API-GETEWAY

Netflix Zuul Microservice with Spring Boot

API-GETEWAY

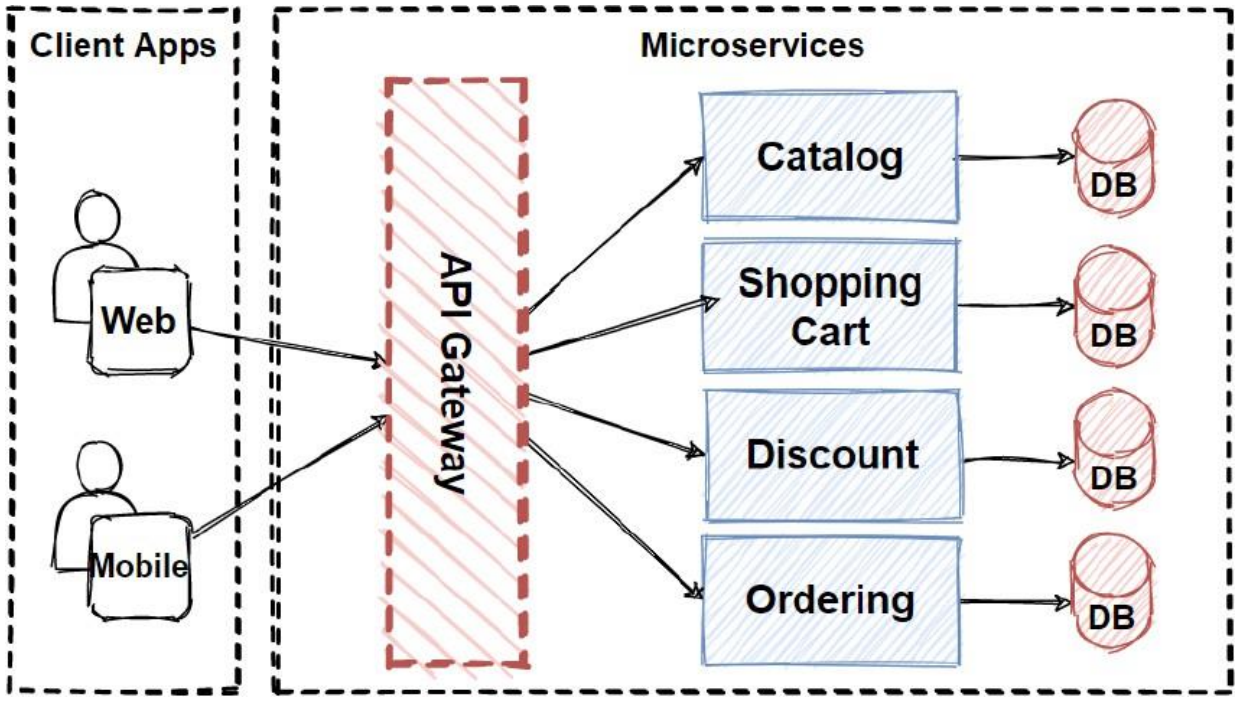
In a microservices application, there are many small services running separately each handling a specific task like user login, payments, or showing product details. Now, imagine a user wants to use your application. Instead of the user directly calling each microservice, we give them **one single entry point**. That entry point is called the **API Gateway**.

Netflix Zuul is a tool used to create this API Gateway in a Spring Boot project. It acts like a **smart gate** or **traffic controller** that receives all user requests and then sends them to the correct microservice behind the scenes. It can also add security, log requests, and handle errors.

Using **Netflix Zuul**, we can make our system simpler, more secure, and easier to manage.

Pointwise Explanation (Very Simple)

1. **Microservices** = Small parts of a big application, each doing one job (like login, payment, order, etc.).
2. **Problem** = User has to call each microservice directly — messy and hard to manage.
3. **Solution** = Use an **API Gateway** — one place where all user requests come in.
4. **Netflix Zuul** = A tool that helps create the API Gateway in a Spring Boot project.
5. **What Zuul Does**:
    - Receives requests from users
    - Sends them to the correct microservice
    - Returns the response to the user
    - Can also handle logging, authentication, and errors
6. **Why Use Zuul**:
    - Easier to manage microservices
    - More secure (you can add security rules in one place)
    - Better control over traffic
7. **Spring Boot + Zuul** = Simple way to build and run an API Gateway quickly.

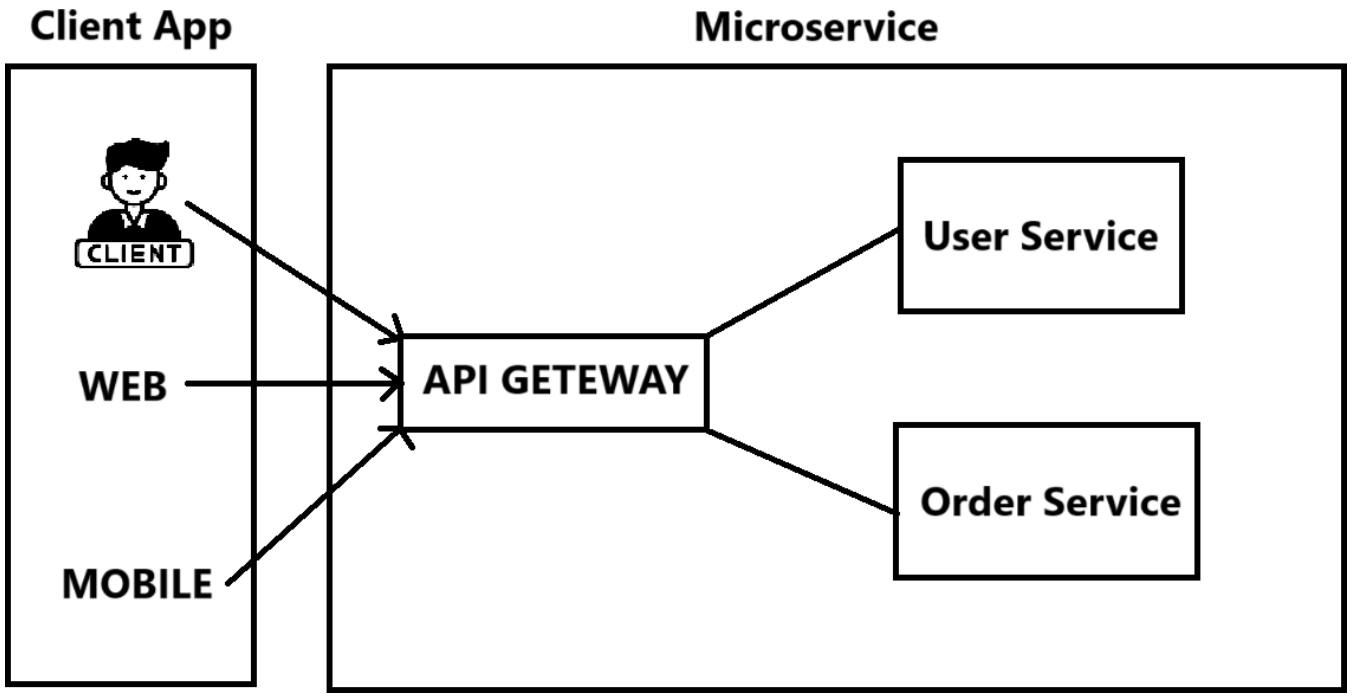


Netflix Zuul API Gateway Example with Spring Boot

We have two microservices,

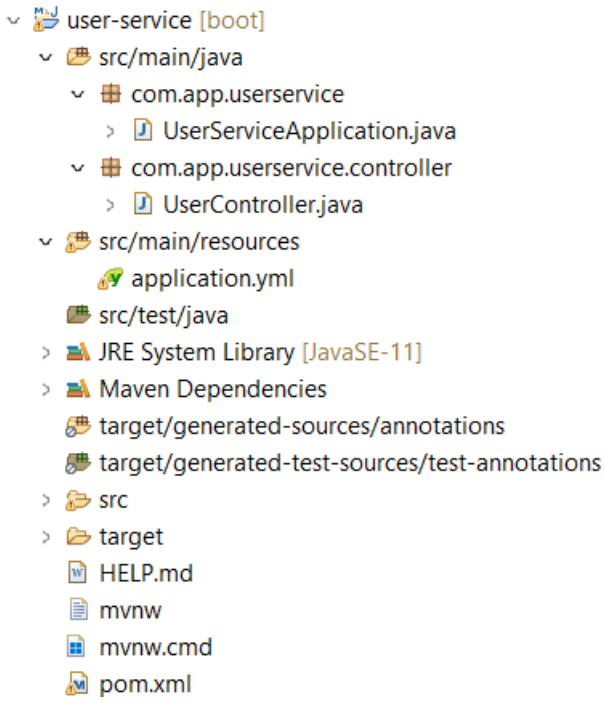
1. **User-Service**
2. **Order-Service,**

integrated with an **API-Gateway**



1. User Service:

Project Structure



Class: UserServiceApplication.Java

```java
package com.app.userservice;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class UserServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(UserServiceApplication.class, args);
    }

}
```

Class : UserController

```java
package com.app.userservice.controller;

import org.springframework.web.bind.annotation.GetMapping;

@RestController
@RequestMapping("/users")
public class UserController {

    @GetMapping("/{id}")
    public Map<String, Object> getUser(@PathVariable String id) {
        Map<String, Object> user = new HashMap<>();
        user.put("id", id);
        user.put("name", "User " + id);
        user.put("email", "user" + id + "@example.com");
        user.put("service", "user-service");
        return user;
    }

    @GetMapping
    public Map<String, Object> getAllUsers() {
        Map<String, Object> response = new HashMap<>();
        response.put("message", "All users from user-service");
        response.put("service", "user-service");
        response.put("port", "8081");
        return response;
    }
}
```

YAML: application.yml

application.yml

```yaml
server:
  port: 8081

spring:
  application:
    name: user-service

logging:
  level:
    com.example.userservice: DEBUG
```

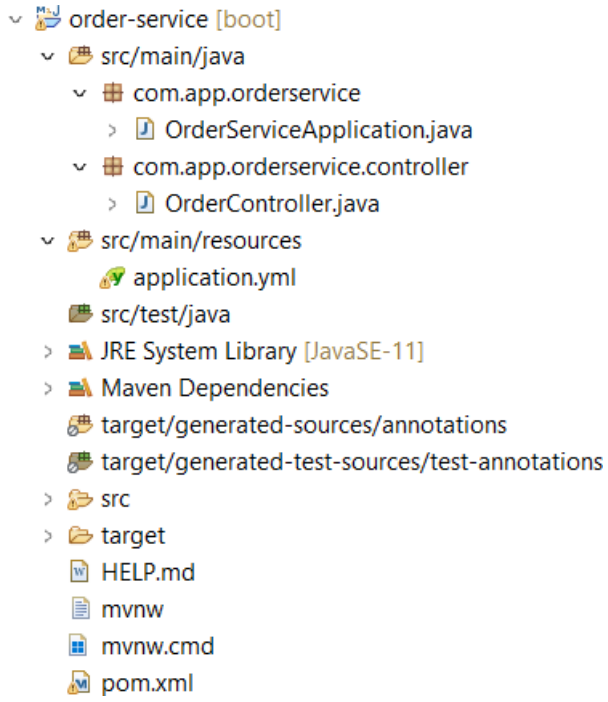
Pom.xml

```
http://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation with catalog)
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>user-service</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.12.RELEASE</version>
        <relativePath/>
    </parent>

    <properties>
        <java.version>11</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

2. Order Service

Project Structure



Class : OrderServiceApplication.java

OrderServiceApplication.java

```java
package com.app.orderservice;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class OrderServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(OrderServiceApplication.class, args);
    }

}
```

Class: OrderController.java

OrderController.java

```java
package com.app.orderservice.controller;

import org.springframework.web.bind.annotation.GetMapping;

@RestController
@RequestMapping("/orders")
public class OrderController {

    @GetMapping("/{id}")
    public Map<String, Object> getOrder(@PathVariable String id) {
        Map<String, Object> order = new HashMap<>();
        order.put("id", id);
        order.put("product", "Product " + id);
        order.put("amount", 100.00);
        order.put("service", "order-service");
        return order;
    }

    @GetMapping
    public Map<String, Object> getAllOrders() {
        Map<String, Object> response = new HashMap<>();
        response.put("message", "All orders from order-service");
        response.put("service", "order-service");
        response.put("port", "8082");
        return response;
    }
}
```

YAML: application.yml

application.yml

```yaml
server:
  port: 8082

spring:
  application:
    name: order-service

logging:
  level:
    com.example.orderservice: DEBUG
```

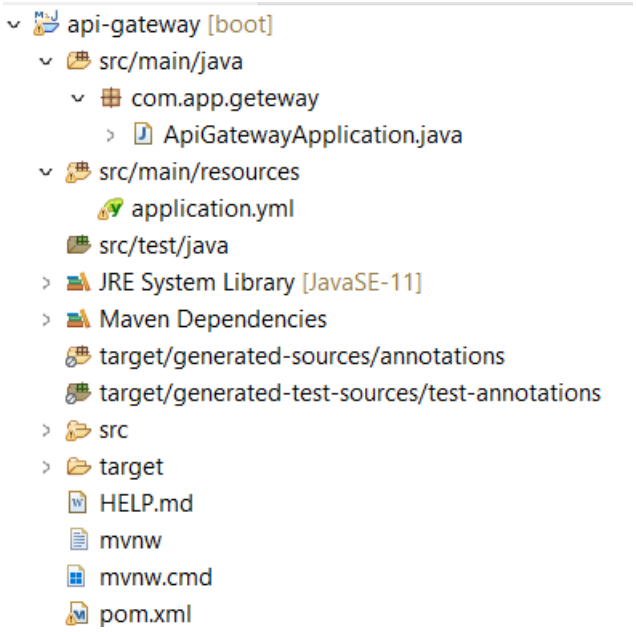
Pom.xml

order-service/pom.xml

```
http://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation with catalog)
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>order-service</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.12.RELEASE</version>
        <relativePath/>
    </parent>

    <properties>
        <java.version>11</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

After developing two services finally we have to develop API-GETEWAY, Lets do it…

API-GETEWAY



Class: ApiGatewayApplication.java

ApiGatewayApplication.java

```java
package com.app.geteway;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@EnableZuulProxy
public class ApiGatewayApplication {

    public static void main(String[] args) {
        SpringApplication.run(ApiGatewayApplication.class, args);
    }

}
```

YAML: application.yml

application.yml

```yaml
server:
  port: 8080

spring:
  application:
    name: api-gateway

# Zuul Configuration
zuul:
  routes:
    user-service:
      path: /users/**
      url: http://localhost:8081
      strip-prefix: false

    order-service:
      path: /orders/**
      url: http://localhost:8082
      strip-prefix: false

  # Global settings
  host:
    connect-timeout-millis: 20000
    socket-timeout-millis: 20000

  # Remove sensitive headers
  sensitive-headers: Cookie,Set-Cookie,Authorization

logging:
  level:
    com.netflix.zuul: DEBUG
    org.springframework.cloud.netflix.zuul: DEBUG
```

Pom.xml

api-gateway/pom.xml

```
http://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation with catalog)
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>api-gateway</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.12.RELEASE</version>
        <relativePath/>
    </parent>

    <properties>
        <java.version>11</java.version>
        <spring-cloud.version>Hoxton.SR12</spring-cloud.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
    </dependencies>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

How to Run

Start the services in order

 # Terminal 1 - Start User Service
cd user-service
mvn spring-boot:run

 # Terminal 2 - Start Order Service
cd order-service
mvn spring-boot:run

 # Terminal 3 - Start API Gateway
cd api-gateway
mvn spring-boot:run

Test the API Gateway

 # Access User Service through Gateway
http://localhost:8080/users
http://localhost:8080/users/123

 # Access Order Service through Gateway
http://localhost:8080/orders
http://localhost:8080/orders/456

How API Gateway Works

1. **Single Entry Point**: All client requests go through the API Gateway (port 8080)
2. **Request Routing**: Zuul routes requests based on path patterns:
    - /users/** → User Service (port 8081)
    - /orders/** → Order Service (port 8082)

Key Zuul Concepts

- **@EnableZuulProxy**: Enables Zuul proxy functionality
- **Routes**: Define how requests are mapped to backend services
- **Filters**: Pre, Route, Post, and Error filters for request processing
- **Load Balancing**: Built-in load balancing with Ribbon
- **Circuit Breaker**: Integration with Hystrix for fault tolerance

==========Thank you==========