



Instituto Politécnico Nacional

Escuela Superior de Cómputo



DESARROLLO DE SISTEMAS DISTRIBUIDOS

Práctica 7

Profesor: Chadwick Carreto Arellano

Grupo: 4CM11

Alumnos:

González Hipólito Miguel Ángel

Ponce López José Angel

Contents

1	Antecedentes	3
2	Planteamiento del problema.....	3
3	Materiales y Métodos empleados.....	3
3.1	Materiales Utilizados.....	3
4	Desarrollo	4
4.1	Proyecto base.....	4
4.2	Microservicio de estudiantes	5
4.3	Microservicio de grupos	6
4.4	Microservicio Eureka	8
4.5	Microservicio Gateway	9
5	Conclusiones	9

1 Antecedentes

El enfoque tradicional para el diseño de aplicaciones se centraba en la arquitectura monolítica, en que todos los elementos que pueden implementarse estaban contenidos en una sola aplicación. Este enfoque tiene sus desventajas: cuanto más grande es la aplicación, más difícil es solucionar los problemas que se presentan y agregar funciones nuevas rápidamente. En cambio, si las aplicaciones se diseñan con microservicios, se resuelven dichos problemas y se impulsa el desarrollo y la capacidad de respuesta.

Los microservicios son tanto un estilo de arquitectura como un modo de programar software. Con los microservicios, las aplicaciones se dividen en sus elementos más pequeños e independientes entre sí. A diferencia del enfoque tradicional y monolítico de las aplicaciones, en el que todo se compila en una sola pieza, los microservicios son elementos independientes que funcionan en conjunto para llevar a cabo las mismas tareas. Cada uno de esos elementos o procesos es un microservicio. Este enfoque de desarrollo de software valora el nivel de detalle, la sencillez y la capacidad para compartir un proceso similar en varias aplicaciones. Es un elemento fundamental de la optimización del desarrollo de aplicaciones hacia un modelo nativo de la nube.

2 Planteamiento del problema

En muchas instituciones educativas, la gestión de tareas escolares, estudiantes y su información asociada se realiza mediante sistemas monolíticos o incluso procesos manuales que dificultan la escalabilidad, mantenibilidad y evolución tecnológica. Estos enfoques centralizados provocan una alta dependencia entre módulos, lo que complica la implementación de nuevas funcionalidades, genera problemas de disponibilidad, y limita el desarrollo en paralelo por múltiples equipos.

Ante esta situación, surge la necesidad de diseñar e implementar una arquitectura basada en microservicios, donde cada componente (por ejemplo, gestión de estudiantes, tareas, calificaciones, etc.) funcione de forma independiente y se comunique a través de APIs bien definidas, coordinadas por un gateway. Esta arquitectura debe facilitar el mantenimiento, escalar por demanda, y permitir el desarrollo ágil de nuevas interfaces, como aplicaciones web modernas construidas con React.

Donde utilizaremos

3 Materiales y Métodos empleados

3.1 Materiales Utilizados

- **Lenguaje de programación:** Java

- **Entorno de desarrollo:** IntelliJ
- **Framework:** Spring boot
- **Sistema operativo:** Windows 11

4 Desarrollo

4.1 Proyecto base

Primero generamos un archivo pom.xml donde declaramos la versión a utilizar de spring boot, así como la versión de java y los microservicios que serán los módulos de nuestro sistema.

```
m pom.xml (microservice) x
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6         <groupId>org.springframework.boot</groupId>
7         <artifactId>spring-boot-starter-parent</artifactId>
8         <version>3.5.3</version>
9     </parent>
10
11     <groupId>com.microservice</groupId>
12     <artifactId>microservice</artifactId>
13     <version>0.0.1-SNAPSHOT</version>
14     <packaging>pom</packaging>
15
16     <name>microservice</name>
17     <description>Demo project for Spring Boot</description>
18
19     <modules>
20         <module>microservice-eureka</module>
21         <module>microservice-config</module>
22         <module>microservice-students</module>
23         <module>microservice-group</module>
24         <module>microservice-gateway</module>
25     </modules>
26
27     <properties>
28         <java.version>21</java.version>
29     </properties>
30
```

Crearemos 4 microservicios para que se comuniquen entre sí.

```
> microservice-eureka
> microservice-gateway
> microservice-group
> microservice-students
```

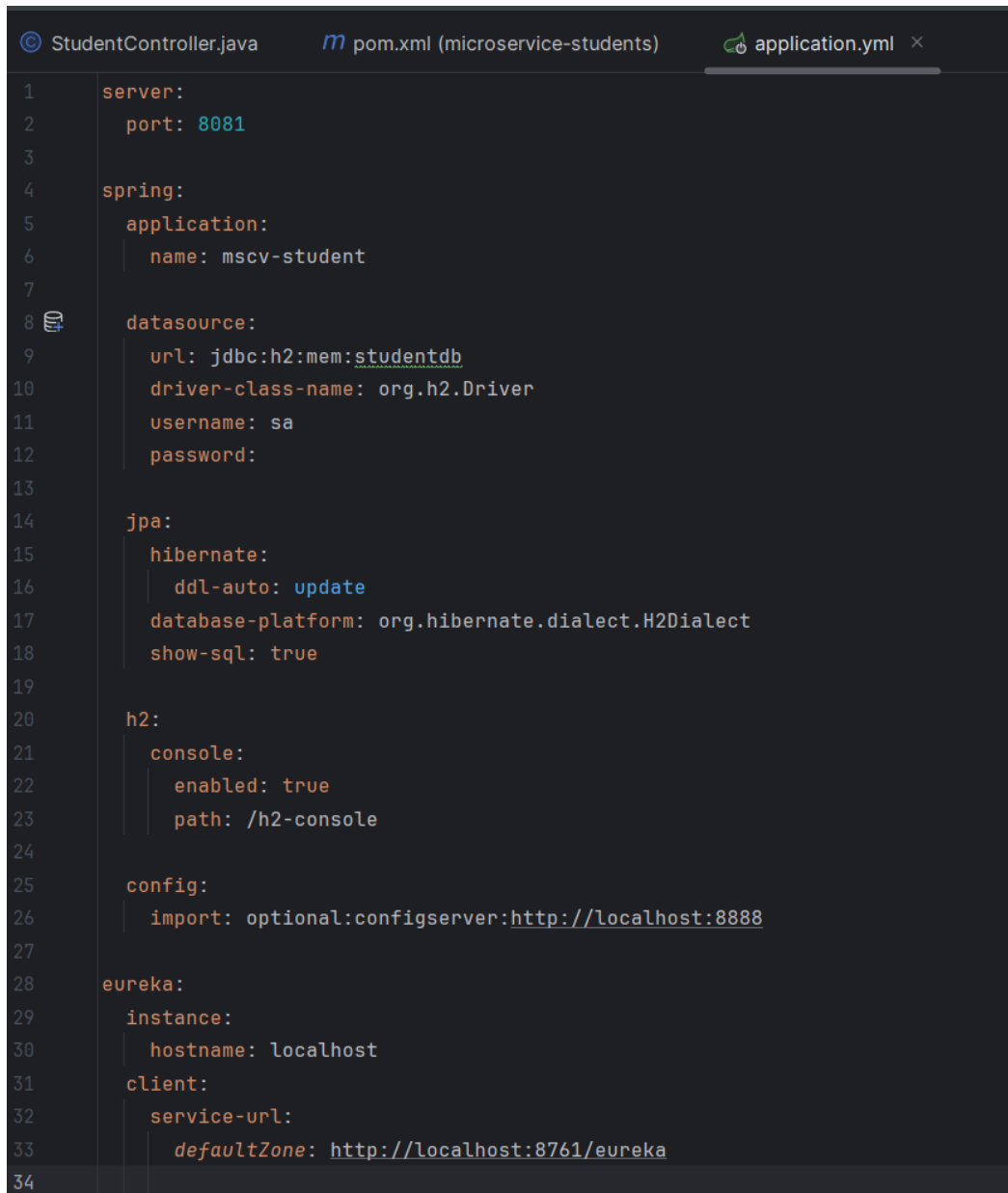
4.2 Microservicio de estudiantes

Es el **controlador REST del microservicio de estudiantes**. Expone un conjunto de endpoints HTTP para que otros servicios o clientes (como el frontend) puedan interactuar con el sistema y realizar operaciones CRUD sobre los estudiantes.

Este controlador **llama a IStudentService**, que a su vez maneja la lógica de negocio y conexión a la base de datos.

```
10  @RestController  Miguel Gonzalez Sydech
11  @RequestMapping("/api/student")
12  public class StudentController {
13
14      @Autowired
15      private IStudentService studentService;
16
17      @PostMapping("/create")  Miguel Gonzalez Sydech
18      @ResponseStatus(HttpStatus.CREATED)
19      public void saveStudent(@RequestBody Student student) {
20          studentService.save(student);
21      }
22
23      @GetMapping("/all")  Miguel Gonzalez Sydech
24      public ResponseEntity<?> findAll(){
25          return ResponseEntity.ok(studentService.findAll());
26      }
27
28      @GetMapping("/search/{id}")  Miguel Gonzalez Sydech
29      public ResponseEntity<?> findById(@PathVariable Long id) {
30          return ResponseEntity.ok(studentService.findById(id));
31      }
32
33      @GetMapping("/search-by-group/{groupId}")  Miguel Gonzalez Sydech
34      public ResponseEntity<?> findStudentsByIdGroup(@PathVariable Long groupId){
35          return ResponseEntity.ok(studentService.findById(groupId));
36      }
37
38  }
```

La configuración del microservicio es la siguiente:

The image shows a code editor with three tabs: 'StudentController.java', 'pom.xml (microservice-students)', and 'application.yml'. The 'application.yml' tab is active, displaying a YAML configuration file. The configuration includes settings for the server port (8081), Spring application name (mscv-student), a H2 database datasource (jdbc:h2:mem:studentdb), JPA Hibernate settings (ddl-auto: update, database-platform: org.hibernate.dialect.H2Dialect, show-sql: true), H2 console settings (enabled: true, path: /h2-console), a config server import (optional:configserver:http://localhost:8888), and Eureka client settings (instance: localhost, service-url: http://localhost:8761/eureka).

```
1  server:
2    port: 8081
3
4  spring:
5    application:
6      name: mscv-student
7
8    datasource:
9      url: jdbc:h2:mem:studentdb
10     driver-class-name: org.h2.Driver
11     username: sa
12     password:
13
14     jpa:
15       hibernate:
16         ddl-auto: update
17       database-platform: org.hibernate.dialect.H2Dialect
18       show-sql: true
19
20     h2:
21       console:
22         enabled: true
23         path: /h2-console
24
25     config:
26       import: optional:configserver:http://localhost:8888
27
28     eureka:
29       instance:
30         hostname: localhost
31       client:
32         service-url:
33           defaultZone: http://localhost:8761/eureka
34
```

4.3 Microservicio de grupos

Es el **controlador REST** del **microservicio de grupos**. También expone endpoints para crear y consultar grupos de estudiantes.

```

@RestController  Miguel Gonzalez Sydech
@RequestMapping("/api/groups")
public class GroupController {

    @Autowired
    private IGroupService groupService;

    @PostMapping("/create")  Miguel Gonzalez Sydech
    @ResponseStatus(HttpStatus.CREATED)
    public void SaveGroup(@RequestBody Group group) {
        groupService.saveGroup(group);
    }

    @GetMapping("/all")  Miguel Gonzalez Sydech
    public ResponseEntity<?> findAllGroups() {
        return ResponseEntity.ok(groupService.findAll());
    }

    @GetMapping("/search/{id}")  Miguel Gonzalez Sydech
    public ResponseEntity<?> findById(@PathVariable Long id) {
        return ResponseEntity.ok(groupService.findById(id));
    }

    @GetMapping("/searchStudentsByGroupId/{groupId}")  Miguel Gonzalez Sydech
    public ResponseEntity<?> findAllByGroupId(@PathVariable Long groupId){
        return ResponseEntity.ok(groupService.findStudentsByGroupId(groupId));
    }
}

```

Por otra parte se conecta, se genera un cliente que conecta al microservicio msvc-student en localhost:8080/api/student y permite invocar su endpoint:

```

@FeignClient(name = "msvc-student", url = "localhost:8080/api/student") 2 usages  Miguel Gonzalez Sydech
public interface StudentClient {

    @GetMapping("/search-by-group/{groupId}")  Miguel Gonzalez Sydech
    List<StudentDto> findAllStudentByGroup(@PathVariable Long groupId);
}

```

La configuración del microservicio es la siguiente, este se ejecuta en el puerto 8081

```
GroupController.java StudentByGroupResponse.java StudentClient.java
1 server:
2   port: 8082
3
4 spring:
5   application:
6     name: mscv-group
7   datasource:
8     url: jdbc:postgresql://localhost:5432/groups_db
9     driver-class-name: org.postgresql.Driver
10    username: postgres
11    password: mik3123
12
13   jpa:
14     hibernate:
15       ddl-auto: create
16       database: postgresql
17       dialect: org.hibernate.dialect.PostgreSQLDialect
18       show-sql: true
19   config:
20     import: optional:configserver:http://localhost:8888
21
22 eureka:
23   instance:
24     hostname: localhost
25   client:
26     service-url:
27     defaultZone: http://localhost:8761/eureka
```

Es decir, desde otro microservicio (como el de grupos), puedes llamar directamente a este método para **obtener estudiantes de un grupo**, sin preocuparte por los detalles de la llamada HTTP.

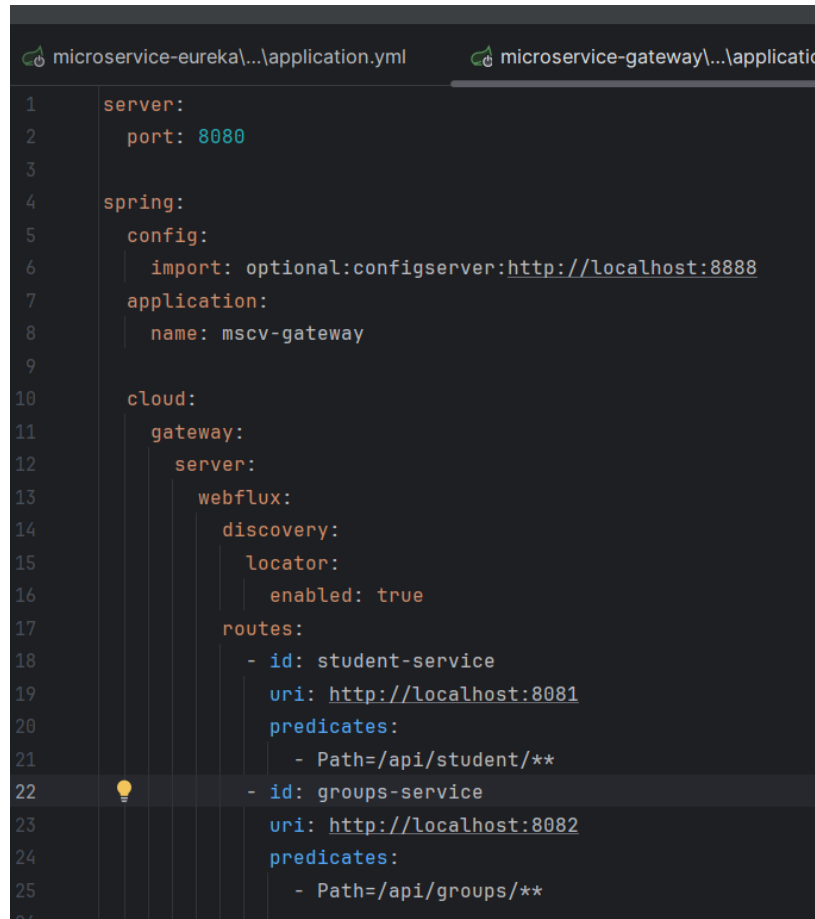
4.4 Microservicio Eureka

Este archivo configura un **servidor de registro Eureka**, el cual actúa como un "directorio" donde los microservicios se registran para poder descubrirse entre sí.

```
microservice-eureka\...\application.yml x microservice-gateway\...\application.yml
1 server:
2   port: 8761
3
4 spring:
5   application:
6     name: mscv-eureka
7   config:
8     import: optional:configserver:http://localhost:8888
9 eureka:
10   instance:
11     hostname: localhost
12   client:
13     register-with-eureka: false
14     fetch-registry: false
15     server-url:
16     defaultZone: http://localhost:${server.port}/eureka/
17
```


4.5 Microservicio Gateway

Este archivo configura un **API Gateway** utilizando Spring Cloud Gateway. Este componente centraliza las entradas al sistema y enruta peticiones hacia los microservicios correspondientes.



```
microservice-eureka\...\application.yml  microservice-gateway\...\applicatio
1  server:
2    port: 8080
3
4  spring:
5    config:
6      import: optional:configserver:http://localhost:8888
7    application:
8      name: mscv-gateway
9
10   cloud:
11     gateway:
12       server:
13         webflux:
14           discovery:
15             locator:
16               enabled: true
17           routes:
18             - id: student-service
19               uri: http://localhost:8081
20               predicates:
21                 - Path=/api/student/**
22             - id: groups-service
23               uri: http://localhost:8082
24               predicates:
25                 - Path=/api/groups/**
26
```

5 Conclusiones

Durante el desarrollo del sistema se implementó una arquitectura basada en microservicios utilizando el framework **Spring Boot**, lo que permitió dividir la aplicación en componentes independientes y especializados. Esta separación facilitó la gestión, el mantenimiento y la escalabilidad del sistema, permitiendo que cada microservicio evolucione de forma aislada y con responsabilidades bien definidas.

En la arquitectura se utilizaron dos componentes principales para la orquestación y descubrimiento de servicios:

- **Eureka Server (mscv-eureka):** permite el **registro y descubrimiento dinámico de microservicios**. Cada servicio, al iniciar, se registra automáticamente en Eureka, lo que elimina la necesidad de

gestionar manualmente direcciones IP fijas y facilita la comunicación entre servicios mediante nombres lógicos.

- **Spring Cloud Gateway (mscv-gateway):** actúa como **punto de entrada único** para todas las peticiones del cliente. A través de las rutas definidas en el archivo `application.yml`, el gateway enruta de manera dinámica las peticiones hacia los microservicios correspondientes, como el microservicio de estudiantes o el de grupos. Esta solución permite centralizar aspectos como la seguridad, el balanceo de carga y el control de acceso.

Asimismo, se utilizó **Feign Client** para facilitar la comunicación REST entre microservicios, simplificando el código y promoviendo una estructura más legible y mantenible. También se aplicó el principio de responsabilidad única en cada microservicio, separando controladores, servicios y repositorios, lo que contribuye a una arquitectura limpia y escalable.

Finalmente, la solución propuesta demuestra cómo los microservicios, acompañados de herramientas de descubrimiento y enrutamiento, pueden mejorar significativamente la modularidad y flexibilidad de aplicaciones orientadas a entornos distribuidos.