

proyecto final

Desarrollo Web Basado en Servicios y Componentes



12 de junio de 2019

universidad de almería

Escuela Superior de Ingeniería

Juan Antonio Rodríguez Baeza

Índice

[1. microservices.eureka.server 3](#_Toc11279155)

[2. Despliegue de la base de datos 5](#_Toc11279156)

[3. microservice.eureka.client.manageplayer 6](#_Toc11279157)

[3.1. Player.java 8](#_Toc11279158)

[3.2. PlayerService.java 8](#_Toc11279159)

[3.3. PlayerServiceImpl.java 9](#_Toc11279160)

[3.4. PlayerController.java 10](#_Toc11279161)

[3.5. Application.java (players) 11](#_Toc11279162)

[4. microservice.eureka.client.managegame 14](#_Toc11279163)

[4.1. Games.java 16](#_Toc11279164)

[4.2. GamesRepository.java 16](#_Toc11279165)

[4.3. GamesController.java 17](#_Toc11279166)

[4.4. Aplication.java (games) 18](#_Toc11279167)

[5. microservice.eureka.client. 20](#_Toc11279168)

[6. Segundo punto 21](#_Toc11279169)

Ilustraciones.

[Ilustración 1estructura eureka server. 3](#_Toc11279173)

[Ilustración 2 eureka server corriendo. 4](#_Toc11279174)

[Ilustración 3 UI de eureka server. 4](#_Toc11279175)

[Ilustración 4 diagrama de la base de datos. 5](#_Toc11279176)

[Ilustración 5 estructura microservicio manageplayer. 6](#_Toc11279177)

[Ilustración 6 time zone unrecognized, configuracion de MySQL. 11](#_Toc11279178)

[Ilustración 7 manageplayer corriendo 12](#_Toc11279179)

[Ilustración 8 UI eureka, microservicio manageplayer. 12](#_Toc11279180)

[Ilustración 9 estructura microservicio managegame 14](#_Toc11279181)

[Ilustración 10 manage games corriendo. 18](#_Toc11279182)

# microservices.eureka.server

Se trata del centro neurálgico del proyecto, Eureka Server es una aplicación que contiene la información sobre todas las aplicaciones de servicio al cliente. Cada microservicio se registrará en el servidor Eureka y el servidor Eureka conoce todas las aplicaciones cliente que se ejecutan en cada puerto y dirección IP. Eureka Server también se conoce como Discovery Server.

Nuestro servidor Eureka tiene la siguiente estructura:

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Ilustración 1estructura eureka server.

Vemos que sus recursos son:

* applicaction.yml, con el contenido.

server:

port: 8761

eureka:

server:

enable-self-preservation: false

client:

register-with-eureka: false

fetch-registry: false

* bootstrap.yml, con el contenido:

spring:

application:

name: eureka-server

En el primer fichero declaramos la configuración del servicio como es el puerto donde estará escuchando, además le estamos diciendo que no habilite el modo de auto preservación, ni que se registre a si mismo como microservicio.

Si corremos el servicio de Eureka desde la consola Boot dashboard.

Imagen que contiene captura de pantalla, monitor, negro

Descripción generada automáticamente

Ilustración 2 eureka server corriendo.

El servicio quedará levantado y escuchando en el puerto 8761, esperando a que los microservicios se registren.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Ilustración 3 UI de eureka server.

# Despliegue de la base de datos

Antes de continuar con los microservicios, vamos a realizar el despliegue de la base de datos, hemos escogido MySQL como sistema gestor de base de datos.

A continuación, mostramos el diagrama de nuestra base de datos.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Ilustración 4 diagrama de la base de datos.

Como se puede apreciar, las tablas principales “players” y “games” guardan relación a través de la tabla intermedia “player\_games” que simboliza que juegos tiene un usuario guardados como favoritos; por otro lado, la tabla “followings” guarda la relación existente entre jugadores seguidos y seguidores. Como aclaración, hemos establecido el DNI del jugador como tipo varchar(45) por la necesidad de validar el dato como si de un DNI real se tratase, esto lo veremos y explicaremos en su momento cuando veamos el microservicio data\_validator.

# microservice.eureka.client.manageplayer

Se trata del servicio que implementa la funcionalidad referida a los jugadores, vemos a continuación su estructura.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Ilustración 5 estructura microservicio manageplayer.

Y sus recursos:

* applicaction.yml, con el contenido.

server:

port: 50001

eureka:

instance:

hostname: localhost

instance\_id: ${eureka.instance.hostname}:${spring.application.name}:${spring.application.instance\_id:${random.value}}

client:

service-url:

default-zone: http://localhost:8761/

* bootstrap.yml, con el contenido:

spring:

application:

name: client-manageplayer

vemos que le hemos establecido un puerto fijo, el 50001, esto se debe a que al estar en entorno de desarrollo se hace bastante pesado estar siempre pendiente del puerto que Eureka le asigna aleatoriamente para realizar los testeos. En un entorno de producción lo ideal podría ser decirle que pida una asignación de puerto aleatoria de la siguiente forma

server:

port: ${PORT:${SERVER\_PORT:0}}

También debemos configurar las dependencias del proyecto, como por ejemplo la que hace referencia a MySQL, para ello vamos al fichero POM de maven, e incluimos:

* hibernate
* mysql

<?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>

  <parent>

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

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

    <version>2.1.4.RELEASE</version>

    <relativePath/> <!-- lookup parent from repository -->

  </parent>

  <groupId>dwsc</groupId>

  <artifactId>microservice.eureka.client.manageplayer</artifactId>

  <version>0.0.1-SNAPSHOT</version>

  <name>microservice.eureka.client.manageplayer</name>

…

    <dependency>

    <groupId>org.hibernate.javax.persistence</groupId>

    <artifactId>hibernate-jpa-2.1-api</artifactId>

    <version>1.0.0.Final</version>

    </dependency>

    <dependency>

     <groupId>mysql</groupId>

     <artifactId>mysql-connector-java</artifactId>

     <version>8.0.16</version>

    </dependency>

  </dependencies>

Para hacer funcionar este y todos los proyectos, hemos tenido que cambiar a la versión 2.1.4, como vemos en la muestra anterior. Al final de la muestra vemos las importaciones de las dependencias mencionadas.

Clases

## Player.java

Esta clase representa al modelo player:

package dwsc.microservice.eureka.client.manageplayer.domain;

import javax.xml.bind.annotation.XmlRootElement;

import javax.persistence.\*;

@*XmlRootElement*

@*Entity*

public class Player {

  @*Id*

  private *String* dni;

  private *String* name;

  private *String* surname;

  private *Integer* age;

## PlayerService.java

Se trata de la interfaz de este servicio, declara los funciones que va a realizar como sacar la lista de jugadores, o uno concreto, crear un jugador, etc.

package dwsc.microservice.eureka.client.manageplayer.service;

import java.util.ArrayList;

import dwsc.microservice.eureka.client.manageplayer.domain.Player;

public interface PlayerService {

  public *ArrayList*<*Player*> getPlayersFromDB();

  public *Player* getPlayerByDNIFromDB(*String* *dni*);

  public *ArrayList*<*Player*> getPlayerByNameFromDB(*String* *name*);

  public *ArrayList*<*Player*> getPlayerBySurnameFromDB(*String* *surname*);

  public *boolean* createPlayerInDB(*String* *dni*, *String* *name*, *String* *surname*, *int* *age*);

  public *boolean* deletePlayerByDNIFromDB(*String* *dni*);

  public *boolean* updatePlayerInDB(*String* *dni*, *String* *name*, *String* *surname*, *int* *age*);

}

## PlayerServiceImpl.java

Esta clase implementa la interfaz anterior, se encarga de darle la lógica correspondiente a las distintas funciones. Contiene una función especial, se trata de la conexión con la base de datos, en el extracto siguiente la vemos al principio de la clase.

package dwsc.microservice.eureka.client.manageplayer.service;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.ArrayList;

import org.springframework.stereotype.Service;

import dwsc.microservice.eureka.client.manageplayer.domain.Player;

@*Service*

public class PlayerServiceImpl implements *PlayerService*{…

  private *Connection* connect2DB() {…

  @*Override*

  public *ArrayList*<*Player*> getPlayersFromDB() {…

  @*Override*

  public *Player* getPlayerByDNIFromDB(*String* *dni*) {…

  @*Override*

  public *ArrayList*<*Player*> getPlayerByNameFromDB(*String* *name*){…

  @*Override*

  public *ArrayList*<*Player*> getPlayerBySurnameFromDB(*String* *surname*){…

  @*Override*

  public *boolean* createPlayerInDB(

*String* *dni*, *String* *name*, *String* *surname*, *int* *age*) {…

  @*Override*

  public *boolean* deletePlayerByDNIFromDB(*String* *dni*) {…

  @*Override*

  public *boolean* updatePlayerInDB(

*String* *dni*, *String* *name*, *String* *surname*, *int* *age*) {…

}

## PlayerController.java

El controlador del Proyecto es el que finalmente hace uso de las funciones anteriormente declaras e implementadas. Mostramos un ejemplo sencillo como es conseguir la lista de players

  // Mapping the path in the microservice to get all players

  @*RequestMapping*(value = "/", method = RequestMethod.GET)

  public @*ResponseBody* *ArrayList*<*Player*> getPlayers() {

    return playerService.getPlayersFromDB();

  }

Pero también tenemos que mencionar como esta clase hace uso de DiscoveryClient para buscar la funcionalidad del validador cuando necesita verificar el DNI del objeto player.

Se declara una propiedad privada de la calse del tipo DiscoveryClient

@*RestController*

public class PlayerController {

  @*Autowired*

  private *PlayerService* playerService;

  @*Autowired*

  private *DiscoveryClient* discoveryClient;

que luego usamos en el método createPlayer

  @*RequestMapping*(value = "/player/", method = RequestMethod.POST)

  public @*ResponseBody* *int* createPlayer(@*RequestParam*("dni") *String* *dni*,

      @*RequestParam*("name") *String* *name*,

      @*RequestParam*("surname") *String* *surname*,

      @*RequestParam*("age") *int* *age*) {

*boolean* validator = false;

*Map*<*String*, *String*> params = new *TreeMap*<*String*, *String*>();

    params.put("data", dni);

*List*<*ServiceInstance*> serviceList = discoveryClient.getInstances(

        "client-data\_validator");

    if(serviceList != null && serviceList.size() > 0) {

*URI* uri = serviceList.get(0).getUri();

*String* url = uri.toString() + "/{data}";

      if(uri != null) {

        validator = (new RestTemplate()).getForObject(url, Boolean.class, params);

      }

    }

    if(validator) {

*boolean* inserted = playerService.createPlayerInDB(dni, name, surname, age);

      if(inserted){

        return 1;

      }

      return 0;

    }

    return -1;

  }

Como vemos en el extracto anterior, el booleano “validador” almacena la respuesta recibida por el microservicio llamado “client-data-validator”

## Application.java (players)

Además, en la clase que contiene el método principal, tendremos que añadirle una anotación para que haga uso del descubridor de Eureka, como mostramos a continuación.

package dwsc.microservice.eureka.client.manageplayer;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@*SpringBootApplication*

@*EnableDiscoveryClient*

public class Application {

  public static *void* main(*String*[] *args*) {

    SpringApplication.run(Application.class, args);

  }

}

Levantamos el servicio, y se registrará en Eureka. Puede ser necesario revisar la configuración horaria del servicio de MySQL, como vemos a continuación:

Imagen que contiene texto

Descripción generada automáticamente

Ilustración 6 time zone unrecognized, configuracion de MySQL.

Para ello podemos usar los siguientes comandos de SQL.

SELECT @@global.time\_zone;

SET GLOBAL time\_zone = '+1:00';

Una vez arreglada la configuración horaria de MySQL, el registro del microservicio en Eureka es exitoso.

Imagen que contiene captura de pantalla, monitor

Descripción generada automáticamente

Ilustración 7 manageplayer corriendo

Y veremos como Eureka lo muestra entre los servicios que esta registrando.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Ilustración 8 UI eureka, microservicio manageplayer.

Si ahora hacemos un Request HTTP de tipo GET a la dirección localhost:50001, la aplicación debe respondernos con los datos que corresponden al metod getPlayers del controlador.

Imagen que contiene captura de pantalla, negro, monitor

Descripción generada automáticamente

Ilustración 9

# microservice.eureka.client.managegame

Se trata del servicio que implementa la funcionalidad referida a los jugadores, vemos a continuación su estructura.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Ilustración 10 estructura microservicio managegame

Y sus recursos:

* applicaction.yml, con el contenido.

server:

port: 50003

eureka:

instance:

hostname: localhost

instance-id: ${eureka.instance.hostname}:${spring.application.name}:${spring.application.instance\_id:${random.value}}

client:

service-url:

default-zone: http://localhost:8761/

spring:

datasource:

url: jdbc:mysql://localhost:3306/games\_microservices

username: root

password: root

driver-class-name: com.mysql.cj.jdbc.Driver

En esta ocasión hemos añadido la configuración de la conexión a la base de datos en el recurso del proyecto, ya que la gestionara Spring Data.

* bootstrap.yml, con el contenido:

spring:

application:

name: client-managegames

Configurar las dependencias

<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>

  <parent>

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

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

    <version>2.1.4.RELEASE</version>

    <relativePath/> <!-- lookup parent from repository -->

  </parent>

  <groupId>dwsc</groupId>

  <artifactId>microservice.eureka.client.managegame</artifactId>

  <version>0.0.1-SNAPSHOT</version>

  <name>microservice.eureka.client.managegame</name

…

<dependencies>

<dependency>

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

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

…

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

</dependencies>

Clases

## Games.java

Esta clase representa al modelo games:

package dwsc.microservice.eureka.client.managegame.domain;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.xml.bind.annotation.XmlRootElement;

@*XmlRootElement*

@*Entity*

public class Games {

  @*Id*

  @*GeneratedValue*(strategy = GenerationType.IDENTITY)

  private *Integer* game\_id;

  private *String* name;

  private *String* description;

  private *String* cover\_url;

## GamesRepository.java

Este microservicio hace uso de Spring Data, por eso esta interfaz extiende de CrudRepository, para implementar la funcionalidad que necesitemos con esta librería.

package dwsc.microservice.eureka.client.managegame.repository;

import java.util.Optional;

import org.springframework.data.repository.CrudRepository;

import dwsc.microservice.eureka.client.managegame.domain.Games;

public interface GamesRepository extends *CrudRepository*<*Games*, *Integer*> {

*Iterable*<*Games*> findAll();

*Optional*<*Games*> findById(*Integer* *id*);

*Games* findByName(*String* *name*);

  <*S* extends *Games*> *S* save(*S* *entity*);

*void* deleteById(*Integer* *id*);

}

## GamesController.java

Esta clase es el controlador de la funcionalidad, e implementa los siguientes métodos:

@*RestController*

public class GamesController {

@*Autowired*

*GamesRepository* gameRepository;

// Mapping the path in the microservice to get all games

@*RequestMapping*(value = "/games", method = RequestMethod.GET)

public *ResponseEntity*<*Object*> findGames(){…

// Mapping the path in the microservice to get games by their id

@*RequestMapping*(value = "/games/id/{id}", method = RequestMethod.GET)

public *ResponseEntity*<*Object*> findGameById(@*PathVariable* *Integer* *id*){…

// Mapping the path in the microservice to get games by their name

@*RequestMapping*(value = "/games/name/{name}", method = RequestMethod.GET)

public *ResponseEntity*<*Object*> findGameByName(@*PathVariable* *String* *name*){…

// Mapping the path in the microservice to add a game

@*RequestMapping*(value = "/games/add", method = RequestMethod.POST)

public *ResponseEntity*<*Object*> insertGame(@*RequestParam*("name") *String* *name*,

@*RequestParam*("description") *String* *description*,

@*RequestParam*("cover\_url") *String* *cover\_url*){…

// Mapping the path in the microservice to delete a game

@*RequestMapping*(value = "/games/delete/{id}", method = RequestMethod.DELETE)

public *ResponseEntity*<*Object*> deleteGame(@*PathVariable* *Integer* *id*){…

// Mapping the path in the microservice to update a game

@*RequestMapping*(value = "/games/update", method = RequestMethod.PUT)

public *ResponseEntity*<*Object*> updateGame(@*RequestParam*("game\_id") *Integer* *game\_id*,

@*RequestParam*("name") *String* *name*,

@*RequestParam*("description") *String* *description*,

@*RequestParam*("cover\_url") *String* *cover\_url*){…

}

## Aplication.java (games)

Además, en la clase que contiene el método principal, tendremos que añadirle una anotación para que haga uso del descubridor de Eureka, como mostramos a continuación.

package dwsc.microservice.eureka.client.managegame;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@*SpringBootApplication*

@*EnableDiscoveryClient*

public class Application {

  public static *void* main(*String*[] *args*) {

    SpringApplication.run(Application.class, args);

  }

}

Levantamos el servicio, y se registrará en Eureka:

Imagen que contiene captura de pantalla, monitor

Descripción generada automáticamente

Ilustración 11 manage games corriendo.

En la ilustración 8 vemos que el servidor Eureka ya ha registrado este microservicio.

Si hacemos un Request de ejemplo como en el caso anterior, al puerto 50003, obtendremos la lista de los juegos, para diferenciarlo del caso de los jugadores hemos puesto el método findGames en la ruta /games.

Imagen que contiene captura de pantalla, negro

Descripción generada automáticamente

Ilustración 12

# microservice.eureka.client.

# Segundo punto