Problem Description: Network Resource Management System

You are required to develop a **Producer–Consumer system** using **Spring Boot** where the Producer exposes REST APIs for managing network resources, and the Consumer consumes these APIs. The goal is to simulate a real-world network domain service with advanced REST API concepts, content negotiation, internationalization, and proper API practices.

Scenario

Your organization manages a set of network devices (e.g., routers, switches, firewalls) and their configurations. The Producer service acts as the **Network Resource Management API**, and the Consumer service acts as a **client dashboard or automation tool** that consumes this API.

Each **network resource** has the following attributes:

Field	**Type**	**Description**
id	Long	Unique identifier
name	String	Device name
type	String	Device type (Router, Switch, etc.)
ipAddress	String	Device IP address
location	String	Physical or logical location
status	String	Active, Inactive, Maintenance
createdDate	LocalDate	Date of registration

Producer Module Requirements

Participants need to implement:

1. **CRUD APIs** for network resources:
    - GET /api/v1/resources – List all resources
    - GET /api/v1/resources/{id} – Get a specific resource
    - POST /api/v1/resources – Create a new resource (idempotent, must return **Location** header)
    - PUT /api/v1/resources/{id} – Update resource
    - DELETE /api/v1/resources/{id} – Delete resource
2. **Versioning**:
    - Implement **URL-based API versioning**, e.g., /api/v1/resources.

3. **Content Negotiation**:
    - Support application/json and application/xml responses based on Accept header.
4. **Internationalization (i18n)**:
    - Messages and error responses must be translatable into **English, Hindi, French**.
5. **Exception Handling**:
    - Handle not-found resources and invalid requests gracefully using @ControllerAdvice.
6. **HATEOAS Links**:
    - Include self-links and related links in responses where applicable.
7. **Dynamic Filtering**:
    - Allow clients to request only selected fields using a query parameter (e.g., /api/v1/resources?fields=id,name).
8. **Swagger/OpenAPI Documentation**:
    - Document all APIs using Swagger or Springdoc.
9. **Async Processing**:
    - Demonstrate at least one asynchronous API using @Async.
10. **Lifecycle Methods**:
    - Use @PostConstruct to initialize data and @PreDestroy to perform cleanup.
11. **Persistence**:
    - Use **H2 in-memory database** with Spring Data JPA.
    - Log all database operations.

Consumer Module Requirements

Participants need to implement:

1. A Spring Boot REST client using **RestTemplate** or **RestClient** to consume the Producer APIs.
2. Demonstrate:
    - GET all resources
    - GET by ID
    - POST new resource
    - Update resource
3. Use **headers from the producer**, such as X-Request-ID.
4. Demonstrate **content negotiation** and **logging** of responses.

5. Handle errors received from the Producer gracefully.
6. Include a small console or REST interface to display the data fetched from the Producer.

Additional Requirements

- **Idempotent POST**: Ensure that repeated POSTs with the same X-Request-ID do not create duplicate resources.
- **Proper Logging**: Log API requests/responses, database operations, and async processing.
- **Validation**: Validate IP addresses, device types, and mandatory fields.
- **Optional Challenge**: Include a simple **filtering or sorting mechanism** on GET requests (e.g., ?type=Router&status=Active).