



UNIVERSIDAD DE ALMERÍA

PROYECTO FINAL

Desarrollo Web Basado en Servicios y Componentes

Juan Antonio Rodríguez Baeza

12 DE JUNIO DE 2019

UNIVERSIDAD DE ALMERÍA
Escuela Superior de Ingeniería



UNIVERSIDAD DE ALMERÍA

Índice

1.	microservices.eureka.server.....	3
2.	Despliegue de la base de datos	5
3.	microservice.eureka.client.manageplayer	6
3.1.	Player.java	8
3.2.	PlayerService.java	8
3.3.	PlayerServiceImpl.java	9
3.4.	PlayerController.java	10
3.5.	Application.java (players).....	11
4.	microservice.eureka.client.managegame	14
4.1.	Games.java	16
4.2.	GamesRepository.java	16
4.3.	GamesController.java	17
4.4.	Aplication.java (games)	18
5.	microservice.eureka.client.player_games.....	20
5.1.	PlayerGames.java	21
5.2.	PlayerGamesService.java	22
5.3.	PlayerGamesServiceImpl.java.....	22
5.4.	PlayerGamesController.java	22
5.5.	Application.java (player_games).....	23
6.	microservice.eureka.client.followings	24
6.1.	Followings.java	25
6.2.	FollowingsService.java	26
6.3.	FollowingsServiceImpl.java	26
6.4.	FollowingsController.java.....	27
6.5.	Application.java (followings)	27
7.	microservice.eureka.client.data_validator.....	28
7.1.	DataValidatorService.java	29
7.2.	DataValidatorServiceImpl.java.....	29
7.3.	DataValidatorController.java	29
7.4.	Application.java (data_validator).....	30
8.	microservice.webclient.manager	31
9.	Pasos de la ejecución.....	33
9.1.	MAMP	33
9.2.	Eureka server.	34
9.3.	Microservicios	34
9.4.	Postman	35
9.5.	Añadir juego	36



Ilustraciones.

Ilustración 1 estructura eureka server.....	3
Ilustración 2 eureka server corriendo.....	4
Ilustración 3 UI de eureka server.	4
Ilustración 4 diagrama de la base de datos.	5
Ilustración 5 estructura microservicio manageplayer.	6
Ilustración 6 time zone unrecognized, configuracion de MySQL.	11
Ilustración 7 manageplayer corriendo.....	12
Ilustración 8 UI eureka, microservicio manageplayer.	12
Ilustración 9 postman recibiendo los datos desde players.....	13
Ilustración 10 postman insertando un dato en player.	13
Ilustración 11 estructura microservicio managegame.....	14
Ilustración 12 manage games corriendo.....	18
Ilustración 13 postman recibiendo los datos desde games.....	19
Ilustración 14 estructura microservicio player_games.	20
Ilustración 15 player_games corriendo.	23
Ilustración 16 UI eureka, microservicio player_games.....	23
Ilustración 17 estructura microservicio followings.	24
Ilustración 18 followings corriendo.....	27
Ilustración 19 UI eureka, microservicio followings.	27
Ilustración 20 estructura microservicio data_validator.....	28
Ilustración 21 descargando data_validator desde Spring Initializr.	29
Ilustración 22 data_vaidator corriendo.....	30
Ilustración 23 UI eureka, microservicio data_validator.....	30
Ilustración 24 estructura microservicio webclient.manager.....	31
Ilustración 25 webclient.manager corriendo.....	31
Ilustración 26 formulario add game.....	32
Ilustración 27 añadiendo un juego desde el cliente web.....	32
Ilustración 28 enciendo MySQL server.....	33
Ilustración 29 puerto de la base de datos.....	33
Ilustración 30 time_zone.....	33
Ilustración 31 eureka server encendido.	34
Ilustración 32 microservicios encendidos.....	34
Ilustración 33 postman - http://localhost:50001/.....	35
Ilustración 34 postman - http://localhost:50003/games.....	35
Ilustración 35 insertando un nuevo juego.	36

1. 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:

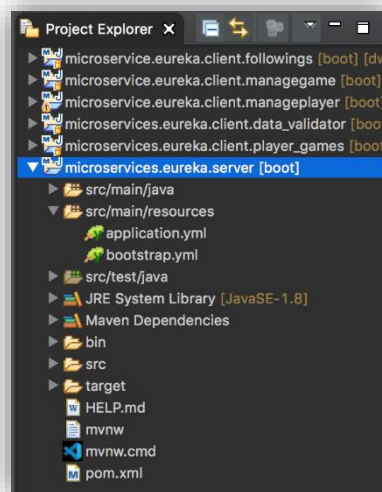


Ilustración 1 estructura eureka server.

Vemos que sus recursos son:

- application.yml, con el contenido.

```
# eureka
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.

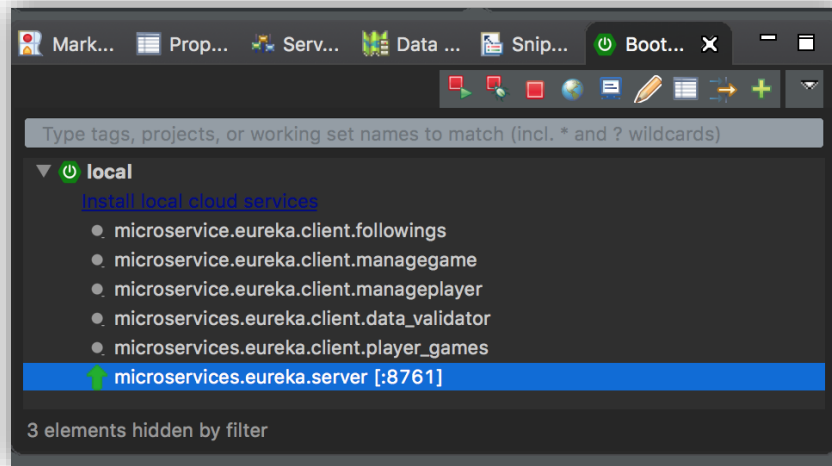


Ilustración 2 eureka server corriendo.

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

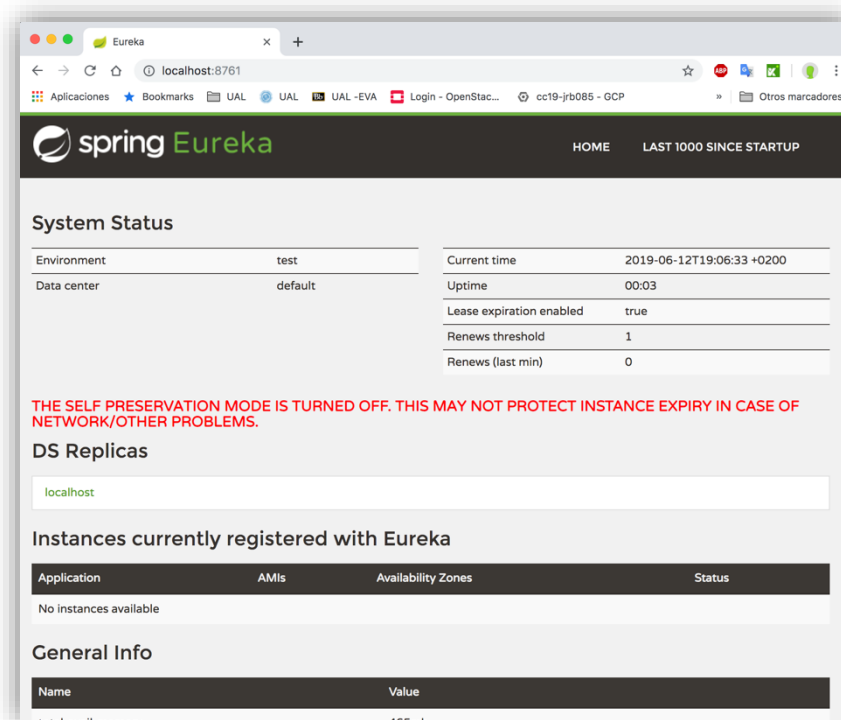


Ilustración 3 UI de eureka server.

2. 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.

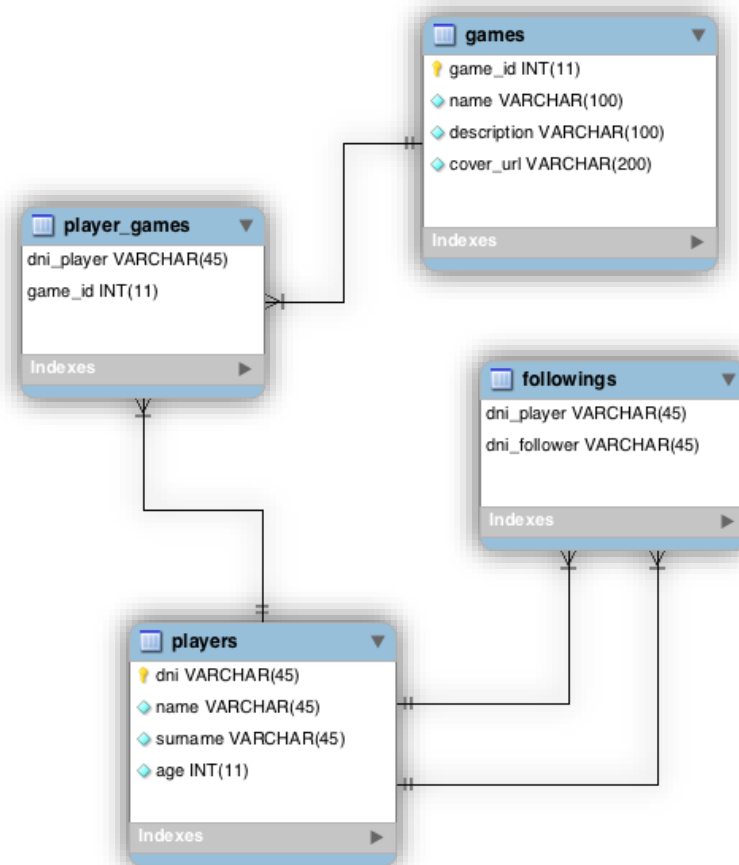


Ilustración 4 diagrama de la base de datos.

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.

3. microservice.eureka.client.manageplayer

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

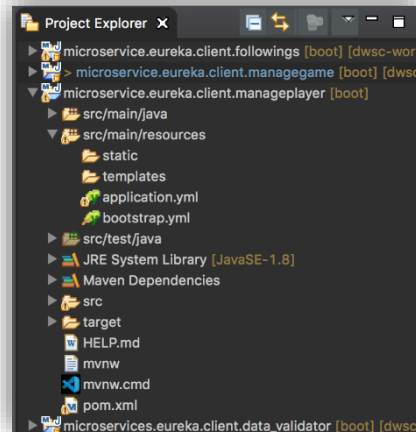


Ilustración 5 estructura microservicio manageplayer.

Y sus recursos:

- application.yml, con el contenido.

```
# manageplayer

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 tests. 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

3.1. 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;
```

3.2. PlayerService.java

Se trata de la interfaz de este servicio, declara las 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);
}
```

3.3. 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) {...

}
```

3.4. PlayerController.java

El controlador del Proyecto es el que finalmente hace uso de las funciones anteriormente declaradas 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 ResponseEntity<ArrayList<Object>> getPlayers() {
    ArrayList<Players> players = playerService.getPlayersFromDB();

    if(players.isEmpty()) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(players);
    }
    else {
        return ResponseEntity.status(HttpStatus.OK).body(players);
    }
}
```

Para hacer referencia al control de códigos, vamos a ver la implementación del método createPlayer, que establece una serie de estructuras de control para gestionarlo.

```
@RequestMapping(value = "/player/", method = RequestMethod.POST)
public ResponseEntity<Players> createPlayer(@RequestParam("dni") String dni,
    @RequestParam("name") String name,
    @RequestParam("surname") String surname,
    @RequestParam("age") int age) {
    Players player = playerService.getPlayerByDNIFromDB(dni);
    if(player == null) {
        boolean validator = dataValidatorClient.validateData(dni).getBody();

        if(validator) {
            Players inserted = playerService.createPlayerInDB(dni, name, surname, age);
            if(inserted != null){
                return ResponseEntity.status(HttpStatus.CREATED).body(inserted);
            }
            else {
                return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(inserted);
            }
        }
        else {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
        }
    }
    else {
        return ResponseEntity.status(HttpStatus.CONFLICT).body(null);
    }
}
```

También podemos ver el uso de la validación para el DNI del objeto player.

Como vemos en el extracto anterior, se realiza un control de las posibles respuestas http, en los casos en que la inserción sea fallida, o que no este disponible el recurso, que el dato ya exista o si la inserción fue correcta.

3.5. 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.

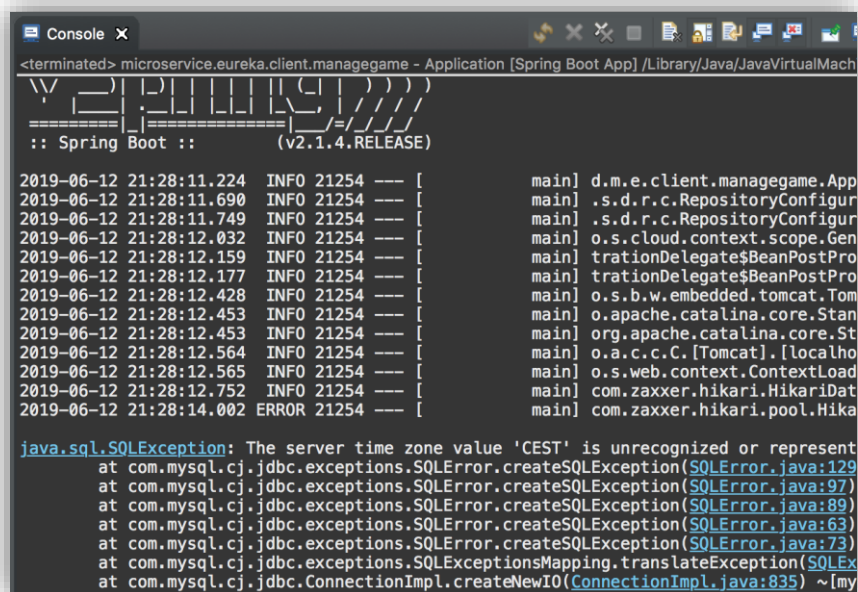
```
package dwsc.microservice.eureka.client.manageplayer;

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:



```
<terminated> microservice.eureka.client.managegame - Application [Spring Boot App] /Library/Java/JavaVirtualMach
=====
:: Spring Boot :: (v2.1.4.RELEASE)

2019-06-12 21:28:11.224 INFO 21254 --- [main] d.m.e.client.managegame.App
2019-06-12 21:28:11.690 INFO 21254 --- [main] .s.d.r.c.RepositoryConfigur
2019-06-12 21:28:11.749 INFO 21254 --- [main] .s.d.r.c.RepositoryConfigur
2019-06-12 21:28:12.032 INFO 21254 --- [main] o.s.cloud.context.scope.Gen
2019-06-12 21:28:12.159 INFO 21254 --- [main] trationDelegate$BeanPostPro
2019-06-12 21:28:12.177 INFO 21254 --- [main] trationDelegate$BeanPostPro
2019-06-12 21:28:12.428 INFO 21254 --- [main] o.s.b.w.embedded.tomcat.Tom
2019-06-12 21:28:12.453 INFO 21254 --- [main] o.apache.catalina.core.Stan
2019-06-12 21:28:12.453 INFO 21254 --- [main] org.apache.catalina.core.St
2019-06-12 21:28:12.564 INFO 21254 --- [main] o.a.c.c.C.[Tomcat].[localho
2019-06-12 21:28:12.565 INFO 21254 --- [main] o.s.web.context.ContextLoad
2019-06-12 21:28:12.752 INFO 21254 --- [main] com.zaxxer.hikari.HikariDat
2019-06-12 21:28:14.002 ERROR 21254 --- [main] com.zaxxer.hikari.pool.Hika

java.sql.SQLException: The server time zone value 'CEST' is unrecognized or represent
at com.mysql.cj.jdbc.exceptions.SQLException.createSQLException(SQLException.java:129)
at com.mysql.cj.jdbc.exceptions.SQLException.createSQLException(SQLException.java:97)
at com.mysql.cj.jdbc.exceptions.SQLException.createSQLException(SQLException.java:89)
at com.mysql.cj.jdbc.exceptions.SQLException.createSQLException(SQLException.java:63)
at com.mysql.cj.jdbc.exceptions.SQLException.createSQLException(SQLException.java:73)
at com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLEx
at com.mysql.cj.jdbc.ConnectionImpl.createNewIO(ConnectionImpl.java:835) ~[my
```

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.

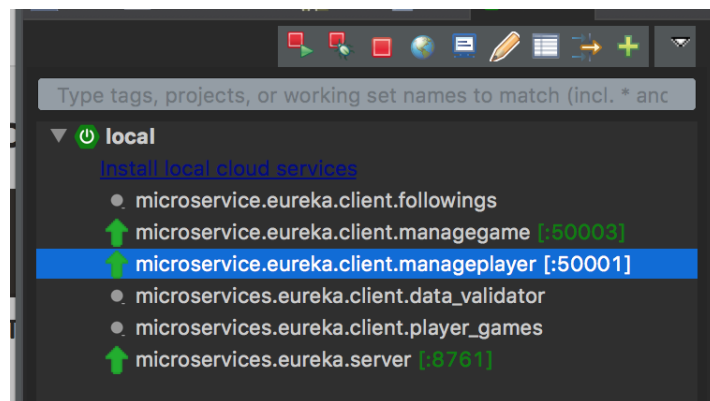


Ilustración 7 manageplayer corriendo

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

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
CLIENT-MANAGEGAMES	n/a (1)	(1)	UP (1) - localhost:client-managegames:97f036f8d221dbdc47c4f4a7387596fd
CLIENT-MANAGEPLAYER	n/a (1)	(1)	UP (1) - localhost:client-manageplayer:ff3ad8615598a34e445d2dd4cd481ac9

Ilustración 8 UI eureka, microservicio manageplayer.

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

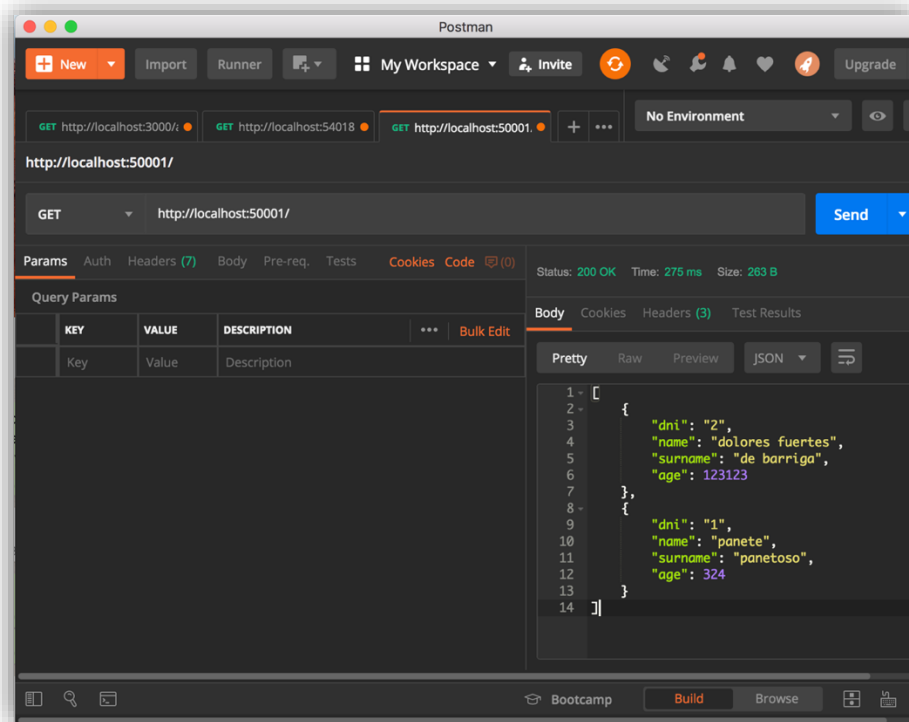


Ilustración 9 postman recibiendo los datos desde players.

```

/**
 * Los datos de la imagen anterior aun no han
 * pasado la validación del DNI.
 */

```

La siguiente captura de postman se ha hecho al finalizar todo el proceso, vemos que esta vez si se introduce un DNI valido.

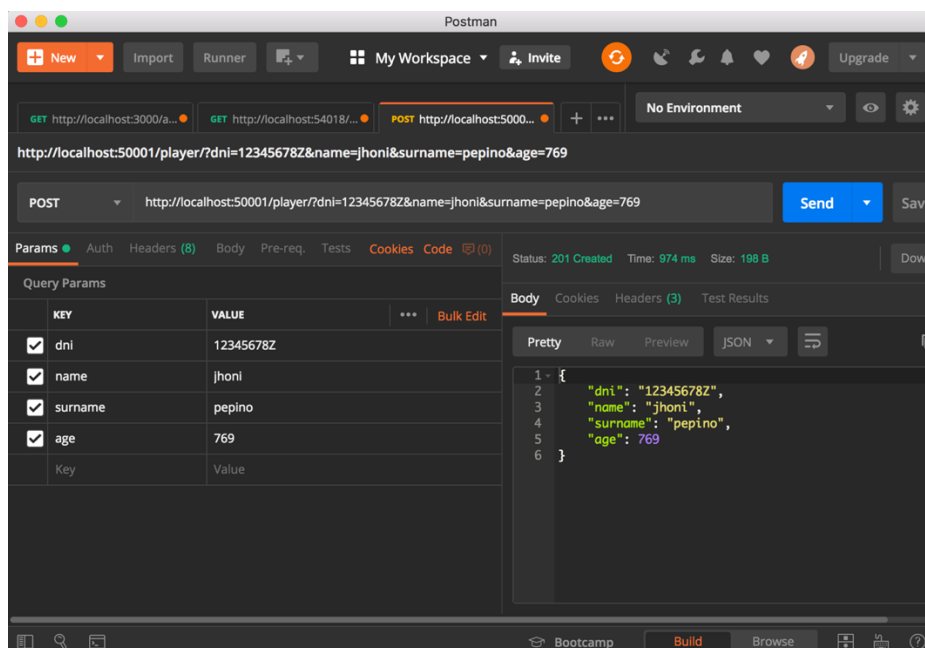


Ilustración 10 postman insertando un dato en player.

4. microservice.eureka.client.managegame

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

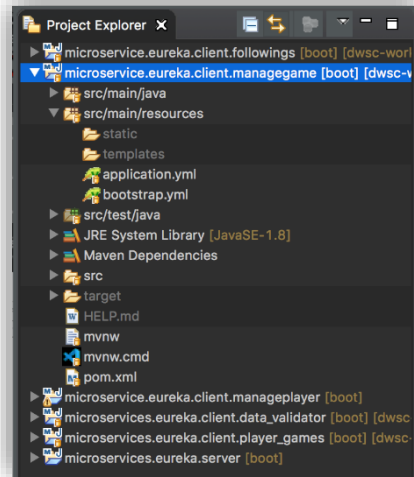


Ilustración 11 estructura microservicio managegame

Y sus recursos:

- application.yml, con el contenido.

```
# managegames
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>
```




Classes

4.1. 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;
}
```

4.2. 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);
}
```

4.3. GameController.java

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

```
@RestController
public class GameController {

    @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_un){...

    // 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_un){...
}
```

4.4. Application.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:

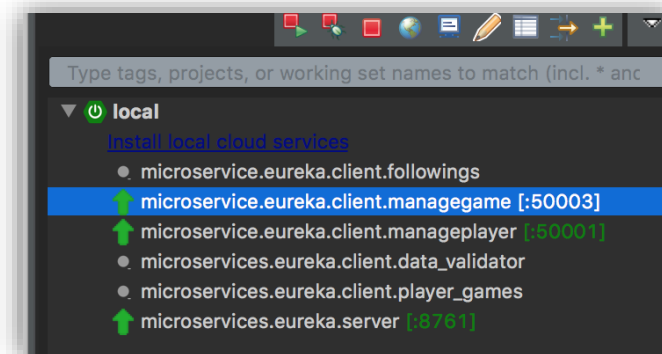


Ilustración 12 manage games corriendo.

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.

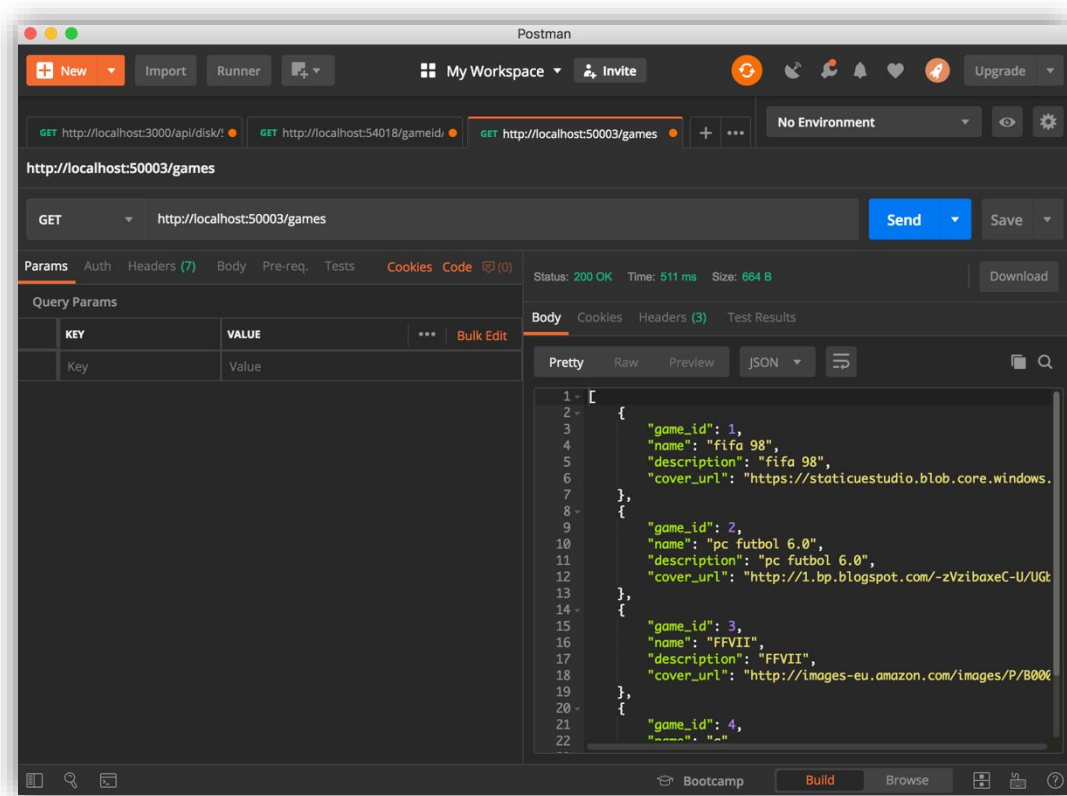


Ilustración 13 postman recibiendo los datos desde games.

5. microservice.eureka.client.player_games

Se trata del servicio que implementa la funcionalidad referida a los jugadores y la relación con sus juegos favoritos, vemos a continuación su estructura.

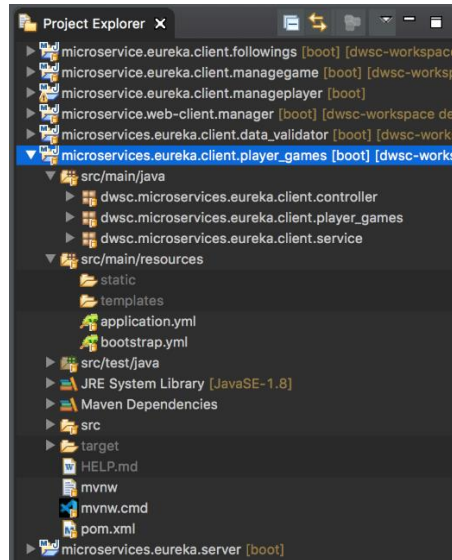


Ilustración 14 estructura microservicio player_games.

Y sus recursos:

- application.yml, con el contenido.

```
# player_games
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/
```

- Bootstrap.yml

f

```
spring:
  application:
    name: client-player_games
```

Y las dependencias

```
<?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>microservices.eureka.client.player_games</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>microservices.eureka.client.player_games</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>
  </dependency>
</dependencies>
```

Clases

5.1. PlayerGames.java

```
@XmlElement
@Entity
public class PlayerGames {
    @Id
    @JoinColumn(name="dni")
    private String dni_player;

    @Id
    @JoinColumn(name="game_id")
    private Integer game_id;
    /* getters & setters */
}
```

5.2. PlayerGamesService.java

Se trata de la interfaz de este servicio, declara las funciones propias para llevar a cabo su cometido, y además alguna función de las que depende para esos propósitos.

```
package dwsc.microservices.eureka.client.player_games.service;

import java.util.ArrayList;

import dwsc.microservices.eureka.client.player_games.domain.PlayerGames;

public interface PlayerGamesService {
    public ArrayList<PlayerGames> getPlayerGames();
    public PlayerGames getPlayerGame(String dni_player, int game_id);
    public PlayerGames addPlayerGameInDB(String dni_player, int game_id);
    public boolean deletePlayerGameFromDB(String dni_player, int game_id);
}
```

5.3. PlayerGamesServiceImpl.java

Esta clase implementa la interfaz anterior, se encarga de darle la lógica correspondiente a las distintas funciones. Como vemos también lleva su método de conexión a la base de datos.

```
@Service
public class PlayerGamesServiceImpl implements PlayerGamesService{

    private Connection connect2DB() {...

    @Override
    public ArrayList<PlayerGames> getPlayerGames(){...

    @Override
    public PlayerGames getPlayerGame(String dni_player, int game_id) {...

    @Override
    public PlayerGames addPlayerGameInDB(String dni_player, int game_id) {...

    @Override
    public boolean deletePlayerGameFromDB(String dni_player, int game_id) {...

}
```

5.4. PlayerGamesController.java

El controlador se encarga de ejecutar las funciones de la interfaz, como por ejemplo el añadir una relación entre un objeto Player y uno Games a partir de sus identificadores.

```
@Autowired
private PlayerGamesService playerGamesService;
@RequestMapping(value = "/", method = RequestMethod.GET)
public ResponseEntity<Object> getPlayerGames(){
```

5.5. Application.java (player_games)

Añadimos la anotación para que haga uso del descubridor de Eureka.

```
@SpringBootApplication
@EnableDiscoveryClient
public class Application {
```

Levantamos el servicio, y se registrará en Eureka.

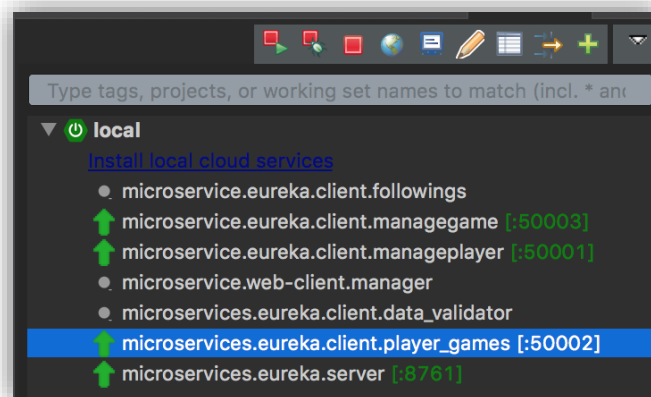


Ilustración 15 player_games corriendo.

Como ya hemos explicado más arriba en este documento, estamos poniendo los puertos directamente, y si vemos la definición de bootstrap.yml para este proyecto, habíamos establecido el puerto 50003 (debido al copiar y pegar), que ya se lo habíamos dado a otro microservicio, lógicamente no podía levantarse y registrarse con éxito. Le hemos cambiado el puerto al 50002, así se registra correctamente en Eureka.

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
CLIENT-MANAGEGAMES	n/a (1)	(1)	UP (1) - localhost:client-managegames:8189b5eafa9766a58153bfbcd89ea9a2
CLIENT-MANAGEPLAYER	n/a (1)	(1)	UP (1) - localhost:client-manageplayer:b57bcd3af3529c409ce6a4cfe4588186
CLIENT-PLAYER_GAMES	n/a (1)	(1)	UP (1) - localhost:client-player_games:e64069d3b7ca68d112d93b8dfaa2de1f

Ilustración 16 UI eureka, microservicio player_games.

6. microservice.eureka.client.followings

Se trata del servicio que implementa la funcionalidad referida a los jugadores y la relación que tiene con otros jugadores a modo de seguidores y seguidos, esta es su estructura

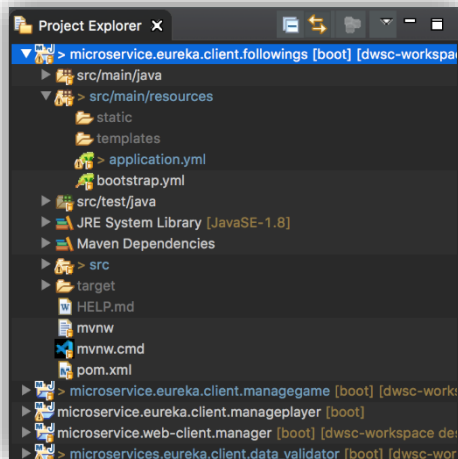


Ilustración 17 estructura microservicio followings.

Y sus recursos:

- application.yml

```
# followings
server:
  port: 50000

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

```
spring:
  application:
    name: client-managefollowers
```

sus dependencias:

```
<?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.followings</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>microservice.eureka.client.followings</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>
  </dependency>
</dependencies>
```

Clases

Siguiendo el mismo esquema de los proyectos anteriores.

6.1. Followings.java

```
@XmlElement
@Entity
public class Followings {
    @Id
    @JoinColumn(name="dni")
    private String dni_player;

    @Id
    @JoinColumn(name="dni")
    private String dni_follower;
    /* getters & setters */
}
```

6.2. FollowingsService.java

```
public interface FollowingsService {  
    public ArrayList<Followings> getFollowings();  
  
    public Followings getFollowing(String dni_player, String dni_follower);  
  
    public Followings addFollowerInDB(String dni_player, String dni_follower);  
  
    public boolean deleteFollowerFromDB(String dni_player, String dni_follower);  
}
```

6.3. FollowingsServiceImpl.java

```
@Service  
public class FollowingsServiceImpl implements FollowingsService{  
  
    private Connection connect2DB() {...  
  
    @Override  
    public ArrayList<Followings> getFollowings(){...  
  
    @Override  
    public Followings getFollowing(String dni_player, String dni_follower) {...  
  
    @Override  
    public Followings addFollowerInDB(String dni_player, String dni_follower) {...  
  
    @Override  
    public boolean deleteFollowerFromDB(String dni_player, String dni_follower) {...  
}
```

6.4. FollowingsController.java

Como en casos anteriores seguimos implementando la lógica del controlador atendiendo al control de errores, mostramos el método getFollowings como ejemplo

```
@RequestMapping(value = "/", method = RequestMethod.GET)
public ResponseEntity<Object> getFollowings(){
    ArrayList<Followings> followings = followingsService.getFollowings();
    if(followings.isEmpty()) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(followings);
    }
    else {
        return ResponseEntity.status(HttpStatus.OK).body(followings);
    }
}
```

6.5. Application.java (followings)

```
@SpringBootApplication
@EnableDiscoveryClient
public class Application {
```

Levantamos el servicio, y se registrará en Eureka.

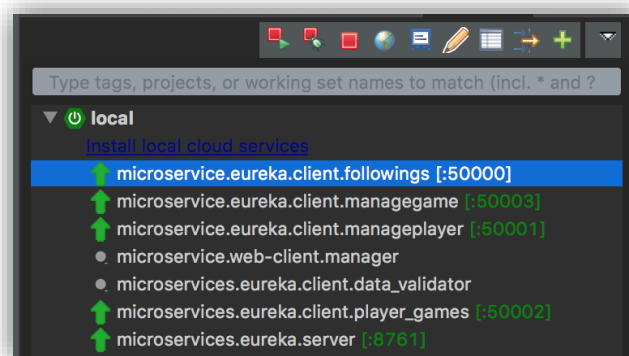


Ilustración 18 followings corriendo.

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
CLIENT-MANAGEFOLLOWERS	n/a (1)	(1)	UP (1) - localhost:client-managefollowers:10836b0bdfa86fde0ff846547796edc2
CLIENT-MANAGEGAMES	n/a (1)	(1)	UP (1) - localhost:client-managegames:8189b5eafa9766a58153bfcd89ea9a2
CLIENT-MANAGEPLAYER	n/a (1)	(1)	UP (1) - localhost:client-manageplayer:b57bcd3af3529c409ce6a4cfe4588186
CLIENT-PLAYER_GAMES	n/a (1)	(1)	UP (1) - localhost:client-player_games:e64069d3b7ca68d112d93b8d8faa2de1f

Ilustración 19 UI eureka, microservicio followings.

7. microservice.eureka.client.data_validator

Se trata del servicio que implementa la funcionalidad referida a la validación del DNI de los jugadores, esta es su estructura

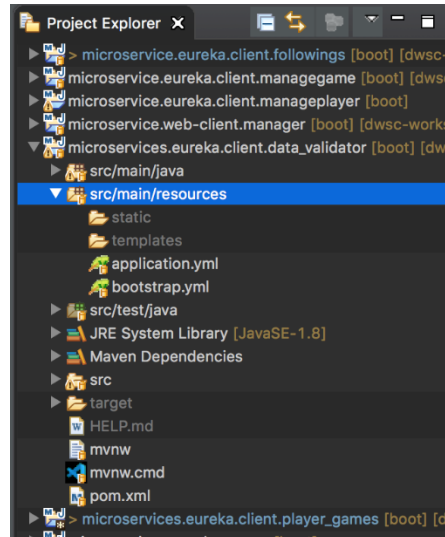


Ilustración 20 estructura microservicio data_validator

Y sus recursos:

- application.yml

```
#data_validator
server:
  port: 50004

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

```
spring:
  application:
    name: client-data-validator
```

Este proyecto no lleva ninguna dependencia más que las que trae cuando lo descargamos de [Spring Initializr](https://spring.io/).

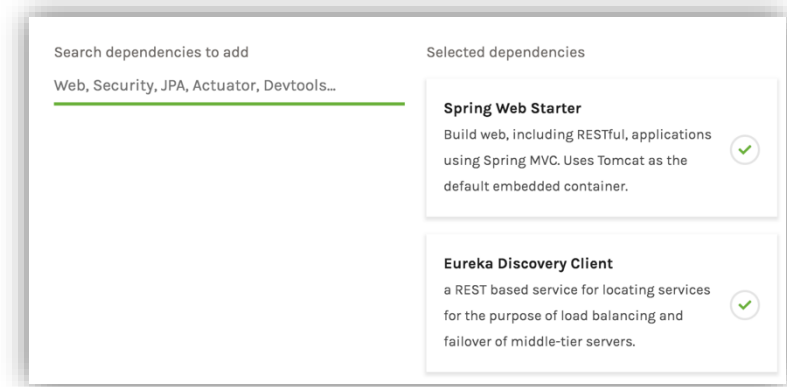


Ilustración 21 descargando data_validator desde Spring Initializr.

Clases

7.1. DataValidatorService.java

Interfaz del microservicio

```
public interface DataValidatorService {
    public boolean validateData(String data);
}
```

7.2. DataValidatorServiceImpl.java

Implementación de la lógica de la interfaz. Se implementa un método más llamado getLetterFromRemainder que se encarga de calcular cual debe ser la letra del DNI, en eso se basa la validación.

```
@Service
public class DataValidatorServiceImpl
implements DataValidatorService {

    @Override
    public boolean validateData(String data) {...

    private String getLetterFromRemainder(int remainder) {...
}
```

7.3. DataValidatorController.java

Controlador del micorservicio que conecta con la interfaz

```
@RestController
public class DataValidatorController {

    @Autowired
    private DataValidatorService dataValidatorService;

    // Mapping the path in the microservice to validate a data given
    @RequestMapping(value = "/{data}", method = RequestMethod.GET)
```

```
public ResponseEntity<Boolean> validateData(
    @PathVariable("data") String data) {...
```

7.4. Application.java (data_validator)

```
@SpringBootApplication
@EnableDiscoveryClient
public class Application {
```

Levantamos el servicio, y se registrará en Eureka.

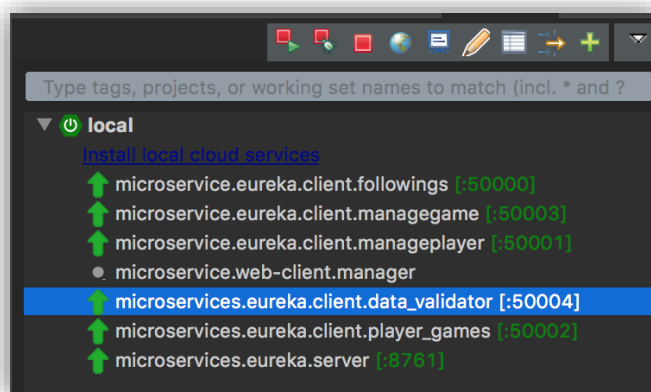


Ilustración 22 data_vaidator corriendo

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CLIENT-DATA-VALIDATOR	n/a (1)	(1)	UP (1) - localhost:client-data-validator:0efe5cdf6e1da8a0105fae9bfbba9cf6
CLIENT-MANAGEFOLLOWERS	n/a (1)	(1)	UP (1) - localhost:client-managefollowers:809073b1d97805d20400378dd1f6f3c0
CLIENT-MANAGEGAMES	n/a (1)	(1)	UP (1) - localhost:client-managegames:7e51103496a4ccc36b58397fd7adf792
CLIENT-MANAGEPLAYER	n/a (1)	(1)	UP (1) - localhost:client-manageplayer:8b7605606268a42ddc599ed6291f905d
CLIENT-PLAYER_GAMES	n/a (1)	(1)	UP (1) - localhost:client-player_games:6807d5aa10216a9da763d0d03abde66a

Ilustración 23 UI eureka, microservicio data_validator.

8. microservice.webclient.manager

Usando Polymer hemos extraído el componente webclient.manager, para que haga de interfaz de usuario del gestor, esta es su estructura.

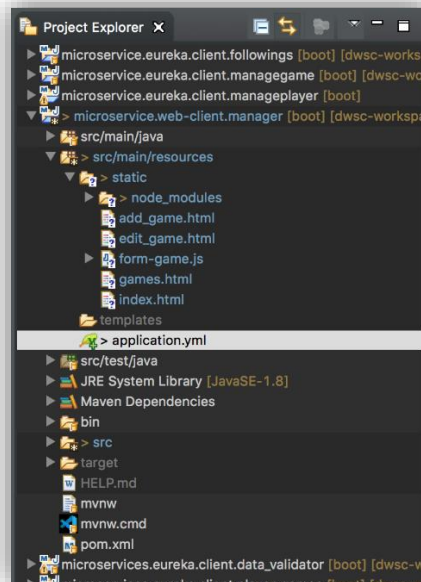


Ilustración 24 estructura microservicio webclient.manager.

A este microservicio le hemos definido el puerto 5000 por similitud.

- application.yml

```
server:
  port: 5000
```

Levantamos este microservicio, que no se va a registrar en Eureka, sino que accedemos a el directamente al puerto que le hemos indicado, a continuación, vemos en la consola de spring como queda el servicio levantado.

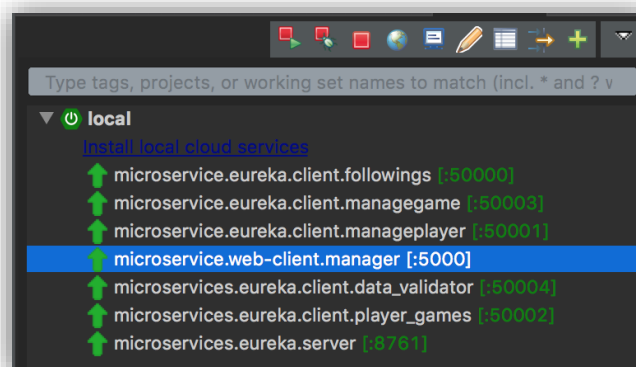


Ilustración 25 webclient.manager corriendo.

La idea de esta aplicación web es conectar con los microservicios gestores para usar sus funciones, se ha creado, por ejemplo, un formulario para el envío de datos para un juego nuevo.

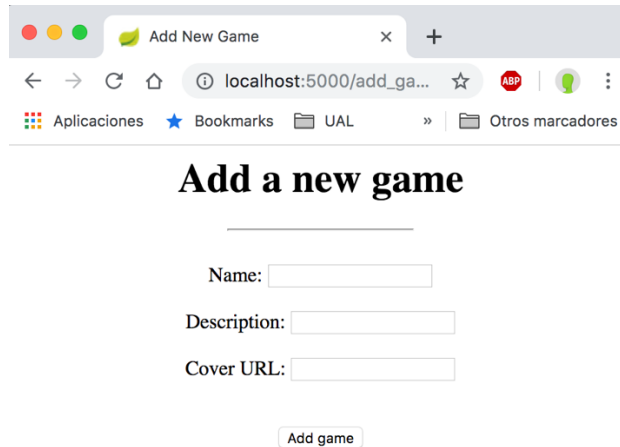


Ilustración 26 formulario add game.

En la siguiente imagen vemos una inserción en la base de datos desde el formulario.

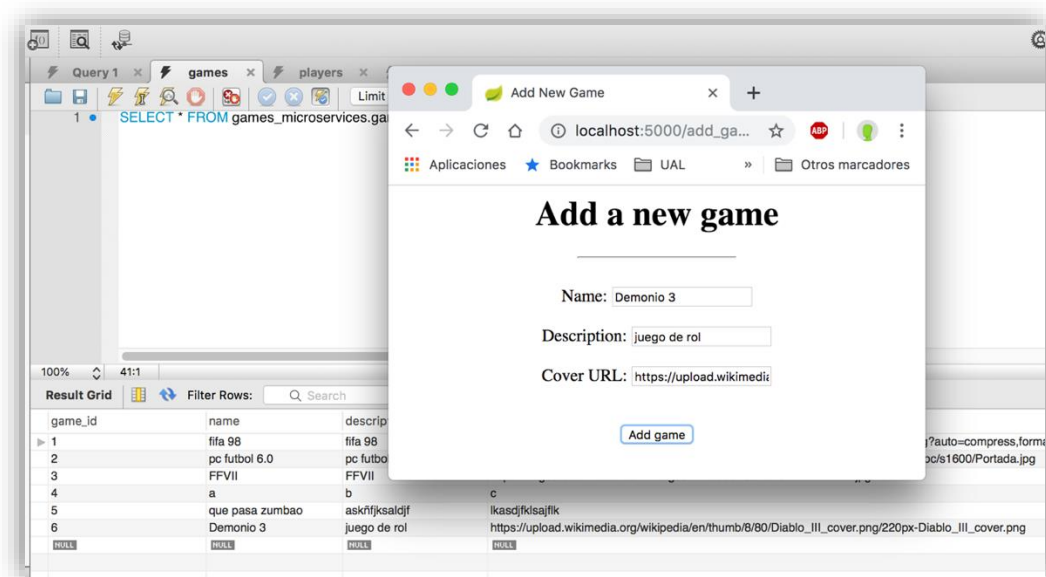


Ilustración 27 añadiendo un juego desde el cliente web.

9. Pasos de la ejecución

9.1. MAMP

Para desplegar el servicio de bases de datos hemos usado MAMP, así que lo primero que hacemos es abrir la aplicación y encender los servicios.



Ilustración 28 enciendo MySQL server

El puerto que usamos es el estándar de MySQL.

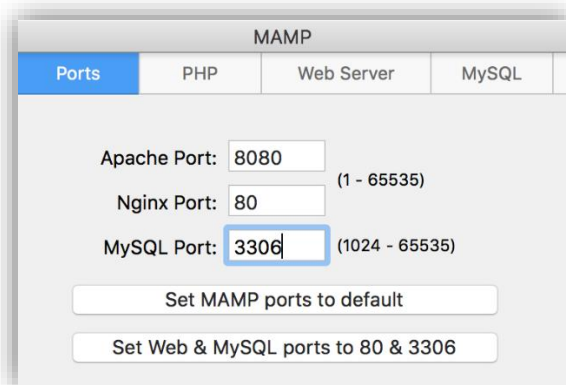


Ilustración 29 puerto de la base de datos

Una vez hecho esto, como ya explicamos habría que comprobar el valor del time_zone y arreglarlo si procede.

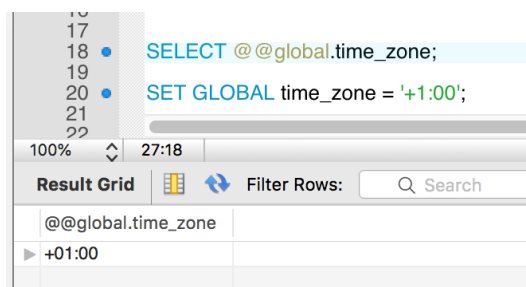


Ilustración 30 time_zone.

9.2. Eureka server.

Lo siguiente será encender Eureka server, lo hacemos así para que al encender el resto de microservicios ya encuentren el servidor de descubrimiento en pie y puedan registrarse sin problema.

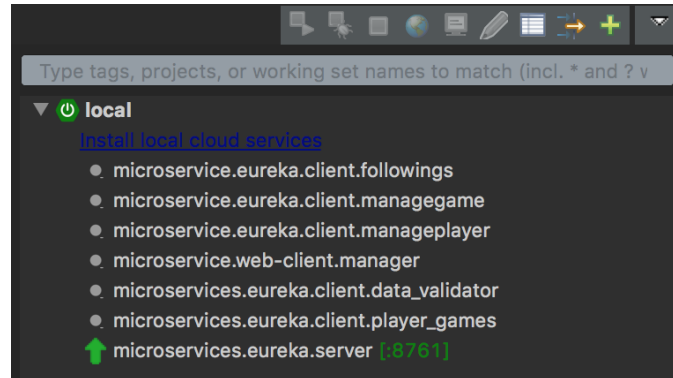


Ilustración 31 eureka server encendido.

9.3. Microservicios

Una vez este Eureka en pie encendemos el resto de microservicios.

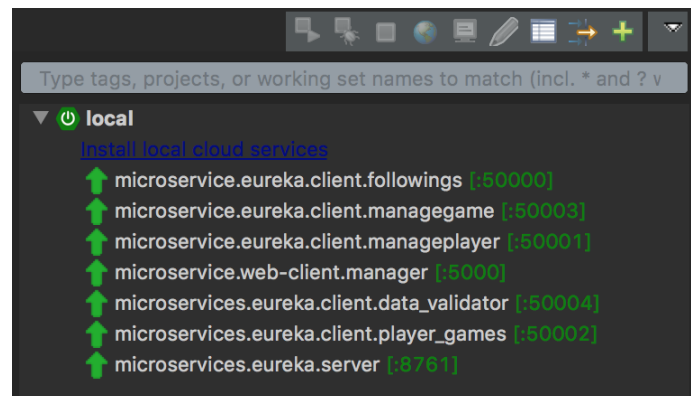


Ilustración 32 microservicios encendidos.

9.4. Postman

Como pasos siguientes podríamos usar Postman para realizar las peticiones HTTP directamente a los microservicios para comprobar que todo funciona correctamente.

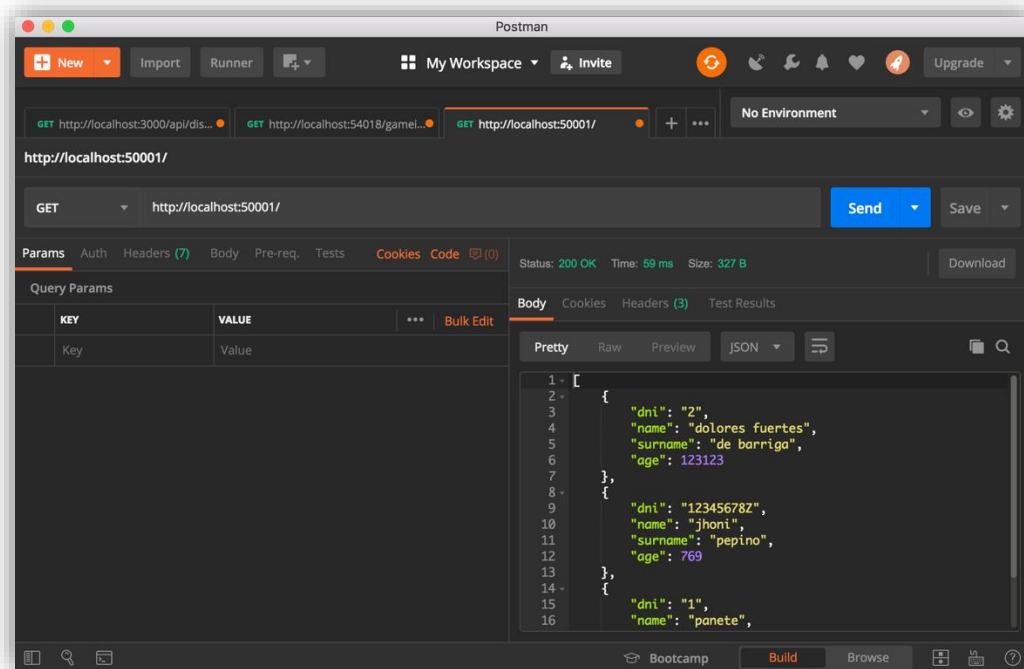


Ilustración 33 postman - `http://localhost:50001/`

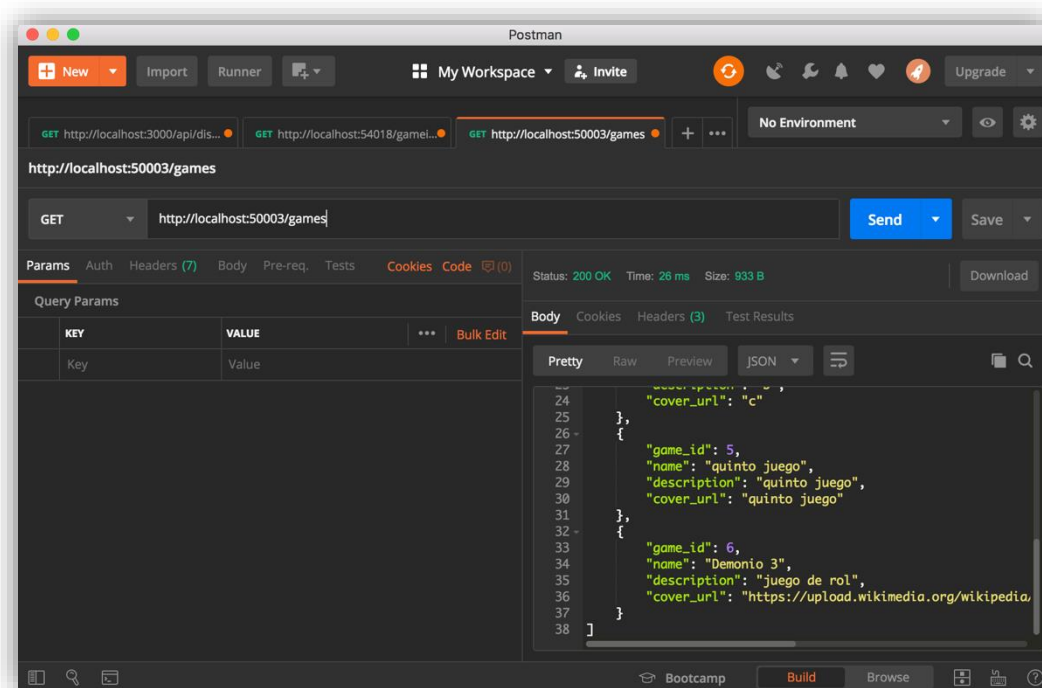


Ilustración 34 postman - `http://localhost:50003/games`

9.5. Añadir juego

Por último, vamos a insertar un nuevo juego desde el cliente web.

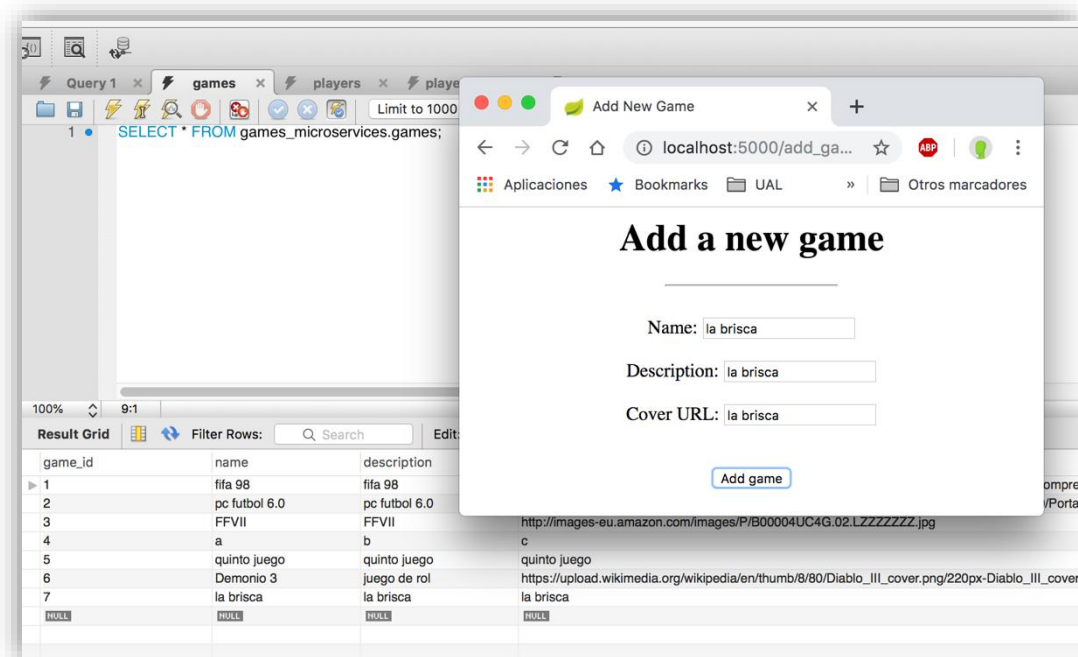


Ilustración 35 insertando un nuevo juego.