**20. How to Develop API Gateway in Java Spring Boot Microservices?**

To **develop an API Gateway in Java Spring Boot for microservices**, you typically use **Spring Cloud Gateway**, which acts as a single entry point for routing requests to various microservices, handling concerns like authentication, logging, rate limiting, and more.

**✅ Step-by-Step Guide to Build an API Gateway using Spring Cloud Gateway**

**1. Create a Spring Boot Project (API Gateway)**

**Dependencies to include:**

* Spring Cloud Gateway
* Spring Boot DevTools
* Spring Web
* (Optional) Spring Cloud Netflix Eureka Client if using service discovery

**2. Add Dependencies (pom.xml)**

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-gateway</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>2021.0.8</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

**3. Configure Application Properties (application.yml)**

server:

port: 8080

spring:

application:

name: api-gateway

cloud:

gateway:

routes:

- id: user-service

uri: lb://USER-SERVICE

predicates:

- Path=/user/\*\*

- id: order-service

uri: lb://ORDER-SERVICE

predicates:

- Path=/order/\*\*

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka/

lb:// indicates service discovery via Eureka.

**4. Enable Eureka Client**

@SpringBootApplication

@EnableDiscoveryClient

public class ApiGatewayApplication {

public static void main(String[] args) {

SpringApplication.run(ApiGatewayApplication.class, args);

}

}

**5. Add Filters (Optional)**

Custom filters for:

* Authentication
* Logging
* Rate limiting

**Example: Logging Filter**

@Component

public class LoggingFilter implements GlobalFilter, Ordered {

@Override

public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {

System.out.println("Incoming request path: " + exchange.getRequest().getPath());

return chain.filter(exchange);

}

@Override

public int getOrder() {

return -1; // priority

}

}

**✅ Key Features You Can Add**

| **Feature** | **How to Achieve** |
| --- | --- |
| **Routing** | spring.cloud.gateway.routes |
| **Service Discovery** | Eureka (lb://service-name) |
| **Authentication** | Use filters or integrate with Keycloak/OAuth2 |
| **Rate Limiting** | Bucket4j, Resilience4j, Redis-based |
| **Load Balancing** | Built-in with Spring Cloud LoadBalancer |
| **Monitoring** | Spring Boot Actuator + Prometheus/Grafana |

**🎯 Summary**

* Spring Cloud Gateway makes it easy to build a production-grade API Gateway in Spring Boot.
* It supports routing, filtering, service discovery, and resilience out of the box.
* You can extend it with custom filters and integrate security providers.