe the personal and medical details of a patient:

| **Property Name** | **Data Type** | **Description** |
| --- | --- | --- |
| patient\_id | String / Integer | Unique identifier for the patient |
| first\_name | String | Patient's first name |
| last\_name | String | Patient's last name |
| date\_of\_birth | Date | Date of birth |
| gender | String | Male / Female / Other |
| contact\_number | String | Phone number |
| email | String | Email address |
| address | String | Home address |
| blood\_type | String | e.g., A+, O-, etc. |
| medical\_history | Text / JSON | Past illnesses, allergies, conditions |
| emergency\_contact | String | Name and number of emergency contact |
| insurance\_info | Text / JSON | Insurance provider and policy details |

**👨‍⚕️ Doctor Properties**

These define the characteristics and qualifications of a doctor:

| **Property Name** | **Data Type** | **Description** |
| --- | --- | --- |
| doctor\_id | String / Integer | Unique identifier for the doctor |
| first\_name | String | Doctor's first name |
| last\_name | String | Doctor's last name |
| specialization | String | Area of medical expertise (e.g., Cardiology) |
| license\_number | String | Medical license identifier |
| email | String | Email address |
| phone\_number | String | Contact number |
| years\_of\_experience | Integer | Number of years in practice |
| available\_days | Array / JSON | Days of the week they are available |
| clinic\_address | String | Location of clinic or hospital |

**📅 Appointment Properties**

These detail a scheduled meeting between a patient and a doctor:

| **Property Name** | **Data Type** | **Description** |
| --- | --- | --- |
| appointment\_id | String / Integer | Unique ID for the appointment |
| patient\_id | String / Integer | Reference to the patient |
| doctor\_id | String / Integer | Reference to the doctor |
| appointment\_date | Date | Date of the appointment |
| appointment\_time | Time | Scheduled time |
| reason | Text | Reason for the visit |
| status | String | e.g., Scheduled, Completed, Cancelled |
| notes | Text | Doctor's notes or follow-up instructions |
| created\_at | DateTime | When the appointment was booked |

**4. Security Architecture**

**JWT-Based Authentication**

* **Spring Security** with JWT is used to implement stateless authentication.
* The Authentication Service validates user credentials and issues JWT tokens.
* Roles are mapped as: PATIENT, DOCTOR, and ADMIN.
* Authorization is enforced via @PreAuthorize annotations in business methods and controllers.

**Flow**

1. User authenticates via /auth/login.
2. JWT is issued and passed in HTTP headers.
3. Gateway verifies the JWT before routing the request to downstream services.

**5. Infrastructure Components**

**5.1. Eureka Discovery Server**

* Enables dynamic registration and discovery of services.
* Eliminates hard-coded service locations.
* Supports client-side load balancing with **Spring Cloud LoadBalancer**.

**5.2. Spring Cloud Config Server**

* Centralized configuration managed via a Git repository.
* Each service fetches configuration at runtime based on environment profiles.
* Secure secrets externalized using placeholders or integration with secret managers.

**6. Layered Service Architecture**

Each microservice adheres to a **three-tier architecture**:

* **Controller Layer**: REST interface with input validation and exception handling.
* **Service Layer**: Encapsulates business logic.
* **Repository Layer**: JPA-based data persistence.

**7. API Documentation**

Swagger/OpenAPI is integrated for each microservice using **springdoc-openapi**.

* Exposes endpoints and schemas at:  
  http://<host>:<port>/swagger-ui/index.html
* Facilitates contract-first development and post-deployment testing.

**8. Inter-Service Communication**

* Services communicate **synchronously** via **Feign Clients**, using logical names from Eureka.
* DTOs are used to maintain encapsulation across services.

**9. API Gateway Design**

The API Gateway (Spring Cloud Gateway) provides:

* Centralized routing for all services.
* JWT token verification and role validation at the entry point.
* Cross-cutting concerns (logging, CORS, rate limiting).

**10. Sample Use Case: Schedule an Appointment**

**Workflow:**

1. **Patient logs in** → JWT issued.
2. Patient requests to book an appointment → Gateway verifies JWT.
3. Gateway routes to Appointment Service.
4. Appointment Service invokes Patient Service and Doctor Service via Feign to validate entities.
5. Appointment is created and persisted.