



# ESCUELA SUPERIOR DE INGENIERÍA

Programación en Internet

Grado en Ingeniería Informática

Implementación y despliegue de un servicio web REST.  
Desarrollo de un cliente jQuery para invocación de  
servicios Restful

Autores:

Antonio Ruiz Rondán y Andrés Martínez Gavira

Supervisores:

Mercedes Rodríguez García y Guadalupe Ortiz Bellot

Cádiz, 24 de diciembre de 2017

# Índice General

<b>Indice de Figuras</b>	<b>5</b>
<b>Introducción</b>	<b>7</b>
<b>1. Instalación del software necesario</b>	<b>9</b>
1.1. Instalación de WAMP	9
1.2. Configuración de MySQL	9
1.1. Creación de un servidor Tomcat en Eclipse	11
1.2. Creación de un proyecto web dinámico	15
1.2.1. Para el Servidor	16
1.2.1.1. El fichero web.xml	18
1.2.2. Para el Cliente	20
1.3. Inclusión de las librerías Jersey	21
1.4. Creación del paquete y de las clases	22
1.4.1. Para el Servidor	22
1.4.2. Para el Cliente	24
<b>2. Implementación del Servicio Web REST</b>	<b>25</b>
2.1. La clase Ponente	25
2.2. Servicios usando estructura estática (Map)	26
2.2.1. Ruta hasta nuestro servicio	26
2.2.3. Instanciamos nuestro sistema de almacenamiento	26
2.2.4. Añadimos operaciones al servicio	27
2.2.4.1. GET	27
2.2.4.2. PUT	28
2.2.4.3. DELETE	29
2.2.4.4. POST	30
2.2.5. Inicio del servicio. Resumen de rutas para pruebas	31
2.3. Servicios usando una base de datos (MySQL)	32
2.3.1. Ruta hasta nuestro servicio	32
2.3.3. Establecemos cómo conectarnos con nuestro servidor MySQL	34
2.3.4. Añadiendo operaciones al servicio	35
2.3.4.1. GET	35

2.3.4.2. PUT	37
2.3.4.3. DELETE	38
2.3.4.4. POST	39
2.3.5. Inicio del servicio. Resumen de rutas para pruebas	41
<b>3. Implementación del Cliente para invocar servicios Restful</b>	<b>43</b>
3.1. Comprendiendo las URLs de invocación de servicios	45
3.2. Creamos las funciones jQuery	45
3.2.1. Función todosPonentes()	46
3.2.2. Función obtenerPonente()	47
3.2.3. Función modificarPonente()	48
3.2.4. Función eliminarPonente()	49
3.2.5. Función nuevoPonente()	49
3.2.6. Función nuevoPonenteForm()	50
3.3. Probamos las funciones en nuestro proyecto	51
<b>Bibliografía</b>	<b>57</b>
<b>Anexo A: Clase Ponente</b>	<b>59</b>
<b>Anexo B: Clase Servicio (con Map)</b>	<b>61</b>
<b>Anexo C: Clase Servicio2 (con MySQL)</b>	<b>66</b>
<b>Anexo D: Funciones jQuery de invocación de servicios</b>	<b>74</b>
<b>Anexo E: DDL y DML de Ponentes (pnet.sql)</b>	<b>79</b>

## Indice de Figuras

Figura 1. Fichero my.ini	11
Figura 2. Creación de un servidor Tomcat en Eclipse. Paso 1	12
Figura 3. Creación de un servidor Tomcat en Eclipse. Paso 2	13
Figura 4. Creación de un servidor Tomcat en Eclipse. Paso 3	14
Figura 5. Creación de un servidor Tomcat en Eclipse. Paso 4	15
Figura 6. Creación de un proyecto web dinámico. Servidor. Paso 1	16
Figura 7. Creación de un proyecto web dinámico. Servidor. Paso 2	17
Código 1.2.1.1. Fichero web.xml	18
Figura 8. Creación de un proyecto web dinámico. Servidor. Paso 3	19
Figura 9. Creación de un proyecto web dinámico. Servidor. Paso 4	19
Figura 10. Creación de un proyecto web dinámico. Cliente. Paso 1	20
Figura 11. Inclusión de las librerías Jersey	21
Figura 12. Creación del paquete y de las clases. El Servidor. Paso 1	22
Figura 13. Creación del paquete y de las clases. El Servidor. Paso 2	23
Código 2.1. Uso de la etiqueta @XmlRootElement	25
Código 2.2.1. Estableciendo la ruta al servicio	26
Código 2.2.3. Sistema de almacenamiento	26
Código 2.2.4.1. GET. Desarrollo operaciones 1 y 2 (Obtener)	27
Código 2.2.4.2. PUT. Operación 3 (Actualizar)	28
Código 2.2.4.3. DELETE. Operación 4 (Eliminar)	29
Código 2.2.4.4. POST. Operación 5 y Operación opcional 1 (Añadir)	30
Código 2.3.1. Estableciendo la ruta al servicio	32
Código 2.3.3. Conexión con MySQL (y desconexión)	34
Código 2.3.4.1. GET. Desarrollo operaciones 1 y 2 (Obtener)	35

Código 2.3.4.2. PUT. Operación 3 (Actualizar)	37
Código 2.3.4.3. DELETE. Operación 4 (Eliminar)	38
Código 2.3.4.4. POST. Operación 5 y Operación opcional 1 (Añadir)	39
Código 3. Ejemplo html para probar servicios.	43
Código 3.2. Variables globales para construir la url	45
Código 3.2.1. Función todosPonentes()	46
Código 3.2.2. Función obtenerPonente()	47
Código 3.2.3. Función modificarPonente()	48
Código 3.2.4. Función eliminarPonente()	49
Código 3.2.5. Función nuevoPonente()	49
Código 3.2.6. Función nuevoPonenteForm()	50
Figura 14. index.html	51
Figura 15. Resultado de listar todos los Ponentes en versión con Map. rest.html	52
Figura 16. Resultado de listar todos los Ponentes en versión con Map. rest2.html	52
Figura 17. Servicios documentados con Swagger integrado en el proyecto	53
Figura 18. Detalle de documentación de una de las operaciones	53
Figura 19. Formulario	54
Figura 20. Operación obtenerPonente(dni), en versión con MySQL	54
Figura 21. Desde el navegador web de un movil.	55

## **Introducción**

En este tutorial vamos a describir cómo implementar y desplegar un servicio web REST, vamos a reutilizar también nuestro sitio web sobre “I Congreso de oftalmología de Cádiz”, para modificarlo y hacer desde ahí las invocaciones a los servicios. Para ello habilitaremos una nueva opción en el menú llamada REST, el cual a su vez es un menú desplegable con otras 3 opciones, la primera de ellas para realizar invocaciones al servicio que se desarrollará con una estructura estática, la segunda igual, pero utilizando MySQL y la tercera contendrá la documentación del servicio con swagger, pero integrado con los estilos y estructura de nuestro sitio.





# 1. Instalación del software necesario

Para la elaboración de nuestro proyecto web necesitaremos, en primer lugar, instalar las correspondientes herramientas en forma de software, como son Eclipse Neon para Java Enterprise Edition y un servidor de bases de datos MySQL como el que viene ya incluido en WAMP, para posteriormente, poder ejecutar el proyecto a través de un navegador web.

Para instalar Eclipse debemos dirigirnos al tutorial proporcionado en el campus virtual: “Instalación Software Tema 3”, donde encontraremos todos los pasos necesarios para desempeñar la tarea a realizar y bajo entorno Windows.

En cuanto a WAMP, al tratarse de una funcionalidad opcional del proyecto, puede omitirse en dicho tutorial. Tan sólo se deberá instalar WAMP si el desarrollador deseara incluir dicha funcionalidad en su proyecto. *Bastaría con instalar y configurar correctamente MySQL, sin necesidad de instalar el paquete WAMP completo, ya que éste incluye, además de MySQL, el intérprete de PHP y el servidor HTTP Apache, los cuales no necesitamos.*

## 1.1. Instalación de WAMP

Es fundamental saber que deberás instalar WAMP solo si utilizas un entorno de trabajo Windows al tiempo que desees implementar la funcionalidad en forma de base de datos como mejora.

Para bajarlo tan sólo deberás dirigirte a la página oficial de WAMP: <http://www.wampserver.com/en/> y descargarte la versión, de acuerdo a la arquitectura de tu ordenador, de 32 o 64 bits.

Posteriormente, la instalación es sencilla, hacer clic en siguiente y dejar las opciones por defecto. De igual modo al finalizar la instalación hacer clic en permitir cuando se nos notifiquen las alertas del firewall de Windows.

## 1.2. Configuración de MySQL

Una vez descargado e instalado WAMP, procederemos a la configuración de MySQL, una de las utilidades de WAMP, fundamental para el desempeño de la tarea opcional de nuestro proyecto.

Para empezar debemos cerciorarnos de que todo está correcto, es decir, necesitaremos, en primer lugar arrancar WAMP como rol administrador y ver que el icono de notificaciones se presenta en color verde.

Entramos en el terminal de MySQL como usuario root y la contraseña que le hayamos otorgado durante la instalación de WAMP, o ninguna en su defecto:

Creamos la base de datos “pnet”.

```
mysql> create database pnet;
```

Creamos el usuario “pnet” y le asignamos “pnet” como contraseña.

```
mysql> create user 'pnet'@'127.0.0.1' identified by 'pnet';
```

Después le otorgamos derechos de administrador a dicho usuario y sobre la base de datos “pnet”.

```
mysql> grant all on pnet.* to 'pnet'@'127.0.0.1';
```

Salimos de MySQL e importamos nuestro fichero **pnet.sql** (en Anexo E) sobre la base de datos recién creada “pnet”.

```
mysql> quit;
```

```
C:\wamp64\bin\mysql\mysql5.7.19\bin> mysql -h 127.0.0.1 -u pnet -p pnet < pnet.sql
```

```
Enter Password: ****
```

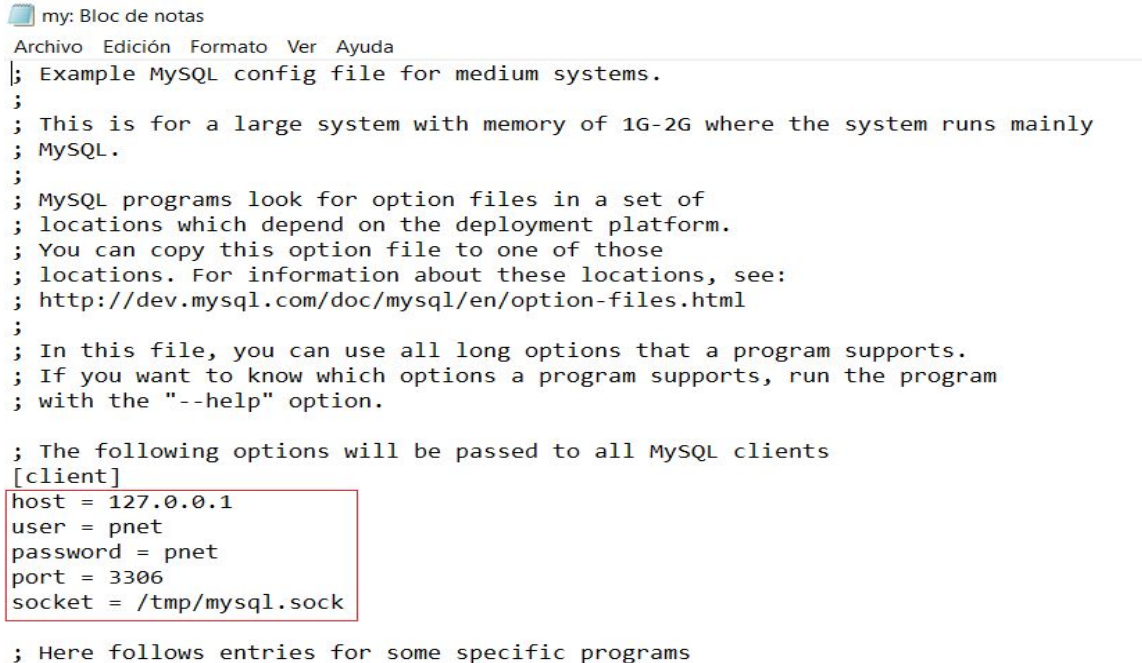
**¡Ojo!** Sólo funcionará si pnet.sql está en el directorio actual. Si no es así deberemos indicarle la ruta desde donde éste se encuentre.

Y listo, ahora debemos dirigirnos al icono de la barra de tareas de notificaciones de WAMP:



-> MySQL -> my.ini

Y se nos abre dicho fichero en modo editable. En él debemos añadir/modificar dichas líneas, las marcadas en rojo:



```
my: Bloc de notas
Archivo Edición Formato Ver Ayuda
; Example MySQL config file for medium systems.
;
; This is for a large system with memory of 1G-2G where the system runs mainly
; MySQL.
;
; MySQL programs look for option files in a set of
; locations which depend on the deployment platform.
; You can copy this option file to one of those
; locations. For information about these locations, see:
; http://dev.mysql.com/doc/mysql/en/option-files.html
;
; In this file, you can use all long options that a program supports.
; If you want to know which options a program supports, run the program
; with the "--help" option.

; The following options will be passed to all MySQL clients
[client]
host = 127.0.0.1
user = pnet
password = pnet
port = 3306
socket = /tmp/mysql.sock

; Here follows entries for some specific programs
```

Figura 1. Fichero my.ini

Y con esto tenemos preparado la parte opcional correspondiente a bases de datos del proyecto, desde la parte correspondiente a WAMP.

**Nota:** WAMP, es sólo un ejemplo también pueden usarse otros servidores de aplicaciones. Nuestro ejemplo viene enfocado sobre WAMP.

## 1.1. Creación de un servidor Tomcat en Eclipse

Ahora que hemos instalado nuestro software podemos empezar a construir nuestro proyecto y al mismo tiempo crear el servidor de aplicaciones Tomcat en Eclipse.

El orden es indiferente, pues son independientes a los proyectos. Para empezar mostraremos capturas de pantalla de cómo realizar el paso a paso de la creación de dicho servidor con el objetivo de que nos resulte todo mucho más fácil.

Hacemos clic en File -> New -> Other -> Server -> Server

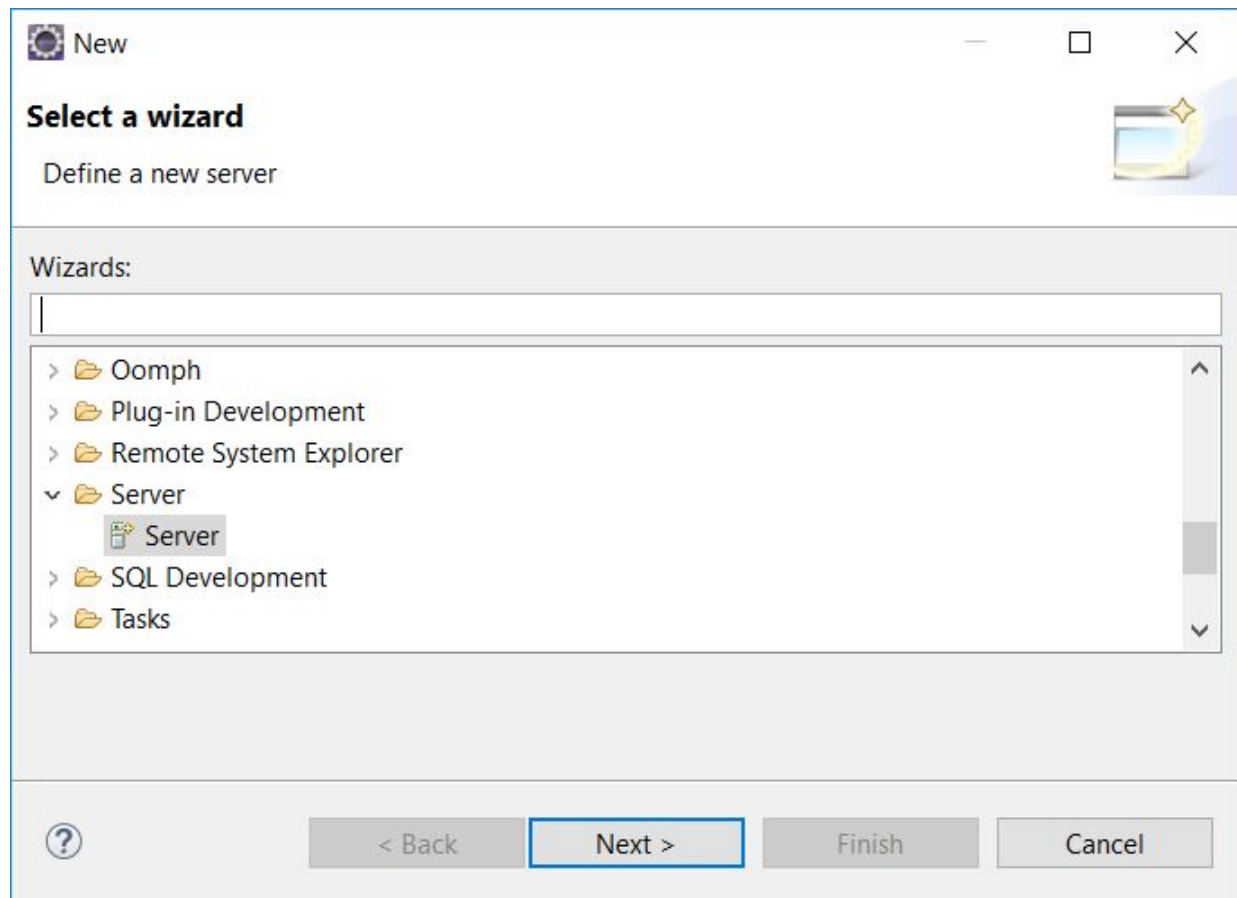


Figura 2. Creación de un servidor Tomcat en Eclipse. Paso 1

Clic en Next y dejar las opciones que vienen marcadas en la imagen, acorde al Tomcat que tenemos descomprimido en C:\Development\apache-tomcat-8.0.47

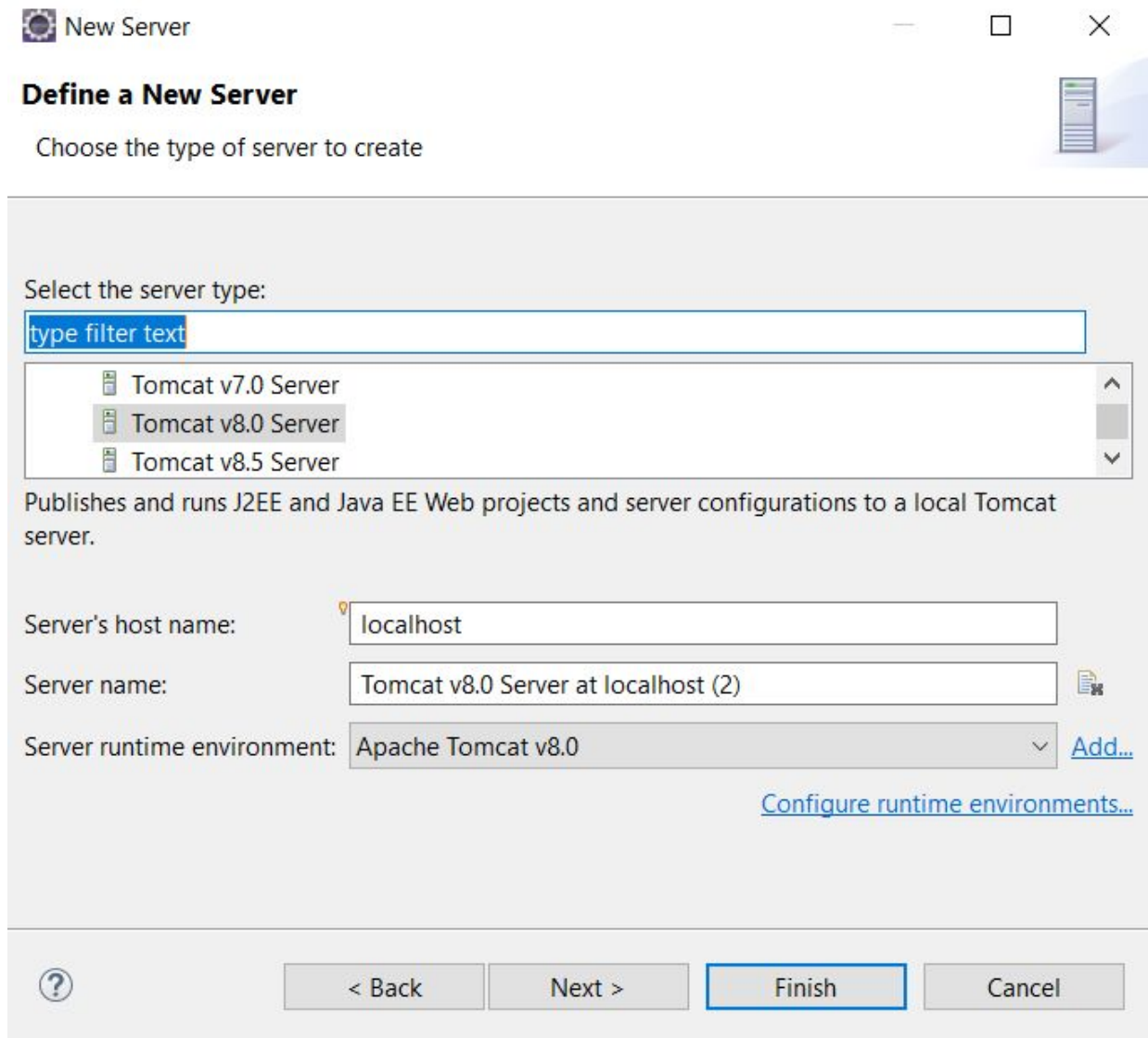


Figura 3. Creación de un servidor Tomcat en Eclipse. Paso 2

Le indicamos la ruta de instalación del Tomcat y le indicamos la ruta de nuestro JDK, no JRE, y clic en finalizar.

Se supone que debemos tener instalado el JDK, Java Development Kit en nuestro equipo, por defecto viene instalado dentro de la carpeta C:\Program Files\Java\jdk1.8.0\_152\bin, y también debemos tener configurada la variable de entorno de sistema Path con la ruta del JDK incluyendo el directorio bin y también la variable de sistema JAVA\_HOME con la ruta del JDK pero sin el directorio bin.

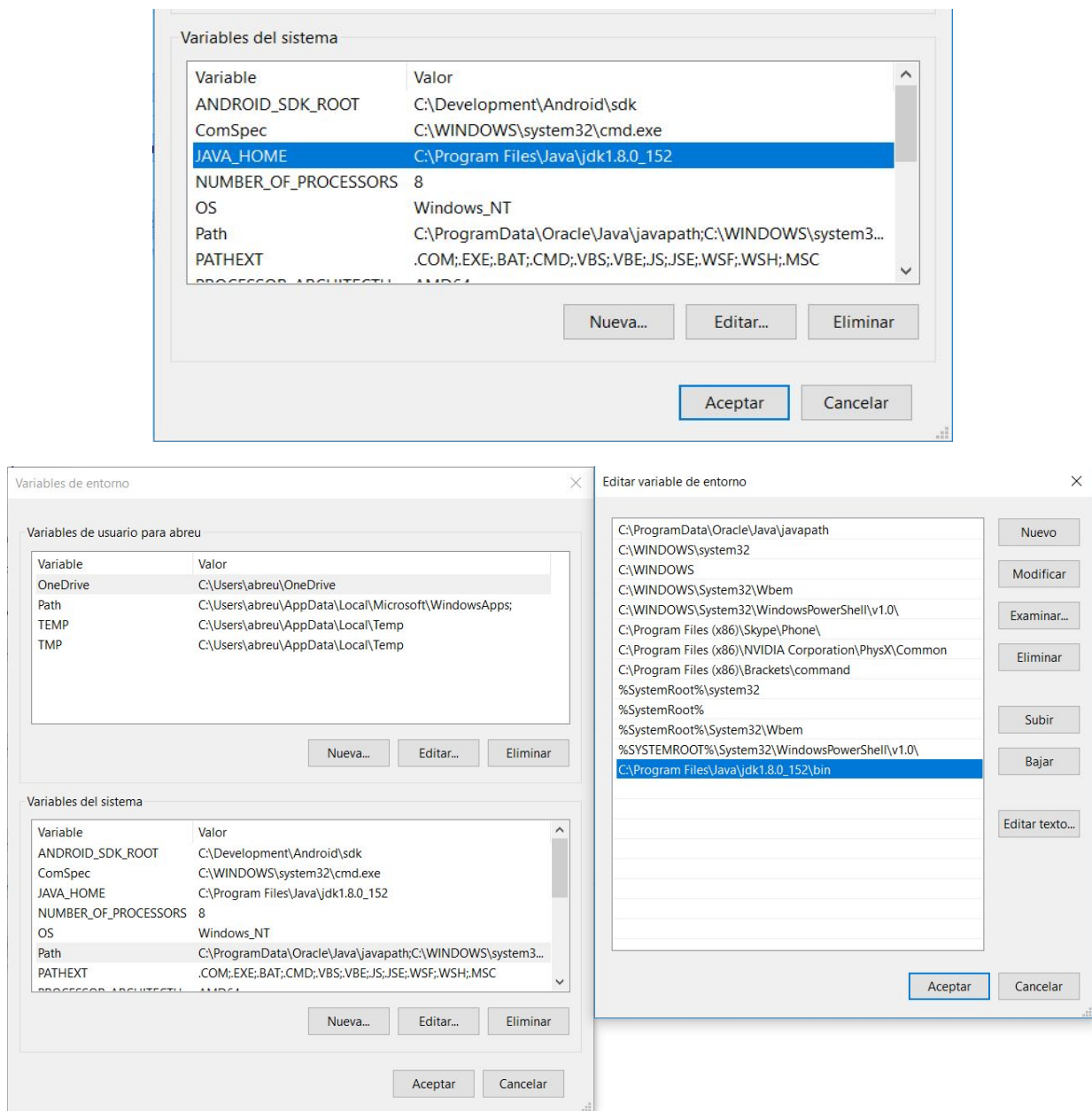


Figura 4. Creación de un servidor Tomcat en Eclipse. Paso 3

Y arrancamos el servidor Tomcat y vemos que funciona perfectamente.

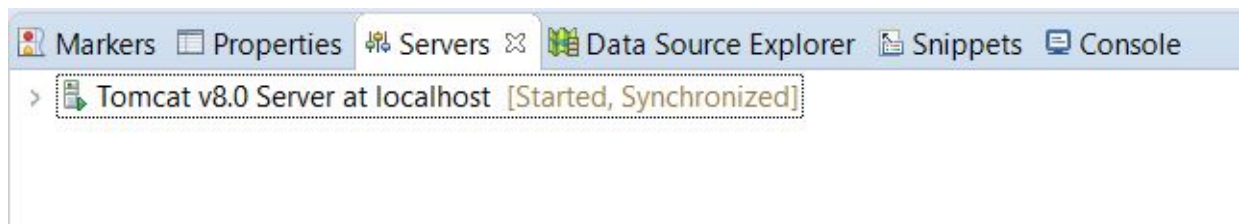


Figura 5. Creación de un servidor Tomcat en Eclipse. Paso 4

## **1.2. Creación de un proyecto web dinámico**

Ahora sí, crearemos nuestro proyecto web dinámico para luego poder desplegarlo sobre el servidor de aplicaciones Tomcat.

En primer lugar y al igual que el apartado anterior explicaremos con breves comentarios y mostraremos los detalles a través de capturas de pantalla.

Finalmente haremos distinción entre servidor y cliente.

### 1.2.1. Para el Servidor

Clic en File -> New -> Dynamic Web Project.

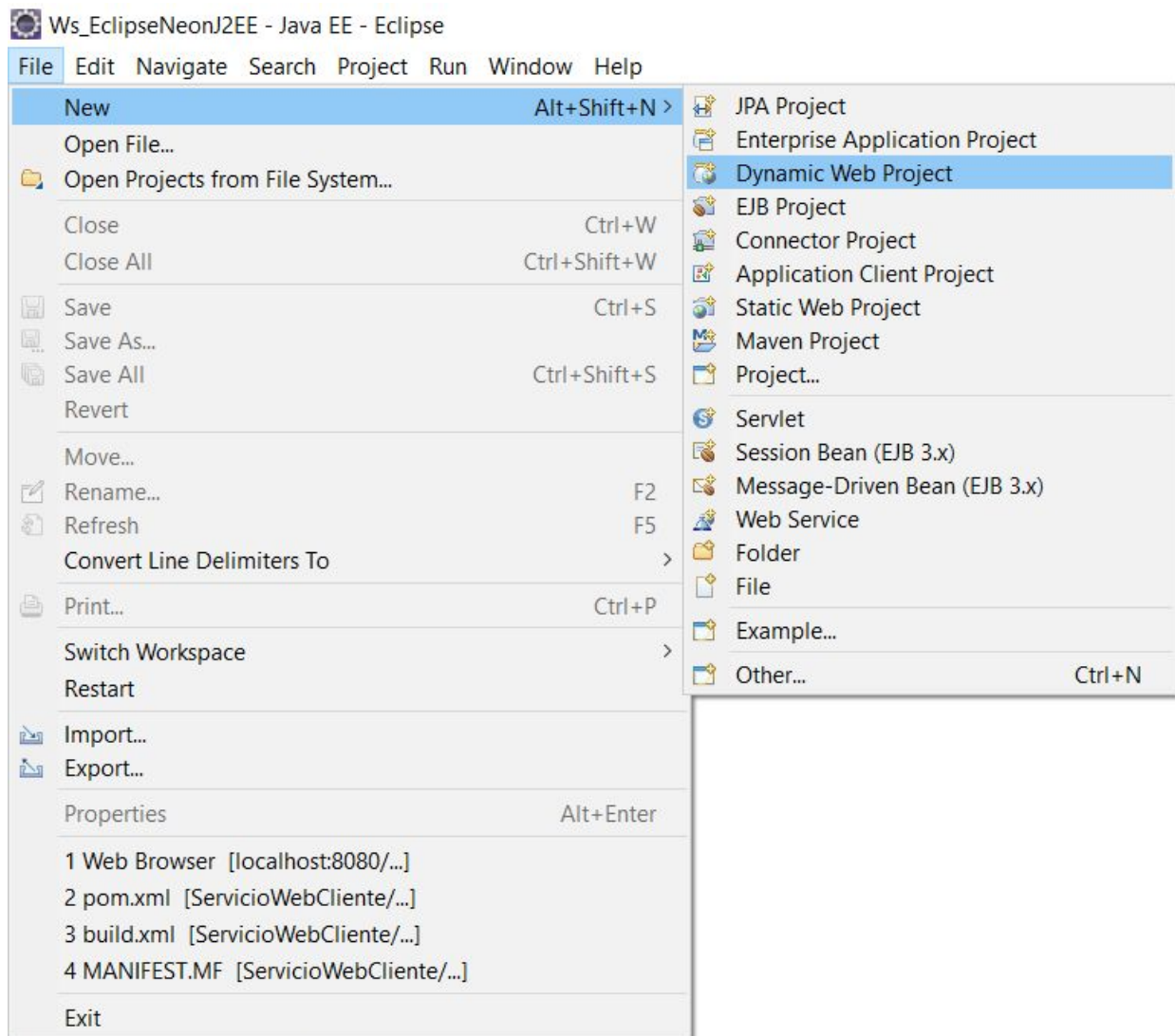
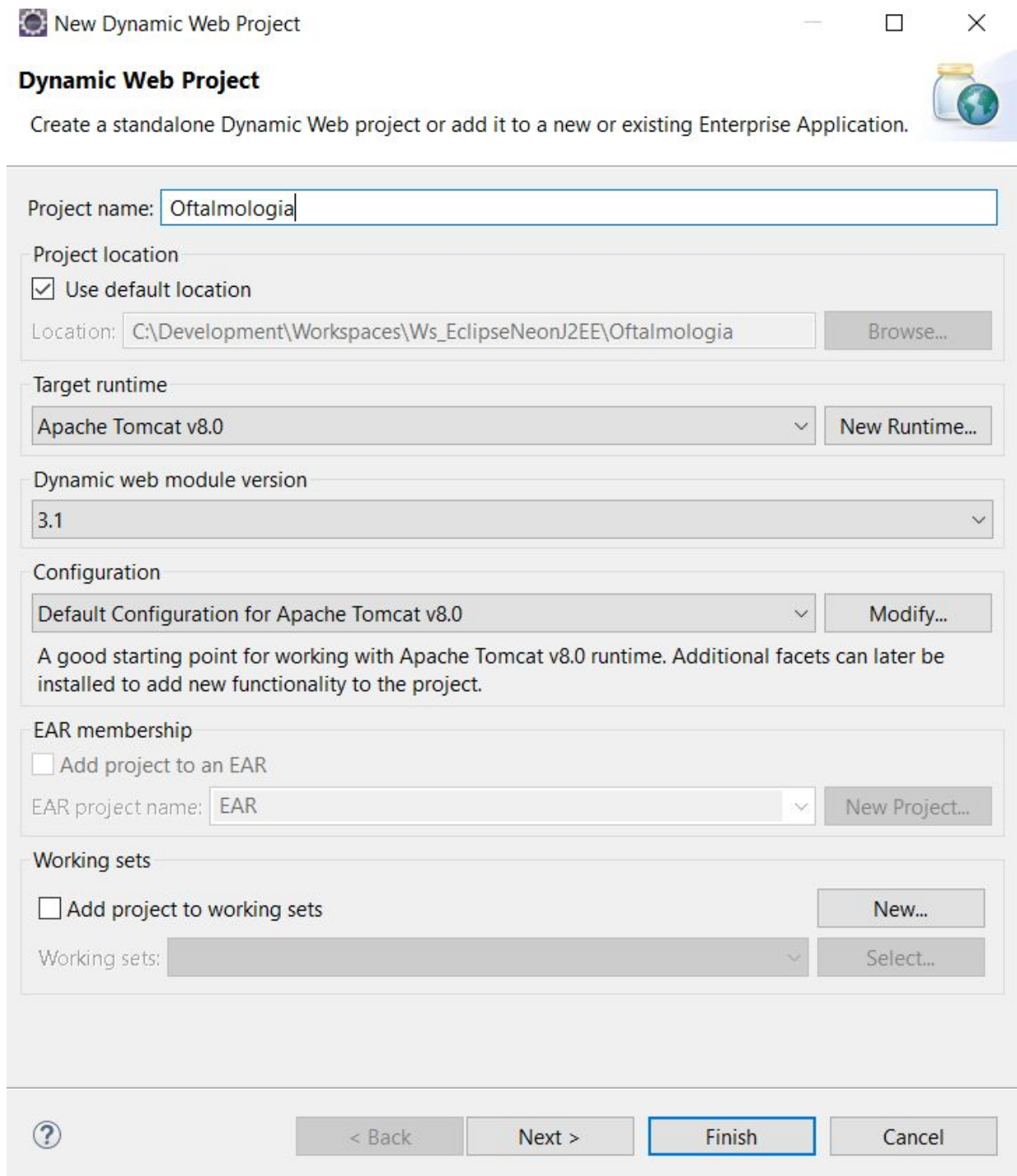


Figura 6. Creación de un proyecto web dinámico. Servidor. Paso 1





**New Dynamic Web Project**

**Dynamic Web Project**

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name:

Project location

☒ Use default location

Location:

Target runtime

Dynamic web module version

Configuration

A good starting point for working with Apache Tomcat v8.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

☐ Add project to an EAR

EAR project name:

Working sets

☐ Add project to working sets

Working sets:

Figura 7. Creación de un proyecto web dinámico. Servidor. Paso 2

Es importante respetar las convenciones de Java, el uso de las mayúsculas y minúsculas cuando sea oportuno dependiendo del objeto a tratar, y clic en Finish.

#### 1.2.1.1. El fichero web.xml

Este fichero es muy importante pues es el encargado de comunicar el servidor de aplicaciones Tomcat con nuestra aplicación. Es por tanto el encargado de realizar el despliegue de nuestra aplicación. Para crearlo debemos hacer lo siguiente:

Primero clic derecho en la carpeta WEB-INF dentro de WebContent de nuestro proyecto en File -> New -> File y escribimos web.xml y pegamos el código siguiente dentro del fichero:

##### Código 1.2.1.1. Fichero web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee"
4. xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
5. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6. http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
7.
8.   <servlet>
9.     <servlet-name>Servicio Web Rest Oftalmologia</servlet-name>
10.    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
11.    <init-param>
12.      <param-name>jersey.config.server.provider.packages</param-name>
13.      <param-value>pnetSw,com.fasterxml.jackson.jaxrs.json</param-value>
14.    </init-param>
15.  </servlet>
16.
17.  <servlet-mapping>
18.    <servlet-name>Servicio Web Rest Oftalmologia</servlet-name>
19.    <url-pattern>/*</url-pattern>
20.  </servlet-mapping>
21.
22. </web-app>
```

Ahora podemos desplegar el servidor de nuestro proyecto sobre el servidor de aplicaciones Tomcat así:

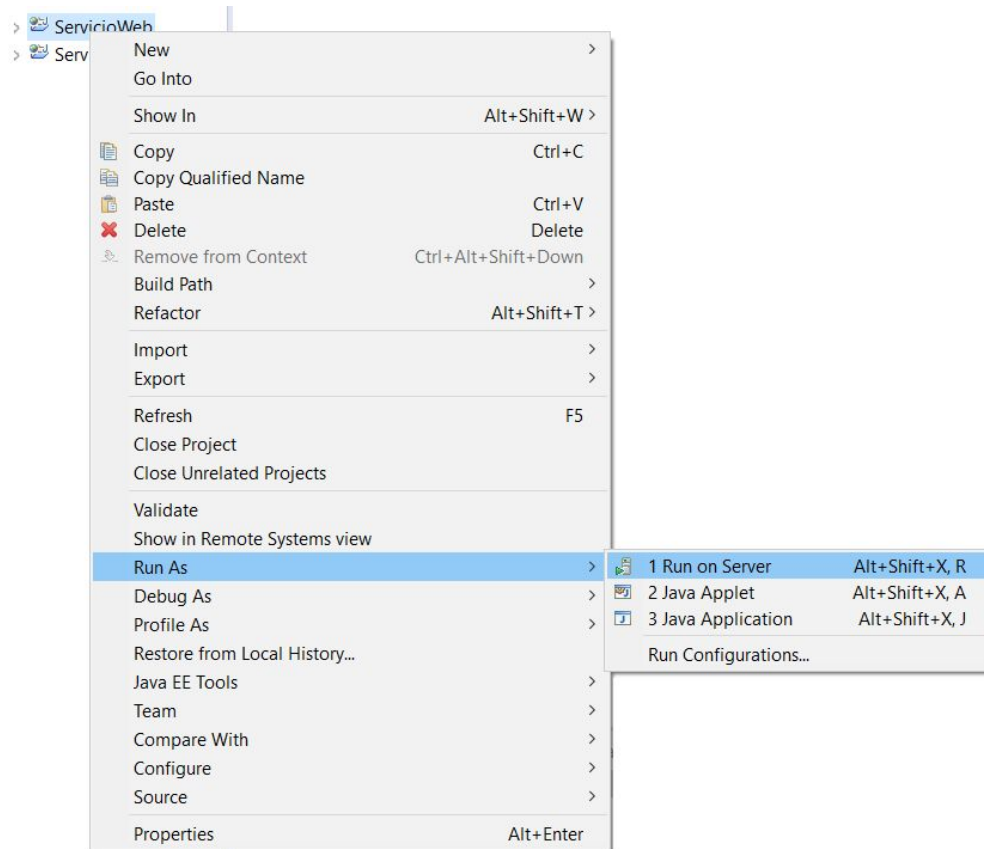


Figura 8. Creación de un proyecto web dinámico. Servidor. Paso 3

Y vemos que, efectivamente, se encuentra desplegado el proyecto sobre el servidor Tomcat.

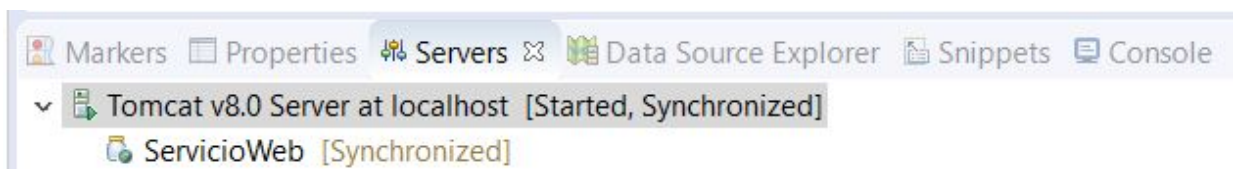


Figura 9. Creación de un proyecto web dinámico. Servidor. Paso 4

### 1.2.2. Para el Cliente

Para el cliente debemos seguir los mismos pasos que el servidor, con las excepciones de que ahora no necesitamos ningún fichero web.xml y además debemos agregar las vistas o páginas de nuestro proyecto web.

Por lo que nos quedaría algo tal que así:

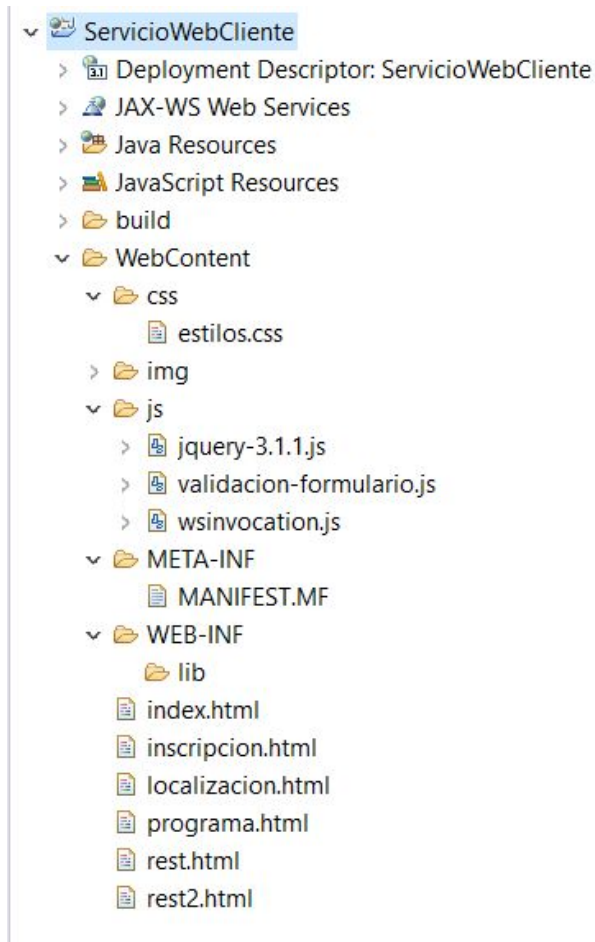


Figura 10. Creación de un proyecto web dinámico. Cliente. Paso 1

Y obsérvese como ahora el proyecto lo llamamos igual pero terminado con el sufijo Cliente.

### 1.3. Inclusión de las librerías Jersey

Copiamos y pegamos las librerías Jersey y Jackson en el directorio lib dentro del proyecto del servidor.

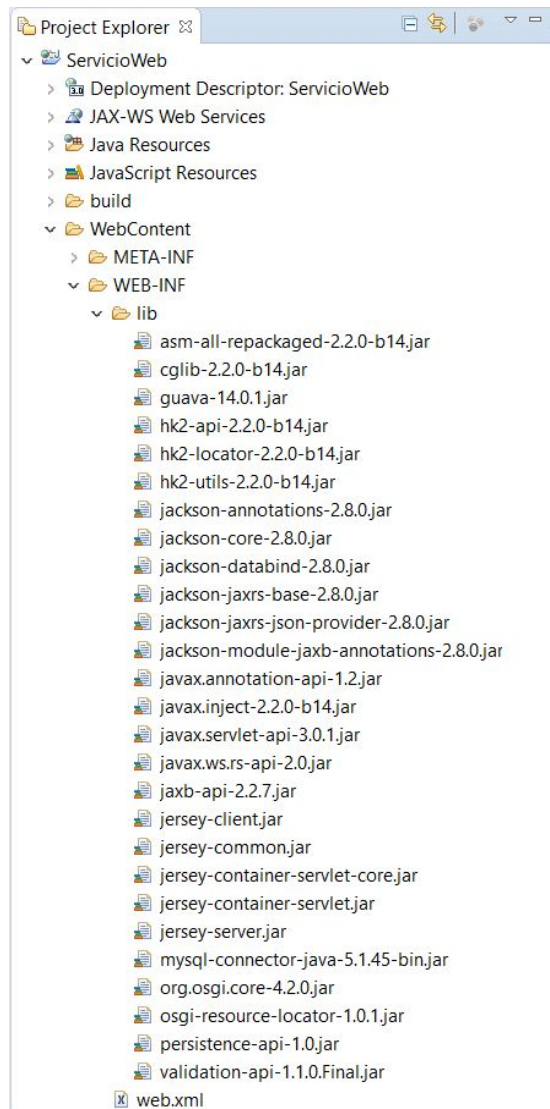


Figura 11. Inclusión de las librerías Jersey

Ahora también incluimos el fichero, junto a dichas librerías, mysql-connector-java-5.1.45-bin.jar, obsérvese en la imagen anterior.

## 1.4. Creación del paquete y de las clases

Justo ahora tenemos que crear los paquetes del proyecto y dentro de éstos las clases java que cuelgan de los mismos. Lo haremos de la siguiente forma:

### 1.4.1. Para el Servidor

Nos situamos en Java Resources, luego clic con el botón derecho del ratón y luego clic en New -> Package

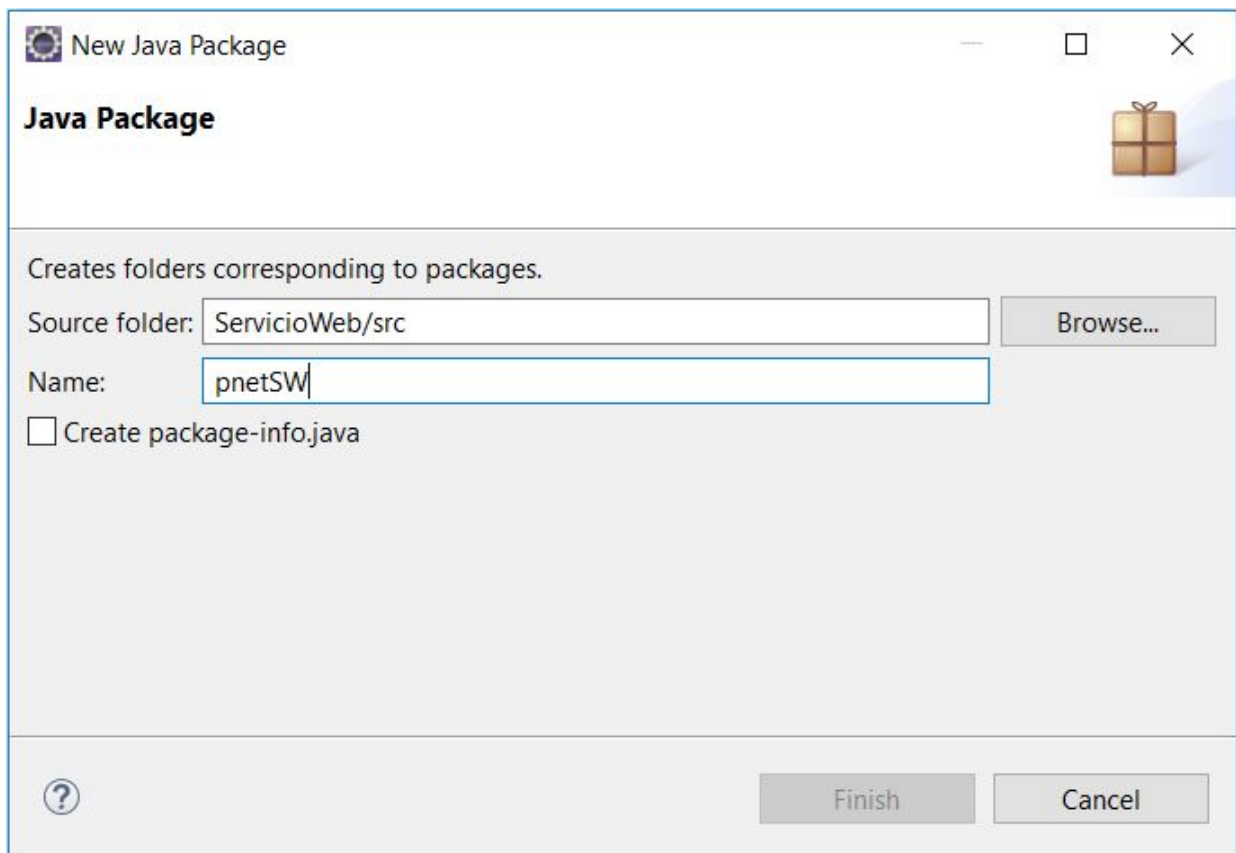


Figura 12. Creación del paquete y de las clases. El Servidor. Paso 1

Y clic en finish.

De igual modo haremos lo mismo para las clases. Clic derecho en el paquete recién creado y creamos las clases una a una: Ponente.java, Servicio.java y Servicio2.java, así:

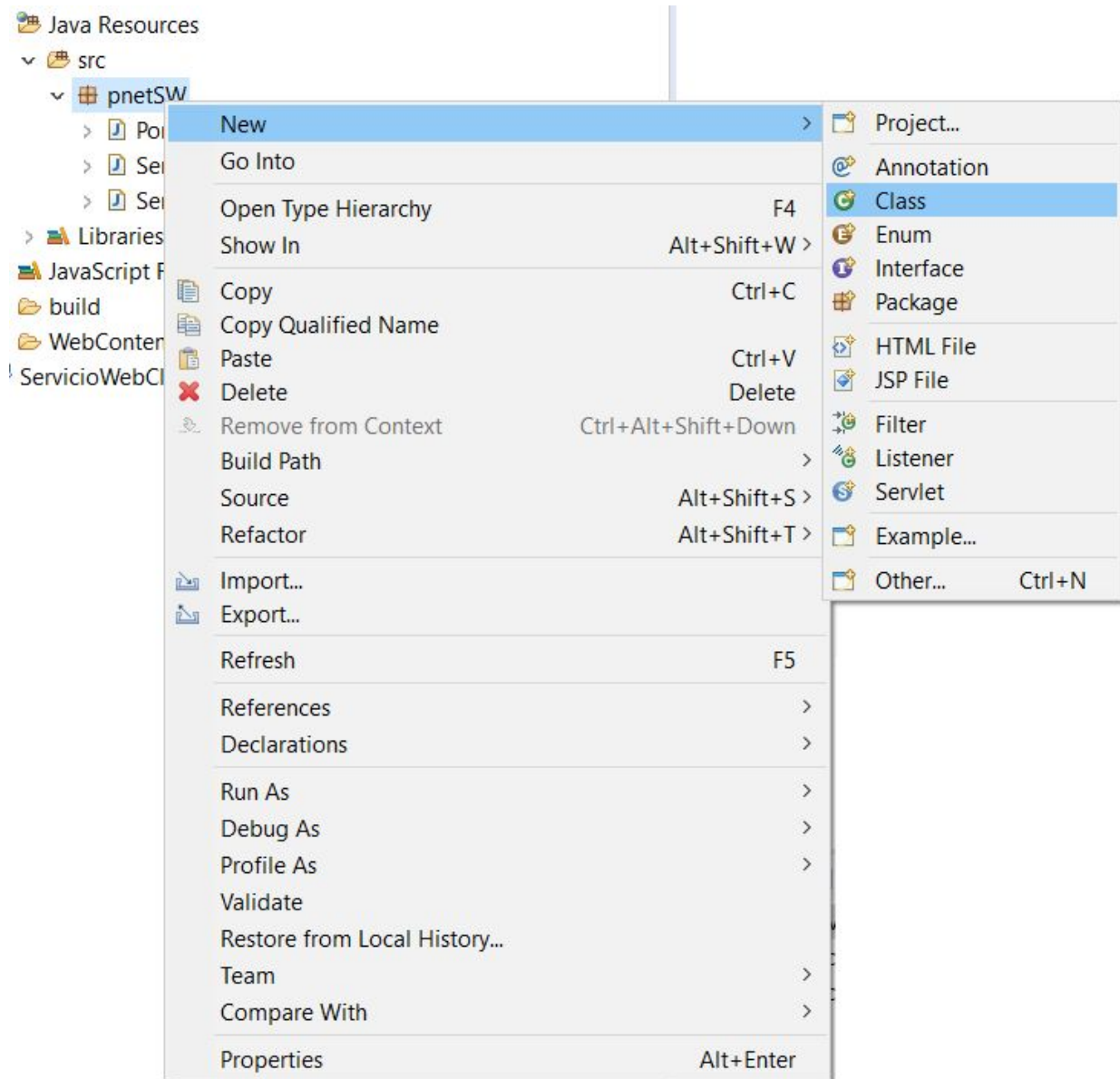


Figura 13. Creación del paquete y de las clases. El Servidor. Paso 2

Y le ponemos el nombre, la primera letra siempre con mayúsculas, al tratarse de una clase. Y repetimos la operación con el resto de las clases.

En los anexos se muestran al detalle todo el código de éstas clases. En ellas se incluyen toda la información de los servicios web programados, se han usado etiquetas para el mapeado de las clases y comunicación entre los servicios y los clientes a través de conexiones HTTP 1.1, petición/respuesta.



### 1.4.2. Para el Cliente

Para el cliente además de añadir las páginas web propias de la aplicación debemos crear las llamadas/invocaciones a los servicios web desde ficheros javascript haciendo uso de AJAX para aumentar el rendimiento en nuestro proyecto, haciéndolo más escalable, más dinámico reduciendo, de forma transparente para el usuario, el número de peticiones hacia el lado del servidor.

De esta forma nuestro proyecto quedaría así, en el lado del cliente:

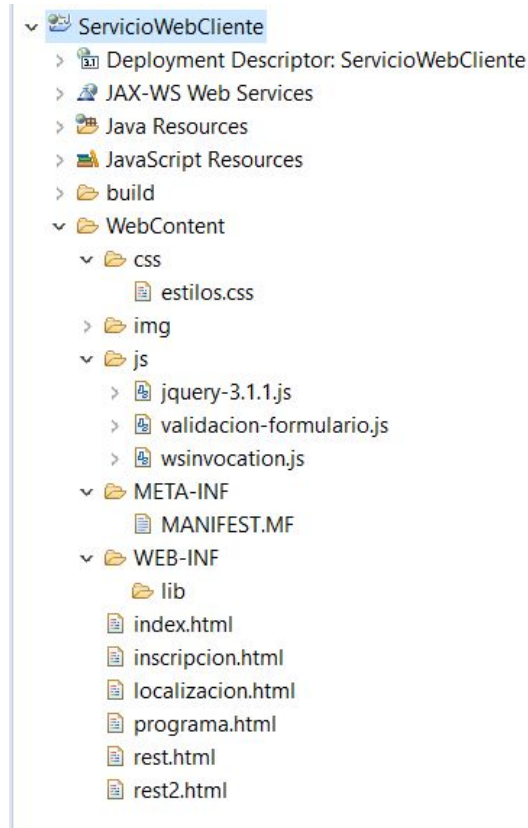


Figura 13. Creación de los ficheros del cliente

Hemos creado dos ficheros javascript dentro de la carpeta js, y hemos añadido el jquery para hacer uso de AJAX, y nótese también las páginas o vistas añadidas dentro de WebContent, sin olvidarnos de las imágenes y los estilos propios de la aplicación web. Ahora ya no hemos necesitado crear paquetes ni clases java. Sólo trabajamos con javascript en los ficheros *validación-formulario.js* y *wsinvocation.js* para hacer las llamadas a los servicios de la parte del servidor.

En los anexos mostramos el código de dichos ficheros.



## 2. Implementación del Servicio Web REST

Hemos creado dos clases para ofrecer el mismo servicio, en la primera de ellas utilizamos una estructura estática para el almacenamiento de los datos y una segunda clase que hace ya uso de una base de datos (MySQL) propuesta como mejora del proyecto. Hemos mantenido ambas y podrán ser accedidas desde el cliente de forma independiente.

### 2.1. La clase Ponente

Esta clase permite modelar un miembro del programa del congreso. Hemos decidido que tenga estos atributos, todos de tipo String:

- DNI: que funcionará además como campo de búsqueda, al ser un campo único. En el esquema de la BD lo usaremos además como la clave primaria.
- Nombre.
- Apellidos.
- Afiliación.
- País.

Además de sus correspondientes constructores (sin ningún y con todos los parámetros), hemos implementado los setters y getters de todos los atributos. Finalmente hemos desarrollado el método toString(), que nos facilitará la implementación de algún que otro método de los servicios web.

Todo lo anterior no tiene nada de particular, es una clase Java sin más. Lo más importante que tenemos en esta clase y que debemos tener en cuenta es que para que un JSON se transforme a un objeto Java, y viceversa, la clase Java tiene que poder ser serializada. Esto lo conseguimos con la etiqueta **@XmlElement**, esta etiqueta tiene que ir colocada antes de la definición de la clase que queremos que sea serializada, en nuestro caso de la clase Ponente, como puede verse en el siguiente fragmento de código (línea 5) o en el propio Anexo A.

Código 2.1. Uso de la etiqueta @XmlElement

```
1. package pnetSw;  
2.  
3. import javax.xml.bind.annotation.XmlRootElement;  
4.  
5. @XmlElement  
6. public class Ponente {  
7.     private String dni, nombre, apellidos, afiliacion, pais;  
8.  
9.     public Ponente() {}  
10.    ...  
11.    ...  
12. }
```

## 2.2. Servicios usando estructura estática (Map)

La clase que desarrollaremos para ello se llama **Servicio**, dentro de **Servicio.java**, será donde nos encontramos las funcionalidades que van a gestionar los datos almacenados de forma estática, es decir, alta, modificación, consulta y eliminación de ponentes.

La estructura de almacenamiento elegida ha sido un Map (HashMap), por parecernos la más adecuada para las operaciones de búsqueda, al no tener que hacer búsquedas secuenciales como en otro tipo de estructuras como las listas por ejemplo, aunque realmente este tipo de detalles no debiera tener importancia, ya que en entornos de producción lo normal es usar una base de datos como más adelante comentaremos.

### 2.2.1. Ruta hasta nuestro servicio

Como nuestro proyecto versaba sobre un congreso de oftalmología, hemos decidido que la ruta para llegar a este servicio sea “oftalmología”, para ello debemos indicar al principio de la clase mediante la etiqueta @Path dicha cadena (línea 3).

#### Código 2.2.1. Estableciendo la ruta al servicio

```
1. package pnetSW;
2.
3. @Path("/oftalmologia")
4. public class Servicio {
5.     ...
6. }
```

### 2.2.3. Instanciamos nuestro sistema de almacenamiento

Necesitamos un lugar donde almacenar la colección de ponentes del congreso para su consulta, actualización, etc. Este objeto es un Map, cuya clave será el DNI del ponente y el valor es el Ponente que tiene ese dni. A continuación de instanciarlo, añadiremos algunos registros a esa estructura.

#### Código 2.2.3. Sistema de almacenamiento

```
1. private static Map<String, Ponente> bbdd = new HashMap<>();
2.
3. static {
4.     Ponente p1 = new Ponente("1", "Andrés", "Martínez Gavira",
5.         "Hospital Puerta del Mar", "España");
6.     Ponente p2 = new Ponente("2", "Antonio", "Ruiz Rondán",
7.         "Hospital Virgen del Rocío", "España");
8.     Ponente p3 = new Ponente("3", "Antonio", "Banderas Malageño",
9.         "Málaga", "España");
10.    bbdd.put(p1.getDni(), p1);
```

```
11.     bbdd.put(p2.getDni(), p2);
12.     bbdd.put(p3.getDni(), p3);
13. }
```

Una vez que tenemos una instancia de este tipo, ya podemos utilizarla, la hemos llamado *bbdd*, aunque es obvio que no lo es...

## 2.2.4. Añadimos operaciones al servicio

Tal y como hemos comentado anteriormente la clase que albergará las funcionalidades es nuestra clase **Servicio**, por lo que todas las funcionalidades que vamos a implementar ahora serán desarrolladas dentro de esta.

### 2.2.4.1. GET

Con este tipo de petición HTTP vamos a desarrollar dos operaciones de nuestro servicio:

- OPERACIÓN 1: Obtener en JSON todos los miembros del comité de programa (DNI, nombre, apellidos, afiliación, país)
- OPERACIÓN 2: Obtener en modo texto los datos de un miembro concreto del comité de programa. El miembro a buscar se podrá especificar con su nombre (DNI) como parámetro del path.

#### Código 2.2.4.1. GET. Desarrollo operaciones 1 y 2 (Obtener)

```
1.  @GET
2.  @Path("/todosPonentes")
3.  @Produces({"application/json"})
4.  public Map<String, Ponente> readAllDoctors() {
5.      return bbdd;
6.  }
7.
8.  @GET
9.  @Path("/obtenerPonente/{dni}")
10. @Produces(MediaType.TEXT_PLAIN)
11. public String readOneDoctor(@PathParam("dni") String key) {
12.     String msg;
13.     try {
14.         if(bbdd.containsKey(key)) {
15.             msg = bbdd.get(key).toString();
16.         } else
17.             msg = "ERROR: No existe un ponente con DNI: " + key;
18.     } catch (Exception e) {
19.         msg = "ERROR (Excepción): " + e.getMessage();
20.     }
21.     return msg;
22. }
```

Ambos métodos tienen tres etiquetas adicionales que pasamos a explicar:

- **@GET:** con esta etiqueta indicamos que ambos métodos responderán a peticiones HTTP de tipo GET y no a otro tipo de peticiones.
- **@Path:** al igual que cuando la colocamos sobre la clase, establece ahora la manera de llegar a este método a través de la URL de nuestro navegador. Es decir especifica la ruta de acceso relativa para nuestros métodos.
- **@Produces:** especifica los tipos de medios MIME de respuesta. En el caso de la operación 1 indicamos que el método devuelve un tipo JSON como estructura y en el caso de la operación 2, dicho método devuelve texto plano.
- Además, se puede proporcionar anotaciones adicionales para los parámetros de los métodos para extraer información de la solicitud. Este es el caso del método elaborado para la operación 2, donde con la etiqueta **@PathParam** indicamos que recibimos a través de la URL el *{dni}* del ponente que queremos obtener los datos. Para ello, en la etiqueta **@Path** hemos añadido a la URL entre llaves una referencia a lo que será nuestro parámetro de entrada, para posteriormente “ligarlo” con el parámetro formal del método, que en este caso llamamos *key*.

#### 2.2.4.2. PUT

Con este tipo de petición HTTP vamos a implementar la OPERACIÓN 3, es decir, actualizar los datos de un miembro, pasando como parámetro del path el {DNI} y los nuevos datos a actualizar como JSON. Volvemos a insistir que para que un JSON se transforme a un objeto Java, y viceversa, la clase Java tiene que poder ser serializada. Esto lo conseguimos con la etiqueta **@XmlElement** sobre la clase Ponente.

##### Código 2.2.4.2. PUT. Operación 3 (Actualizar)

```
1.  @PUT
2.  @Path("/modificarPonente/{dni}")
3.  @Consumes(MediaType.APPLICATION_JSON)
4.  @Produces(MediaType.TEXT_PLAIN)
5.  public String updateDoctor(@PathParam("dni") String key, Ponente p) {
6.      String msg;
7.
8.      try {
9.          if(bbdd.containsKey(key)) {
10.             Ponente actP = bbdd.get(key);
11.             if(p.getNombre() != null && !p.getNombre().isEmpty())
12.                 actP.setNombre(p.getNombre());
13.             if(p.getApellidos() != null && !p.getApellidos().isEmpty())
14.                 actP.setApellidos(p.getApellidos());
15.             if(p.getAfiliacion() != null && !p.getAfiliacion().isEmpty())
16.                 actP.setAfiliacion(p.getAfiliacion());
17.             if(p.getPais() != null && !p.getPais().isEmpty())
18.                 actP.setPais(p.getPais());
19.             bbdd.put(key, actP);
20.             msg = "Ponente actualizado correctamente";
21.         } else
22.             msg = "ERROR: No existe un ponente con DNI: " + key;
```

```
23.     } catch (Exception e) {
24.         msg = "ERROR (Excepción): " + e.getMessage();
25.     }
26.
27.     return msg;
28. }
```

Como novedoso en este método encontramos la etiqueta **@Consumes** que se utiliza para especificar los tipos de medios de petición aceptados. En nuestro caso proporcionaremos a nuestro método un objeto JSON. Ahora no hemos notificado en los parámetros de entrada el objeto JSON que vamos a recibir, sino que se deberá proporcionar este objeto dentro del BODY de la petición PUT que el cliente solicite.

#### 2.2.4.3. DELETE

La OPERACIÓN 4 de nuestro servicio será implementado con este tipo de petición HTTP, es decir, se trata de borrar un miembro del comité de programa, pasando su {DNI} como parámetro del path. Como podemos observar en el código, no presenta nada de dificultad, tampoco aparece ninguna etiqueta nueva que necesitemos explicar.

##### Código 2.2.4.3. DELETE. Operación 4 (Eliminar)

```
1.  @DELETE
2.  @Path("/eliminarPonente/{dni}")
3.  @Produces(MediaType.TEXT_PLAIN)
4.  public String deleteDoctor(@PathParam("dni") String key){
5.      String msg;
6.
7.      try {
8.          if(bbdd.containsKey(key)) {
9.              bbdd.remove(key);
10.             msg = "Eliminado correctamente ponente con DNI: " + key;
11.         } else
12.             msg = "ERROR: No existe un ponente con DNI: " + key;
13.     } catch (Exception e) {
14.         msg = "ERROR (Excepción): " + e.getMessage();
15.     }
16.
17.     return msg;
18. }
```

#### 2.2.4.4. POST

Implementaremos con este tipo de petición HTTP, dos operaciones de nuestro proyecto.

- OPERACIÓN 5: Añadir un nuevo miembro del comité de programa pasándolo como JSON.
- OPERACIÓN opcional 1: Añadir un nuevo miembro del comité de programa pasándolo como parámetro de formulario.

##### Código 2.2.4.4. POST. Operación 5 y Operación opcional 1 (Añadir)

```
1.  @POST
2.  @Path("/nuevoPonente")
3.  @Consumes(MediaType.APPLICATION_JSON)
4.  @Produces(MediaType.TEXT_PLAIN)
5.  public String createDoctor(Ponente p){
6.      String msg;
7.
8.      try {
9.          if(p.getDni() == null || p.getDni().isEmpty()) {
10.              msg = "Tienes que indicar un DNI";
11.          } else if(!p.getDni().matches("\\d+")) {
12.              msg = "Tienes que introducir un número para el DNI";
13.          } else if(!bbdd.containsKey(p.getDni())) {
14.              bdd.put(p.getDni(), p);
15.              msg = "Ponente añadido correctamente";
16.          } else
17.              msg = "ERROR: Existe un ponente con DNI: " + p.getDni();
18.      } catch (Exception e) {
19.          msg = "ERROR (Excepción): " + e.getMessage();
20.      }
21.
22.      return msg;
23.  }
24.
25.
26.  @POST
27.  @Path("/nuevoPonenteForm")
28.  @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
29.  @Produces(MediaType.TEXT_PLAIN)
30.  public String createDoctorForm(@FormParam("dni") String dni,
31.  @FormParam("nom") String nom, @FormParam("ape") String ape,
32.  @FormParam("afi") String afi, @FormParam("pais") String pais) {
33.      String msg;
34.
35.      try {
36.          if(dni == null || dni.isEmpty()) {
37.              msg = "Tienes que indicar un DNI";
38.          } else if(!dni.matches("\\d+")) {
39.              msg = "Tienes que introducir un número para el DNI";
40.          } else if(!bbdd.containsKey(dni)) {
41.              Ponente p = new Ponente(dni, nom, ape, afi, pais);
42.              bdd.put(dni, p);
43.              msg = "Ponente añadido correctamente";
44.          } else
45.              msg = "ERROR: Ya existe un ponente con DNI: " + dni;
46.      } catch (Exception e) {
```

```
47.     msg = "ERROR (Excepción): " + e.getMessage();
48.     }
49.
50.     return msg;
51. }
```

En el primero de los métodos enviamos los datos del nuevo ponente en el body de la petición tipo POST, ya lo hicimos igual anteriormente en la de tipo PUT. Pero en el segundo método añadimos estos datos en un formulario html. En este caso tendremos que indicar en la lista de argumentos de entrada cuáles son los parámetros que le pasamos mediante el formulario y el nombre que le pusimos en el mismo. Para esto se utiliza la etiqueta **@FormParam** que enlaza el parámetro a un valor de formulario. Importante también indicar en la etiqueta **@Consumes**, que se le pasará un formulario en el body de la petición.

### 2.2.5. Inicio del servicio. Resumen de rutas para pruebas

Llegados a este punto ya tenemos implementado nuestro servicio, tenemos el software necesario (servidor Tomcat, bibliotecas necesarias, Java...) para ejecutarlo y tenemos nuestro fichero **web.xml** correctamente descrito para un adecuado despliegue. Ya es hora de probarlo, lo haremos posteriormente cuando desarrollemos el cliente y el conjunto de funciones jQuery apropiadas para invocar a los servicios. De todas formas pueden invocarse utilizando cualquier aplicación como [Postman](#), utilizando las siguientes URLs:

- GET
  - <http://localhost:8080/ServicioWeb/oftalmologia/todosPonentes>
  - <http://localhost:8080/ServicioWeb/oftalmologia/obtenerPonente/{dni}>
- PUT
  - <http://localhost:8080/ServicioWeb/oftalmologia/modificarPonente/{dni}>
- DELETE
  - <http://localhost:8080/ServicioWeb/oftalmologia/eliminarPonente/{dni}>
- POST
  - <http://localhost:8080/ServicioWeb/oftalmologia/nuevoPonente>
  - <http://localhost:8080/ServicioWeb/oftalmologia/nuevoPonenteForm>

Cuando desarrollemos el cliente veremos dónde utilizar estas URLs.

## 2.3. Servicios usando una base de datos (MySQL)

Dentro de **Servicio2.java**, será donde nos encontramos las funcionalidades que van a gestionar los datos almacenados de forma permanente, es decir, alta, modificación, consulta y eliminación de ponentes, esta vez utilizando una base de datos relacional. Como lo hacíamos en el punto anterior, tras reiniciar el servicio, volvíamos a tener los mismos datos de partida, perdiéndose las modificaciones efectuadas sobre la estructura elegida (Map).

También decíamos que en un entorno de producción lo adecuado es utilizar un SGBD como *Oracle Database*, *PostgreSQL*, *Firebird*, *Microsoft SQL Server*, *SQLite* o **MySQL**, este último es el que hemos elegido, pero podría haberse utilizado cualquiera de los otros. En cualquiera de los casos el procedimiento es muy similar a lo que vamos a describir a continuación.

Lo primero que tenemos que hacer es descargarnos el driver JDBC para MySQL del área de descargas de la web oficial de [MySQL](#), y ubicar el correspondiente .jar dentro del directorio **lib** (donde se ubicaron también los ficheros de Jersey), en nuestro caso el fichero descargado se llama *mysql-connector-java-5.1.45-bin.jar*.

También debemos tener un servicio MySQL instalado y configurado. Obviamente tendremos la BD creada y su correspondiente tabla, aportamos DDL y DML en Anexo E.

### 2.3.1. Ruta hasta nuestro servicio

Como el servicio es el mismo, es decir, sobre un congreso de oftalmología, hemos decidido que la ruta para llegar a este servicio sea “oftalmologia2”, para ello debemos indicar al principio de la clase mediante la etiqueta `@Path` dicha cadena (línea 3).

#### Código 2.3.1. Estableciendo la ruta al servicio

```
1. package pnetSw;
2.
3. @Path("/oftalmologia2")
4. public class Servicio2 {
5.     private Connection conexion = null;
6.     private Statement comando = null;
7.     private PreparedStatement ps = null;
8.     private ResultSet registro = null;
9.     private String consulta;
10.    private static AtomicInteger totalPonentes = new AtomicInteger(total());
11.    ...
12.
13.    /**
14.     * OPERACIÓN opcional 3
15.     * Devuelve el nº total de ponentes.
16.     *
17.     * @return nº de ponentes
18.     */
19.    @GET
20.    @Path("/contarPonentes2")
21.    @Produces(MediaType.TEXT_PLAIN)
22.    public String countDoctors() {
```



```
23.     return totalPonentes.toString();
24. }
25.
26.
27. /**
28.  * Devuelve el total de registros contenidos en la tabla ponentes.
29.  *
30.  * @return nº filas de la tabla ponentes. -1 si error.
31.  */
32. private static int total() {
33.     Connection con = null;
34.     Statement sta = null;
35.     ResultSet res = null;
36.     int t = -1;
37.
38.     try {
39.         con = MySQLConnect();
40.         String consulta = "SELECT COUNT(*) FROM ponentes";
41.         sta = con.createStatement();
42.         res = sta.executeQuery(consulta);
43.         res.next();
44.         t = res.getInt(1);
45.     } catch (Exception e) {
46.         System.err.println(e.getMessage());
47.     } finally {
48.         try {
49.             if (res != null) {
50.                 res.close();
51.             }
52.
53.             if (sta != null) {
54.                 sta.close();
55.             }
56.
57.             if (con != null) {
58.                 con.close();
59.             }
60.         } catch (Exception ex) {
61.             System.err.println(ex.getMessage());
62.         }
63.     }
64.
65.     return t;
66. }
67. ...
68. ...
69. }
```

### 2.3.3. Establecemos cómo conectarnos con nuestro servidor MySQL

A continuación detallamos el método encargado de establecer la conexión con nuestro servidor MySQL

#### Código 2.3.3. Conexión con MySQL (y desconexión)

```
1. private static Connection MySQLConnect() {
2.     Connection con = null;
3.     try {
4.         Class.forName("com.mysql.jdbc.Driver");
5.         String user = "pnet";
6.         String pass = "pnet";
7.         String name = "pnet";
8.         String host = "127.0.0.1";
9.         int port = 3306;
10.        String url = String.format("jdbc:mysql://s:%d/%s?useSSL=false", host, port, name);
11.        con = DriverManager.getConnection(url, user, pass);
12.
13.        if (con != null) {
14.            System.out.println("Conexión a base de datos " + url + " ... Ok");
15.        }
16.    } catch (Exception ex) {
17.        System.err.println(ex);
18.    }
19.
20.    return con;
21. }
```

```
1. private void cerrarTodo() {
2.     try {
3.         if (registro != null) {
4.             registro.close();
5.         }
6.
7.         if (ps != null) {
8.             ps.close();
9.         }
10.
11.        if (comando != null) {
12.            comando.close();
13.        }
14.
15.        if (conexion != null) {
16.            conexion.close();
17.        }
18.    } catch (Exception ex) {
19.        System.err.println(ex.getMessage());
20.    }
21. }
```

Obtendremos una de estas conexiones en cada uno de las llamadas a los servicios que implementemos, por eso cerramos en todos esos métodos la conexión.

### 2.3.4. Añadiendo operaciones al servicio

Pasemos a desarrollar las operaciones solicitadas.

#### 2.3.4.1. GET

Con este tipo de petición HTTP vamos a desarrollar dos operaciones de nuestro servicio:

- OPERACIÓN 1: Obtener en JSON todos los miembros del comité de programa (DNI, nombre, apellidos, afiliación, país)
- OPERACIÓN 2: Obtener en modo texto los datos de un miembro concreto del comité de programa. El miembro a buscar se podrá especificar con su nombre (DNI) como parámetro del path.

##### Código 2.3.4.1. GET. Desarrollo operaciones 1 y 2 (Obtener)

```
1.  @GET
2.  @Path("/todosPonentes")
3.  @Produces({"application/json"})
4.  public Map<String, Ponente> readAllDoctors() {
5.      Map<String, Ponente> bbdd = new HashMap<>();
6.
7.      try {
8.          conexion = MySQLConnect();
9.          consulta = "SELECT * FROM ponentes";
10.         comando = conexion.createStatement();
11.         registro = comando.executeQuery(consulta);
12.         System.out.println(consulta);
13.
14.         while(registro.next()) {
15.             bbdd.put(registro.getString(1),
16.                 new Ponente(registro.getString(1), registro.getString(2),
17.                     registro.getString(3), registro.getString(4),
18.                     registro.getString(5)));
19.         }
20.     } catch (Exception ex) {
21.         System.err.println(ex);
22.     } finally {
23.         cerrarTodo();
24.     }
25.
26.     return bbdd;
27. }
28.
29.
30.
31. @GET
32. @Path("/obtenerPonente/{dni}")
33. @Produces(MediaType.TEXT_PLAIN)
34. public String readOneDoctor(@PathParam("dni") String key) {
35.     String msg;
36.     try {
37.         conexion = MySQLConnect();
38.         consulta = "SELECT * FROM ponentes WHERE dni = ?";
```

```
39.     ps = conexion.prepareStatement(consulta);
40.     ps.setString(1, key);
41.     registro = ps.executeQuery();
42.     System.out.println(ps.toString());
43.     registro.next();
44.     msg = "DNI: " + registro.getString(1);
45.     msg += "\nNOMBRE: " + registro.getString(2);
46.     msg += "\nAPELLIDOS: " + registro.getString(3);
47.     msg += "\nAFILIACIÓN: " + registro.getString(4);
48.     msg += "\nPAÍS: " + registro.getString(5);
49. } catch (SQLException ex) {
50.     msg = "ERROR: No existe un ponente con DNI: " + key;
51. } catch (Exception ex) {
52.     msg = "ERROR (Excepción): " + ex.getMessage();
53. } finally {
54.     cerrarTodo();
55. }
56.
57.     return msg;
58. }
```

Ambos métodos tienen tres etiquetas adicionales que pasamos a explicar:

- **@GET:** con esta etiqueta indicamos que ambos métodos responderán a peticiones HTTP de tipo GET y no a otro tipo de peticiones.
- **@Path:** al igual que cuando la colocamos sobre la clase, establece ahora la manera de llegar a este método a través de la URL de nuestro navegador. Es decir especifica la ruta de acceso relativa para nuestros métodos.
- **@Produces:** especifica los tipos de medios MIME de respuesta. En el caso de la operación 1 indicamos que el método devuelve un tipo JSON como estructura y en el caso de la operación 2, dicho método devuelve texto plano.
- Además, se puede proporcionar anotaciones adicionales para los parámetros de los métodos para extraer información de la solicitud. Este es el caso del método elaborado para la operación 2, donde con la etiqueta **@PathParam** indicamos que recibimos a través de la URL el *{dni}* del ponente que queremos obtener los datos. Para ello, en la etiqueta @Path hemos añadido a la URL entre llaves una referencia a lo que será nuestro parámetro de entrada, para posteriormente “ligarlo” con el parámetro formal del método, que en este caso llamamos *key*.

### 2.3.4.2. PUT

Con este tipo de petición HTTP vamos a implementar la OPERACIÓN 3, es decir, actualizar los datos de un miembro, pasando como parámetro del path el {DNI} y los nuevos datos a actualizar como JSON. Volvemos a insistir que para que un JSON se transforme a un objeto Java, y viceversa, la clase Java tiene que poder ser serializada. Esto lo conseguimos con la etiqueta `@XmlElement` sobre la clase Ponente.

#### Código 2.3.4.2. PUT. Operación 3 (Actualizar)

```
1.  @PUT
2.  @Path("/modificarPonente/{dni}")
3.  @Consumes(MediaType.APPLICATION_JSON)
4.  @Produces(MediaType.TEXT_PLAIN)
5.  public String updateDoctor(@PathParam("dni") String key, Ponente p) {
6.      String msg;
7.
8.      try {
9.          conexion = MySQLConnect();
10.         consulta = "SELECT COUNT(*) FROM ponentes WHERE dni = ?";
11.         ps = conexion.prepareStatement(consulta);
12.         ps.setString(1, key);
13.         registro = ps.executeQuery();
14.         registro.next();
15.         if(registro.getInt(1) > 0) {
16.             consulta = "UPDATE ponentes SET nombre = ?, apellidos = ?";
17.             consulta += ", afiliacion = ?, pais = ? WHERE dni = ?";
18.             ps = conexion.prepareStatement(consulta);
19.             ps.setString(1, p.getNombre());
20.             ps.setString(2, p.getApellidos());
21.             ps.setString(3, p.getAfiliacion());
22.             ps.setString(4, p.getPais());
23.             ps.setString(5, key);
24.             ps.executeUpdate();
25.             System.out.println(ps.toString());
26.             msg = "Ponente actualizado correctamente";
27.         } else
28.             msg = "ERROR: No existe un ponente con DNI: " + key;
29.     } catch (Exception e) {
30.         msg = "ERROR (Excepción): " + e.getMessage();
31.     } finally {
32.         cerrarTodo();
33.     }
34.
35.     return msg;
36. }
```

Como novedoso en este método encontramos la etiqueta `@Consumes` que se utiliza para especificar los tipos de medios de petición aceptados. En nuestro caso proporcionaremos a nuestro método un objeto JSON. Ahora no hemos notificado en los parámetros de entrada el objeto JSON que vamos a recibir, sino que se deberá proporcionar este objeto dentro del BODY de la petición PUT que el cliente solicite.

### 2.3.4.3. DELETE

La OPERACIÓN 4 de nuestro servicio será implementado con este tipo de petición HTTP, es decir, se trata de borrar un miembro del comité de programa, pasando su {DNI} como parámetro del path. Como podemos observar en el código, no presenta nada de dificultad, tampoco aparece ninguna etiqueta nueva que necesitemos explicar.

Código 2.3.4.3. DELETE. Operación 4 (Eliminar)

```
1.  @DELETE
2.  @Path("/eliminarPonente/{dni}")
3.  @Produces(MediaType.TEXT_PLAIN)
4.  public String deleteDoctor(@PathParam("dni") String key){
5.      String msg;
6.
7.      try {
8.          conexion = MySQLConnect();
9.          consulta = "SELECT COUNT(*) FROM ponentes WHERE dni = ?";
10.         ps = conexion.prepareStatement(consulta);
11.         ps.setString(1, key);
12.         registro = ps.executeQuery();
13.         registro.next();
14.         if(registro.getInt(1) > 0) {
15.             consulta = "DELETE FROM ponentes WHERE dni = ?";
16.             ps = conexion.prepareStatement(consulta);
17.             ps.setString(1, key);
18.             ps.executeUpdate();
19.             totalPonentes.decrementAndGet();
20.             msg = "Eliminado correctamente ponente con DNI: " + key;
21.             System.out.println(ps.toString());
22.         } else
23.             msg = "ERROR: No existe un ponente con DNI: " + key;
24.     } catch (Exception e) {
25.         msg = "ERROR (Excepción): " + e.getMessage();
26.     } finally {
27.         cerrarTodo();
28.     }
29.
30.     return msg;
31. }
```

#### 2.3.4.4. POST

Implementaremos con este tipo de petición HTTP, dos operaciones de nuestro proyecto.

- OPERACIÓN 5: Añadir un nuevo miembro del comité de programa pasándolo como JSON.
- OPERACIÓN opcional 1: Añadir un nuevo miembro del comité de programa pasándolo como parámetro de formulario.

##### Código 2.3.4.4. POST. Operación 5 y Operación opcional 1 (Añadir)

```
1.  @POST
2.  @Path("/nuevoPonente")
3.  @Consumes(MediaType.APPLICATION_JSON)
4.  @Produces(MediaType.TEXT_PLAIN)
5.  public String createDoctor(Ponente p){
6.      String msg;
7.
8.      try {
9.          conexion = MySQLConnect();
10.         consulta = "SELECT COUNT(*) FROM ponentes WHERE dni = ?";
11.         ps = conexion.prepareStatement(consulta);
12.         ps.setString(1, p.getDni());
13.         registro = ps.executeQuery();
14.         registro.next();
15.         if(p.getDni() == null || p.getDni().isEmpty()) {
16.             msg = "Tienes que indicar un DNI";
17.         } else if(!p.getDni().matches("\\d+")) {
18.             msg = "Tienes que introducir un número para el DNI";
19.         } else if(registro.getInt(1) == 0) {
20.             consulta = "INSERT INTO ponentes (dni, nombre, apellidos, afiliacion, pais)";
21.             consulta += "VALUES (?, ?, ?, ?, ?)";
22.             ps = conexion.prepareStatement(consulta);
23.             ps.setString(1, p.getDni());
24.             ps.setString(2, p.getNombre());
25.             ps.setString(3, p.getApellidos());
26.             ps.setString(4, p.getAfiliacion());
27.             ps.setString(5, p.getPais());
28.             ps.executeUpdate();
29.             totalPonentes.incrementAndGet();
30.             System.out.println(ps.toString());
31.             msg = "Ponente añadido correctamente";
32.         } else
33.             msg = "ERROR: Existe un ponente con DNI: " + p.getDni();
34.         catch (Exception e) {
35.             msg = "ERROR (Excepción): " + e.getMessage();
36.         } finally {
37.             cerrarTodo();
38.         }
39.
40.         return msg;
41.     }
42.
43.
44.
45.
46.  @POST
```

```
47. @Path("/nuevoPonenteForm")
48. @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
49. @Produces(MediaType.TEXT_PLAIN)
50. public String createDoctorForm(@FormParam("dni") String dni,
51.    @FormParam("nom") String nom, @FormParam("ape") String ape,
52.    @FormParam("afi") String afi, @FormParam("pais") String pais) {
53.
54.    String msg;
55.
56.    try {
57.        conexion = MySQLConnect();
58.        consulta = "SELECT COUNT(*) FROM ponentes WHERE dni = ?";
59.        ps = conexion.prepareStatement(consulta);
60.        ps.setString(1, dni);
61.        registro = ps.executeQuery();
62.        registro.next();
63.        if(dni == null || dni.isEmpty()) {
64.            msg = "Tienes que indicar un DNI";
65.        } else if(!dni.matches("\\d+")) {
66.            msg = "Tienes que introducir un número para el DNI";
67.        } else if(registro.getInt(1) == 0) {
68.            consulta = "INSERT INTO ponentes (dni, nombre, apellidos, afiliacion, pais) ";
69.            consulta += "VALUES (?, ?, ?, ?, ?)";
70.            ps = conexion.prepareStatement(consulta);
71.            ps.setString(1, dni);
72.            ps.setString(2, nom);
73.            ps.setString(3, ape);
74.            ps.setString(4, afi);
75.            ps.setString(5, pais);
76.            ps.executeUpdate();
77.            totalPonentes.incrementAndGet();
78.            System.out.println(ps.toString());
79.            msg = "Ponente añadido correctamente";
80.        } else
81.            msg = "ERROR: Ya existe un ponente con DNI: " + dni;
82.        } catch (Exception e) {
83.            msg = "ERROR (Excepción): " + e.getMessage();
84.        } finally {
85.            cerrarTodo();
86.        }
87.
88.        return msg;
89.    }
```

En el primero de los métodos enviamos los datos del nuevo ponente en el body de la petición tipo POST, ya lo hicimos igual anteriormente en la de tipo PUT. Pero en el segundo método añadimos estos datos en un formulario html. En este caso tendremos que indicar en la lista de argumentos de entrada cuáles son los parámetros que le pasamos mediante el formulario y el nombre que le pusimos en el mismo. Para esto se utiliza la etiqueta **@FormParam** que enlaza el parámetro a un valor de formulario. Importante también indicar en la etiqueta **@Consumes**, que se le pasará un formulario en el body de la petición.



### **2.3.5. Inicio del servicio. Resumen de rutas para pruebas**

Llegados a este punto ya tenemos implementado nuestro servicio, tenemos el software necesario (servidor Tomcat, bibliotecas necesarias, Java...) para ejecutarlo y tenemos nuestro fichero **web.xml** correctamente descrito para un adecuado despliegue. Ya es hora de probarlo, lo haremos posteriormente cuando desarrollemos el cliente y el conjunto de funciones jQuery apropiadas para invocar a los servicios. De todas formas pueden invocarse utilizando cualquier aplicación como [Postman](#), utilizando las siguientes URLs:

- GET
  - `http://localhost:8080/ServicioWeb/oftalmologia2/todosPonentes`
  - `http://localhost:8080/ServicioWeb/oftalmologia2/obtenerPonente/{dni}`
- PUT
  - `http://localhost:8080/ServicioWeb/oftalmologia2/modificarPonente/{dni}`
- DELETE
  - `http://localhost:8080/ServicioWeb/oftalmologia2/eliminarPonente/{dni}`
- POST
  - `http://localhost:8080/ServicioWeb/oftalmologia2/nuevoPonente`
  - `http://localhost:8080/ServicioWeb/oftalmologia2/nuevoPonenteForm`

Ahora vamos a desarrollar el cliente donde veremos cómo utilizar estas URLs.



### 3. Implementación del Cliente para invocar servicios Restful

En primer lugar necesitaremos descargar el framework javascript jQuery, nosotros tenemos la versión 3.1.1 y los colocamos dentro del directorio js creado en el punto 1.2.2 de este documento. Asimismo crearemos un fichero javascript para crear las invocaciones a los servicios, dicho fichero lo llamaremos **wsinvocation.js**, también lo ubicamos en ese mismo directorio.

Creamos un fichero html como el siguiente:

Código 3. Ejemplo html para probar servicios.

```
1. <!DOCTYPE HTML>
2. <html lang="es">
3.   <head>
4.     <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
5.     <title>RestFull - I Congreso de Oftalmologia</title>
6.     <script type="text/javascript" src="js/jquery-3.1.1.js"></script>
7.     <script type="text/javascript" src="js/wsinvocation.js"></script>
8.   </head>
9.
10.  <body>
11.    <hr>
12.    <h1>Resultados</h1>
13.    <div id="consola"></div>
14.    <hr>
15.
16.    <p>Operación 1</p>
17.    <input type="button" value="Todos" onclick="todosPonentes('oftalmologia')"><br><hr>
18.
19.    <p>Operación 2</p>
20.    <label for="dni2">DNI</label>
21.    <input type="text" name="dni2" id="dni2"><br>
22.    <input type="button" value="Obtener" onclick="obtenerPonente('oftalmologia')"><br><hr>
23.
24.    <p>Operación 3</p>
25.    <label for="dni3">DNI</label>
26.    <input type="text" name="dni3" id="dni3"><br>
27.    <label for="nom3">Nombre</label>
28.    <input type="text" name="nom3" id="nom3"><br>
29.    <label for="ape3">Apellidos</label>
30.    <input type="text" name="ape3" id="ape3"><br>
31.    <label for="afi3">Afiliación</label>
32.    <input type="text" name="afi3" id="afi3"><br>
33.    <label for="pais3">País</label>
34.    <input type="text" name="pais3" id="pais3"><br>
35.    <input type="button" value="Actualizar"
    onclick="modificarPonente('oftalmologia')"><br><hr>
36.
```

```
37. <p>Operación 4</p>
38. <label for="dni4">DNI</label>
39. <input type="text" name="dni4" id="dni4"><br>
40. <input type="button" value="Eliminar"
    onclick="eliminarPonente('oftalmologia')"><br><hr>
41.
42. <p>Operación 5</p>
43. <label for="dni5">DNI</label>
44. <input type="text" name="dni5" id="dni5"><br>
45. <label for="nom5">Nombre</label>
46. <input type="text" name="nom5" id="nom5"><br>
47. <label for="ape5">Apellidos</label>
48. <input type="text" name="ape5" id="ape5"><br>
49. <label for="afi5">Afiliación</label>
50. <input type="text" name="afi5" id="afi5"><br>
51. <label for="pais5">País</label>
52. <input type="text" name="pais5" id="pais5"><br>
53. <input type="button" value="AñadirJSON" onclick="nuevoPonente('oftalmologia')"><br><hr>
54.
55. <p>OPERACIÓN opcional 1: Añadir un nuevo ponente pasándolo como
56.   parámetro de un formulario</p>
57. <form action="http://localhost:8080/ServicioWeb/oftalmologia/nuevoPonenteForm"
58.       id="idForm" method="POST">
59. <div id="consolaForm"></div>
60. <label for="dni">DNI</label>
61. <input type="text" name="dni" id="dni" required><br>
62. <label for="nom">Nombre</label>
63. <input type="text" name="nom" id="nom"><br>
64. <label for="ape">Apellidos</label>
65. <input type="text" name="ape" id="ape"><br>
66. <label for="afi">Afiliación</label>
67. <input type="text" name="afi" id="afi"><br>
68. <label for="pais">País</label>
69. <input type="text" name="pais" id="pais"><br>
70. <input type="button" value="AñadirFORM" onclick="nuevoPonenteForm()">
71.
72. </body>
73. </html>
```

El código html anterior será para probar el servicio que implementa la estructura estática en memoria con un Map, lo sabemos porque en las llamadas a las funciones jQuery lo acompañamos con el parámetro *'oftalmologia'*, si lo hacemos con *'oftalmologia2'* invocamos a los servicios implementados con MySQL, por tanto el fichero html para probar este servicio es exactamente igual a excepción de esta cadena en la invocación.

Nosotros hemos eliminado de este documento todas las referencias a aspectos estéticos de la aplicación para mayor claridad del código, obviamente en el código final sí va incluido, ya que hemos tenido que integrar este servicio en el sitio web que desarrollamos en la práctica anterior.

### 3.1. Comprendiendo las URLs de invocación de servicios

Anteriormente hemos resumido las URLs que vamos a necesitar para invocar los servicios en los puntos 2.2.5. y 2.3.5. vamos a ver cómo se construyen esas URLs y por qué.

**http://host:port/NombreServicio/Path(clase)/Path(método)/Parámetros**

Pasamos a explicar cada una de las partes de esta URL::

- **host:** Dirección IP del servidor.
- **port:** Puerto por el cual está escuchando Tomcat, por defecto es el 8080.
- **NombreServicio:** Este parámetro hace referencia al nombre dado a nuestro servicio Web, en nuestro caso sería 'ServicioWeb'.
- **servlet-mapping:** Cuando vimos la creación del fichero web.xml vimos que en la etiqueta <servlet-mapping>, dentro de <url-pattern> pusimos que se podría acceder a través de cualquier URL, por lo que este parámetro no existe en nuestro caso. Si hubiésemos puesto otra cosa que no fuera /\*, formaría parte de nuestra URL.
- **Path(clase):** Parámetro @Path especificado en la clase.
- **Path(método):** Parámetro @Path especificado en los métodos de la clase.
- **Parámetros:** Estos serían los parámetros que pasamos a los métodos de nuestra clase. Podrían o no existir.

### 3.2. Creamos las funciones jQuery

Dentro de *wsinvocation.js* vamos a desarrollar las siguientes funciones. Pero antes definimos las siguientes variables globales:

Código 3.2. Variables globales para construir la url

```
1. var protocolo = window.location.protocol + "://";
2. var host = window.location.host; //en producción indicar IP pública o dominio
3. var servicio = "/ServicioWeb/";
4. var url = protocolo + host + servicio;
```

### 3.2.1. Función todosPonentes()

Esta función invocará al primero de los métodos que implementamos en nuestro servicio. En la petición AJAX en jQuery podemos apreciar varios parámetros que determinan:

- **type:** tipo de petición que vamos a realizar. En nuestro caso, queremos un listado completo, es de tipo GET.
- **url:** establece la URL de acceso al servicio. En nuestro caso es `http://localhost:8080/ServicioWeb/oftalmologia/todosPonentes`
- **success:** aquí definimos qué hacer con la respuesta del servidor cuando ha tenido éxito. Nosotros devolvemos una tabla html con los datos de los ponentes.
- **error:** si ha ocurrido un error definimos aquí qué hacemos, en nuestro caso mostrar un mensaje de error.
- **u:** añadimos a algunas funciones el parámetro 'u' el cual nos servirá para ir construyendo a su vez el parámetro url. Simplemente será una cadena, que podrá ser:
  - "oftalmologia": para invocar a los métodos de Servicio.java
  - "oftalmologia2": para invocar a los métodos de Servicio2.java
- **cache:** es importante recalcar que debemos establecer este parámetro a false en todas las funciones para evitar que el navegador web cachee tras cada petición y no podamos ver los resultados deseados en sucesiones peticiones del mismo servicio. Hasta que nos dimos cuenta de este aspecto, nos dió algún que otro dolor de cabeza durante unas horas.
- **dataType:** indica qué tipo de respuesta estamos esperando del servidor cuando tiene éxito. Sabemos que nuestro servicio web devuelve un json.

#### Código 3.2.1. Función todosPonentes()

```
1. function todosPonentes(u) {
2.   $.ajax({
3.     type: "GET",
4.     //url: "http://localhost:8080/ServicioWeb/oftalmologia/todosPonentes",
5.     //url: "http://localhost:8080/ServicioWeb/" +u+ "/todosPonentes",
6.     // mejor lo de abajo, así puede accederse desde otros equipos, no solo localhost.
7.     url: url + u + "/todosPonentes",
8.     dataType: "json",
9.     cache: false,
10.    success: function(data) {
11.      var html = "<table><tr><th>DNI</th><th>Nombre</th><th>Apellidos</th>";
12.      html += "<th>Afiliación</th><th>País</th></tr>";
13.      $.each(data, function(k, v) {
14.        html += "<tr>";
15.        html += "<td>" + v.dni + "</td>" + "<td>" + v.nombre + "</td>";
16.        html += "<td>" + v.apellidos + "</td>" + "<td>" + v.afiliacion + "</td>";
17.        html += "<td>" + v.pais + "</td>";
18.        html += "</td>";
19.      });
20.      html += "</table>";
21.      $("#consola").html(html);
22.    },
23.    error: function(res) {
24.      $("#consola").html("ERROR: " + res.statusText);
```

```
25.     }  
26.   });  
27. }
```

### 3.2.2. Función obtenerPonente()

Esta función también es de tipo GET pero ahora solo obtendremos un ponente indicando su dni a través de la URL. Obtendremos el dni a través de un identificador que identifica a un <input> de tipo 'text'. Lo primero que hacemos en esta función es recuperarlo para poder luego añadirlo a la url de la petición.

#### Código 3.2.2. Función obtenerPonente()

```
1. function obtenerPonente(u) {  
2.   var dni = $("#dni2").val();  
3.   $.ajax({  
4.     type: "GET",  
5.     //url: "http://localhost:8080/ServicioWeb/ofthalmologia/obtenerPonente/" + dni,  
6.     //url: "http://localhost:8080/ServicioWeb/" +u+ "/obtenerPonente/" + dni,  
7.     url: url + u + "/obtenerPonente/" + dni,  
8.     dataType: "text",  
9.     cache: false,  
10.    success: function(data) {  
11.      var html = "<pre>" + data + "</pre>";  
12.      $("#consola").html(html);  
13.    },  
14.    error: function(res) {  
15.      $("#consola").html("ERROR: " + res.statusText);  
16.    }  
17.  });  
18. }
```

### 3.2.3. Función modificarPonente()

Ahora vamos actualizar un ponente. Además de indicar que vamos a enviar una petición de tipo PUT, tenemos dos indicaciones nuevas:

- **contentType**: para indicar el tipo de datos que vamos a enviar al servidor. En nuestro caso al tratarse de datos en formato JSON tendremos que notificarlo añadiendo "application/json".
- **data**: estos son los datos que queremos enviar a nuestro servicio Web. Como queremos enviar datos JSON, hemos utilizado la función JSON.stringify que lo que hace es convertir un string a JSON.

#### Código 3.2.3. Función modificarPonente()

```
1. function modificarPonente(u) {
2.   var dni = $("#dni3").val();
3.   var nom = $("#nom3").val();
4.   var ape = $("#ape3").val();
5.   var afi = $("#afi3").val();
6.   var pais = $("#pais3").val();
7.   $.ajax({
8.     type: "PUT",
9.     //url: "http://localhost:8080/ServicioWeb/ofthalmologia/modificarPonente/" + dni,
10.    //url: "http://localhost:8080/ServicioWeb/" +u+ "/modificarPonente/" + dni,
11.    url: url + u + "/modificarPonente/" + dni,
12.    contentType: "application/json",
13.    dataType: "text",
14.    cache: false,
15.    data: JSON.stringify({
16.      "dni": dni, "nombre": nom, "apellidos": ape,
17.      "afiliacion": afi, "pais": pais
18.    }),
19.    success: function(data) {
20.      $("#consola").html(data);
21.    },
22.    error: function(res) {
23.      $("#consola").html("ERROR: " + res.statusText);
24.    }
25.  });
26. }
```



### 3.2.4. Función eliminarPonente()

Aquí lo novedoso es que indicamos que vamos a realizar un petición de tipo DELETE.

#### Código 3.2.4. Función eliminarPonente()

```
1. function eliminarPonente(u) {
2.   var dni = $("#dni4").val();
3.   $.ajax({
4.     type: "DELETE",
5.     //url: "http://localhost:8080/ServicioWeb/oftaImologia/eliminarPonente/" + dni,
6.     //url: "http://localhost:8080/ServicioWeb/" +u+ "/eliminarPonente/" + dni,
7.     url: url + u + "/eliminarPonente/" + dni,
8.     dataType: "text",
9.     cache: false,
10.    success: function(data) {
11.      $("#consola").html(data);
12.    },
13.    error:function(res) {
14.      $("#consola").html("ERROR: " + res.statusText);
15.    }
16.  });
17. }
```

### 3.2.5. Función nuevoPonente()

Pocas novedades presenta esta función, sólo que decimos que vamos a enviar una petición al servidor de tipo POST y volvemos a hacer uso de la función JSON.stringify para enviar los datos.

#### Código 3.2.5. Función nuevoPonente()

```
1. function nuevoPonente(u) {
2.   var dni = $("#dni5").val();
3.   var nom = $("#nom5").val();
4.   var ape = $("#ape5").val();
5.   var afi = $("#afi5").val();
6.   var pais = $("#pais5").val();
7.   $.ajax({
8.     type: "POST",
9.     //url: "http://localhost:8080/ServicioWeb/oftaImologia/nuevoPonente",
10.    //url: "http://localhost:8080/ServicioWeb/" +u+ "/nuevoPonente",
11.    url: url + u + "/nuevoPonente",
12.    contentType: "application/json",
13.    dataType: "text",
14.    cache: false,
15.    data: JSON.stringify({
16.      "dni": dni, "nombre": nom, "apellidos": ape,
17.      "afiliacion": afi, "pais": pais
18.    }),
19.    success: function(data) {
20.      $("#consola").html(data);
21.    }
22.  });
23. }
```

```
21. },
22. error:function(res) {
23.     $("#consola").html("ERROR: " + res.statusText);
24. }
25. });
26. }
```

### 3.2.6. Función nuevoPonenteForm()

Muy parecida a la anterior sólo que esta vez los datos nos llega mediante un formulario, para ello primero obtenemos dicho formulario mediante su identificador y luego los datos los obtenemos con `frm.serialize()`, siendo `frm` la variable que contiene el formulario.

#### Código 3.2.6. Función nuevoPonenteForm()

```
1. function nuevoPonenteForm() {
2.     var frm = $('#idForm');
3.     $.ajax({
4.         dataType: "text",
5.         type: frm.attr('method'),
6.         url: frm.attr('action'),
7.         data: frm.serialize(),
8.         cache: false,
9.         success: function(data) {
10.            $("#consolaForm").html(data);
11.        },
12.        error:function(res) {
13.            $("#consolaFormError").html("ERROR: " + res.statusText);
14.        }
15.    });
16. }
```

## 3.3. Probamos las funciones en nuestro proyecto

Adjuntamos muestras del proyecto web desde el navegador con capturas de pantalla, en nuestro caso ya tenemos añadidas los debidos ficheros css para mejorar el diseño anteriormente expuesto:

## Servicio web REST - Cliente jQuery para servicios Restful



Figura 14. index.html

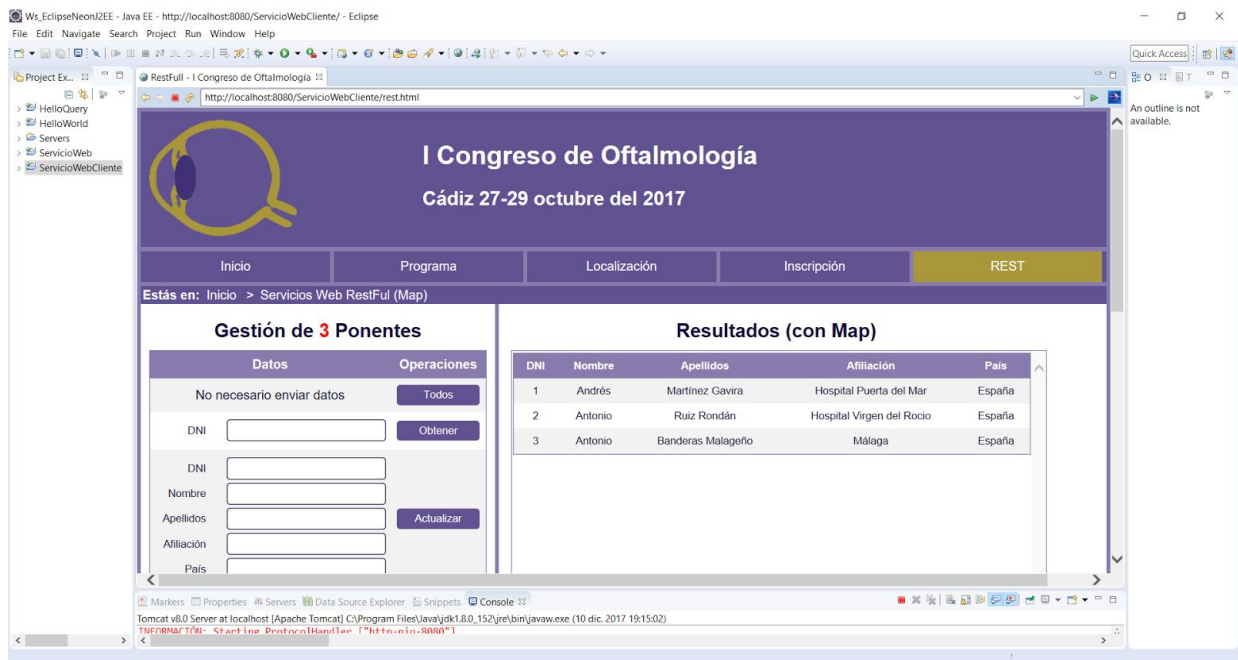


Figura 15. Resultado de listar todos los Ponentes en versión con Map. rest.html

## Servicio web REST - Cliente jQuery para servicios Restful

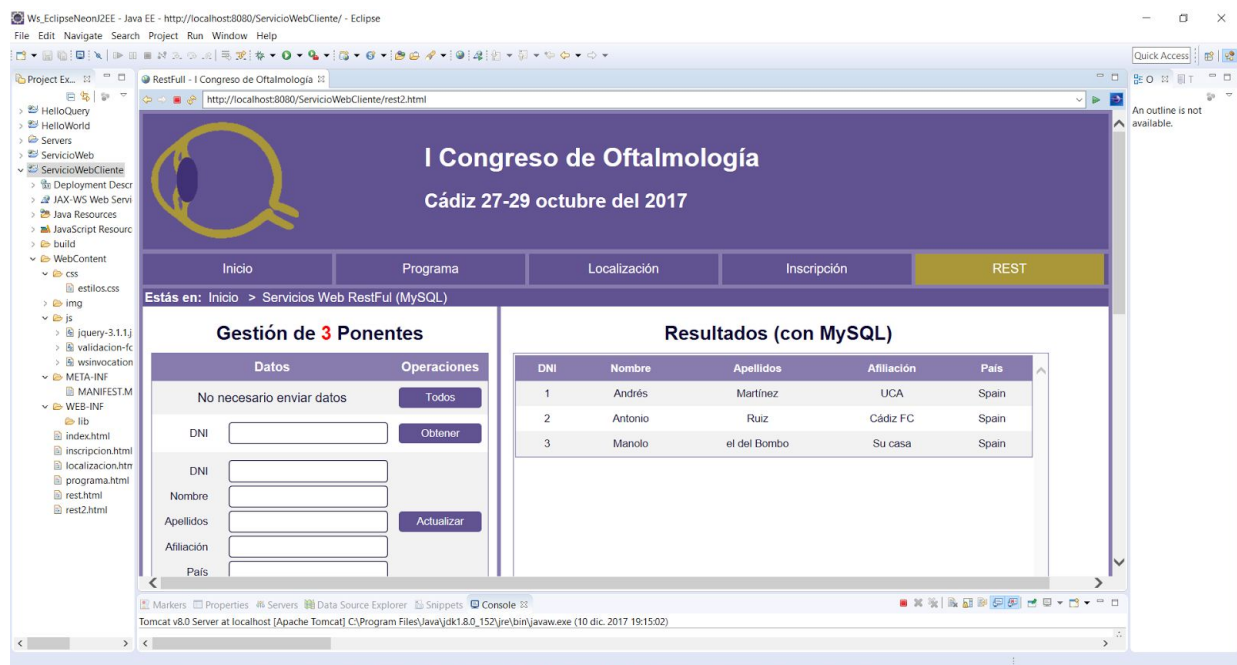


Figura 16. Resultado de listar todos los Ponentes en versión con Map. rest2.html

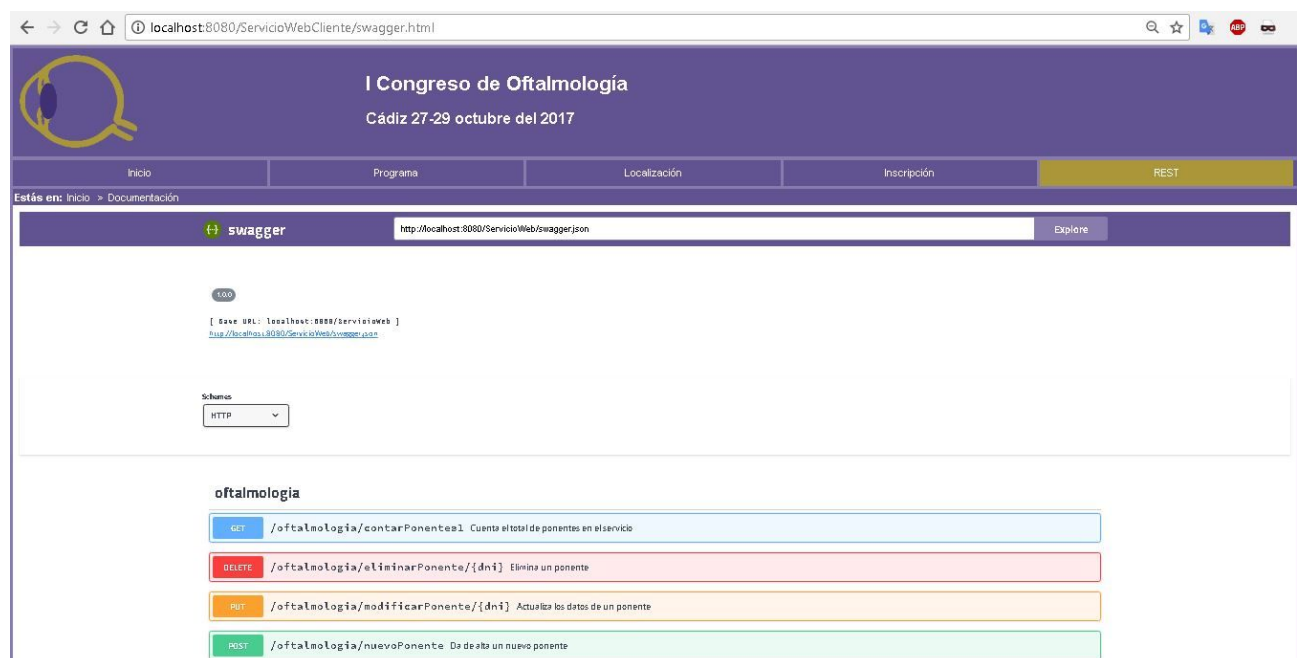


Figura 17. Servicios documentados con Swagger integrado en el proyecto

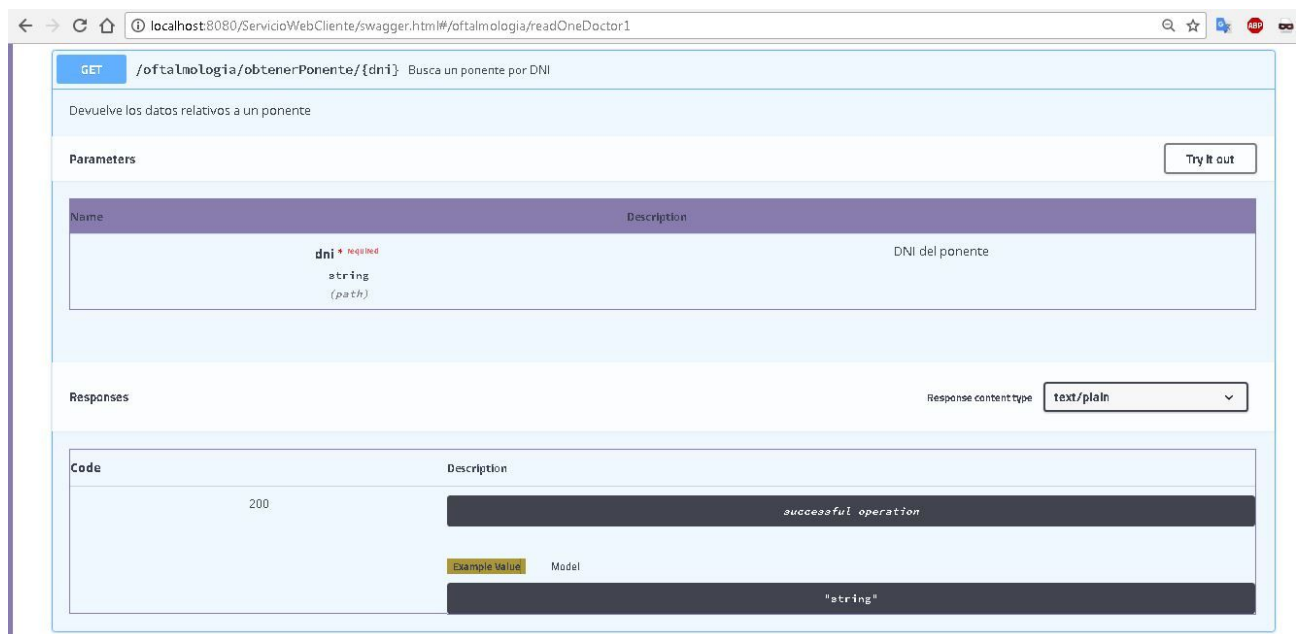


Figura 18. Detalle de documentación de una de las operaciones

**OPERACIÓN opcional 1: Añadir un nuevo ponente pasándolo como parámetro de un formulario**

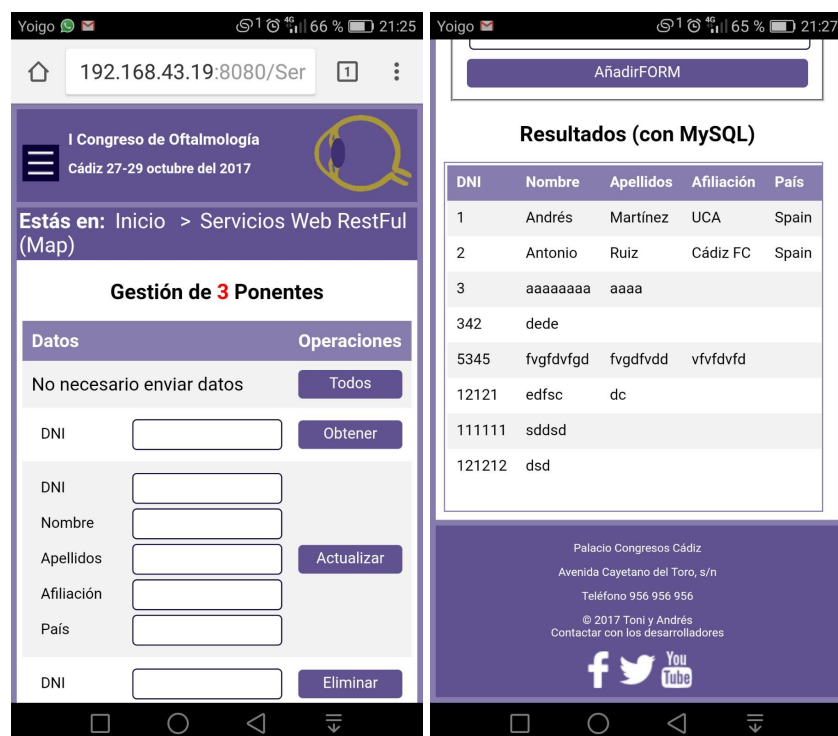
Nuevo Ponente

DNI	<input type="text"/>
Nombre	<input type="text"/>
Apellidos	<input type="text"/>
Afiliación	<input type="text"/>
País	<input type="text"/>
<input type="button" value="AñadirFORM"/>	

Figura 19. Formulario



Figura 20. Operación obtenerPonente(dni), en versión con MySQL



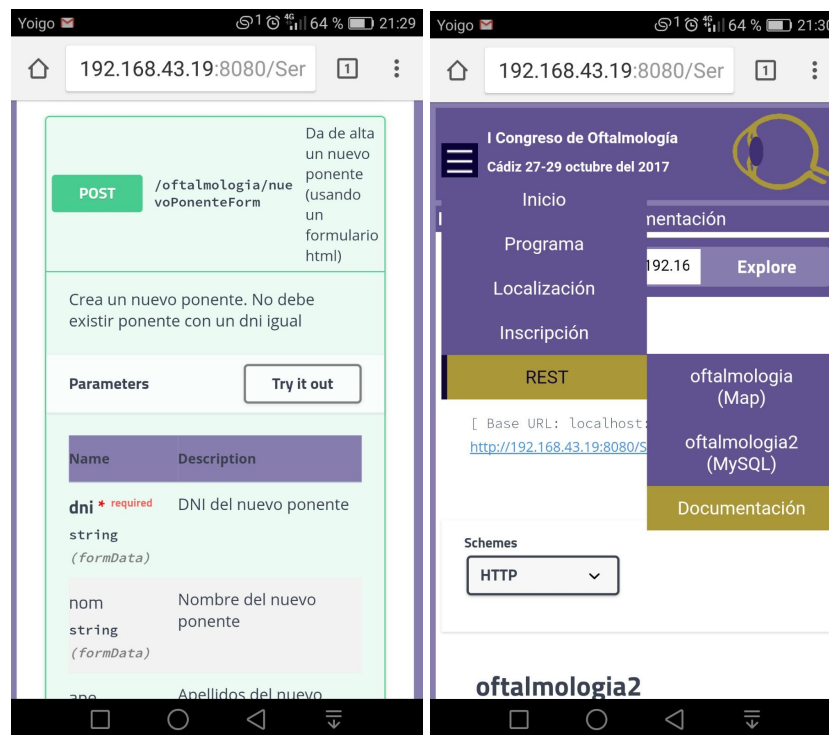


Figura 21. Desde el navegador web de un movil.

## Bibliografía

- [1] **Apache Tomcat - Apache Tomcat 8 Software Downloads**  
<https://tomcat.apache.org/download-80.cgi>  
Fecha de último acceso: 15 de diciembre de 2017
- [2] **Eclipse IDE for Java EE Developers**  
<https://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/neon3>  
Fecha de último acceso: 15 de diciembre de 2017
- [3] **MySQL and Java JDBC - Tutorial (Lars Vogel)**  
<http://www.vogella.com/tutorials/MySQLJava/article.html#jdbcdriver>  
Fecha de último acceso: 20 de diciembre de 2017
- [4] **REST with Java (JAX-RS) using Jersey**  
<http://www.vogella.com/tutorials/REST/article.html>  
Fecha de último acceso: 20 de diciembre de 2017
- [5] **jQuery.ajax() | jQuery API Documentation**  
<http://api.jquery.com/jquery.ajax/>



Fecha de último acceso: 20 de diciembre de 2017

- [6] **Conectar MySQL con Java - Línea de Código**

<http://lineadecodigo.com/java/conectar-mysql-java/>

Fecha de último acceso: 20 de diciembre de 2017

- [7] **Problema con SSL al conectar a MySQL - Stack Overflow**

<https://es.stackoverflow.com/questions/23307/problema-con-ssl-al-conectar-a-mysql>

Fecha de último acceso: 20 de diciembre de 2017

- [8] **Servicios REST documentados y probados con Swagger | adictosaltrabajo**

<https://www.adictosaltrabajo.com/tutoriales/rest-swagger/>

Fecha de último acceso: 20 de diciembre de 2017

- [9] **Documentar un servicio REST, con Swagger | Another day, another bug...**

<https://anotherdayanotherbug.wordpress.com/2014/09/27/documentar-un-servicio-rest-con-swagger/>

Fecha de último acceso: 20 de diciembre de 2017

- [10] **MySQL :: Download Connector/J**

<https://dev.mysql.com/downloads/connector/j/>

Fecha de último acceso: 20 de diciembre de 2017

- [11] **Java Database Connectivity - Wikipedia**

[https://en.wikipedia.org/wiki/Java\\_Database\\_Connectivity](https://en.wikipedia.org/wiki/Java_Database_Connectivity)

Fecha de último acceso: 20 de diciembre de 2017

## Anexo A: Clase Ponente

```
1. package pnetSW;
2.
3. import javax.xml.bind.annotation.XmlRootElement;
4.
5. @XmlRootElement
6. public class Ponente {
7.     private String dni, nombre, apellidos, afiliacion, pais;
8.
9.     public Ponente() {}
10.
11.     public Ponente(String dni, String nombre, String apellidos,
12.                     String afiliacion, String pais) {
13.         this.dni = dni;
14.         this.nombre = nombre;
15.         this.apellidos = apellidos;
16.         this.afiliacion = afiliacion;
17.         this.pais = pais;
18.     }
19.
20.     public String getDni() {
21.         return dni;
22.     }
23.
24.     public void setDni(String dni) {
25.         this.dni = dni;
26.     }
27.
28.     public String getNombre() {
29.         return nombre;
30.     }
31.
```

```
32. public void setNombre(String nombre) {
33.     this.nombre = nombre;
34. }
35.
36. public String getPais() {
37.     return pais;
38. }
39.
40. public void setPais(String pais) {
41.     this.pais = pais;
42. }
43.
44. public String getApellidos() {
45.     return apellidos;
46. }
47.
48. public void setApellidos(String apellidos) {
49.     this.apellidos = apellidos;
50. }
51.
52. public String getAfiliacion() {
53.     return afiliacion;
54. }
55.
56. public void setAfiliacion(String afiliacion) {
57.     this.afiliacion = afiliacion;
58. }
59.
60. public String toString() {
61.     String msg;
62.
63.     msg = "DNI: " + this.dni;
64.     msg += "\nNOMBRE: " + this.nombre;
65.     msg += "\nAPELLIDOS: " + this.apellidos;
66.     msg += "\nAFILIACIÓN: " + this.afiliacion;
67.     msg += "\nPAÍS: " + this.pais;
68.
69.     return msg;
70. }
71. }
```

## Anexo B: Clase Servicio (con Map)

```
1. package pnetSW;
2.
3. import java.util.HashMap;
4. import java.util.Map;
5.
6. import javax.ws.rs.Consumes;
7. import javax.ws.rs.DELETE;
8. import javax.ws.rs.FormParam;
9. import javax.ws.rs.GET;
10. import javax.ws.rs.POST;
11. import javax.ws.rs.PUT;
12. import javax.ws.rs.Path;
13. import javax.ws.rs.PathParam;
14. import javax.ws.rs.Produces;
15. import javax.ws.rs.core.MediaType;
16.
17.
18. /**
19.  * @author      Andrés Martínez <andres.martinezgavira@alum.uca.es>
20.  * @author      Antonio Ruiz <antonio.ruizrondan@alum.uca.es>
21.  * @version     1.0
22.  * @since      1.0
23.  */
24. @Path("/oftalmologia")
25. public class Servicio {
26.
27.     /**
28.      * Objeto estático del servicio donde almacenar la colección de
29.      * ponentes del congreso para su consulta, actualización, etc.
30.      *
31.      * Este objeto es un Map, cuya clave será el DNI del ponente y el
32.      * valor es el Ponente que tiene ese dni.
33.      */
34.     private static Map<String, Ponente> bbdd = new HashMap<>();
35.
36.     /**
37.      * Poblamos la colección inicialmente con un par de ponentes
```

```

38.  * oftalmólogos.
39.  */
40.  static {
41.      Ponente p1 = new Ponente("1", "Andrés", "Martínez Gavira",
42.          "Hospital Puerta del Mar", "España");
43.      Ponente p2 = new Ponente("2", "Antonio", "Ruiz Rondán",
44.          "Hospital Virgen del Rocío", "España");
45.      Ponente p3 = new Ponente("3", "Antonio", "Banderas Malageño",
46.          "Málaga", "España");
47.      bbdd.put(p1.getDni(), p1);
48.      bbdd.put(p2.getDni(), p2);
49.      bbdd.put(p3.getDni(), p3);
50.  }
51.
52.  /**
53.   * OPERACIÓN 1: Obtener en JSON todos los miembros del comité de
54.   * programa (DNI, nombre, apellidos, afiliación, nacionalidad. Ej:
55.   * 1, Andrés, Martínez Gavira, Hospital Puerta del Mar, España)
56.   *
57.   * @return Mapa (diccionario), cuya clave será el DNI del ponente y
58.   *         el valor es el Ponente que tiene ese dni.
59.   */
60.  @GET
61.  @Path("/todosPonentes")
62.  @Produces({"application/json"})
63.  public Map<String, Ponente> readAllDoctors() {
64.      return bbdd;
65.  }
66.
67.  /**
68.   * OPERACIÓN 2: Obtener en modo texto los datos de un miembro
69.   * concreto del comité de programa. El miembro a buscar se podrá
70.   * especificar con su nombre (DNI) como parámetro del path.
71.   *
72.   * @param key Entero que representa el dni del ponente a obtener
73.   * @return Mensaje de error o éxito al obtener a un ponente
74.   */
75.  @GET
76.  @Path("/obtenerPonente/{dni}")
77.  @Produces(MediaType.TEXT_PLAIN)
78.  public String readOneDoctor(@PathParam("dni") String key) {
79.      String msg;
80.
81.      try {
82.          if(bbdd.containsKey(key)) {
83.              msg = bbdd.get(key).toString();
84.          } else
85.              msg = "ERROR: No existe un ponente con DNI: " + key;
86.      } catch (Exception e) {
87.          msg = "ERROR (Excepción): " + e.getMessage();
88.      }
89.
90.      return msg;
91.  }
92.
93.  /**
94.   * OPERACIÓN 3: Actualizar la afiliación y nacionalidad de afiliación
95.   * de un miembro (pasando como parámetro del path el nombre (DNI) y
96.   * los nuevos datos como JSON)
97.   *

```

```

98.      * @param key   Dni del ponente a actualizar
99.      * @param p     Ponente serializado desde JSON a Ponente gracias a
100.     *             la notación XmlRootElement sobre la clase Ponente
101.     * @return      Mensaje de error o éxito al actualizar a un ponente
102.     */
103.     @PUT
104.     @Path("/modificarPonente/{dni}")
105.     @Consumes(MediaType.APPLICATION_JSON)
106.     @Produces(MediaType.TEXT_PLAIN)
107.     public String updateDoctor(@PathParam("dni") String key, Ponente p) {
108.         String msg;
109.
110.         try {
111.             if(bbdd.containsKey(key)) {
112.                 Ponente actP = bbdd.get(key);
113.                 if(p.getNombre() != null && !p.getNombre().isEmpty())
114.                     actP.setNombre(p.getNombre());
115.                 if(p.getApellidos() != null && !p.getApellidos().isEmpty())
116.                     actP.setApellidos(p.getApellidos());
117.                 if(p.getAfiliacion() != null && !p.getAfiliacion().isEmpty())
118.                     actP.setAfiliacion(p.getAfiliacion());
119.                 if(p.getPais() != null && !p.getPais().isEmpty())
120.                     actP.setPais(p.getPais());
121.                 bbdd.put(key, actP);
122.                 msg = "Ponente actualizado correctamente";
123.             } else
124.                 msg = "ERROR: No existe un ponente con DNI: " + key;
125.         } catch (Exception e) {
126.             msg = "ERROR (Excepción): " + e.getMessage();
127.         }
128.
129.         return msg;
130.     }
131.
132.     /**
133.     * OPERACIÓN 4: Borrar un miembro del comité de programa (pasando su
134.     * nombre (DNI) como parámetro del path)
135.     *
136.     * @param key   Entero que representa el dni del ponente a eliminar
137.     * @return      Mensaje de error o éxito al eliminar a un ponente
138.     */
139.     @DELETE
140.     @Path("/eliminarPonente/{dni}")
141.     @Produces(MediaType.TEXT_PLAIN)
142.     public String deleteDoctor(@PathParam("dni") String key){
143.         String msg;
144.
145.         try {
146.             if(bbdd.containsKey(key)) {
147.                 bbdd.remove(key);
148.                 msg = "Eliminado correctamente ponente con DNI: " + key;
149.             } else
150.                 msg = "ERROR: No existe un ponente con DNI: " + key;
151.         } catch (Exception e) {
152.             msg = "ERROR (Excepción): " + e.getMessage();
153.         }
154.
155.         return msg;
156.     }
157.

```

```

158.  /**
159.   * OPERACIÓN 5: Añadir un nuevo miembro del comité de programa
160.   * pasándolo como JSON
161.   *
162.   * @param p Ponente serializado desde JSON a Ponente gracias a la
163.   * notación XmlRootElement indicada sobre la clase Ponente
164.   * @return Mensaje de error o éxito al añadir al nuevo ponente
165.   */
166.  @POST
167.  @Path("/nuevoPonente")
168.  @Consumes(MediaType.APPLICATION_JSON)
169.  @Produces(MediaType.TEXT_PLAIN)
170.  public String createDoctor(Ponente p){
171.      String msg;
172.
173.      try {
174.          if(p.getDni() == null || p.getDni().isEmpty()) {
175.              msg = "Tienes que indicar un DNI";
176.          } else if(!p.getDni().matches("\\d+")) {
177.              msg = "Tienes que introducir un número para el DNI";
178.          } else if(!bbdd.containsKey(p.getDni())) {
179.              bbdd.put(p.getDni(), p);
180.              msg = "Ponente añadido correctamente";
181.          } else
182.              msg = "ERROR: Existe un ponente con DNI: " + p.getDni();
183.      } catch (Exception e) {
184.          msg = "ERROR (Excepción): " + e.getMessage();
185.      }
186.
187.      return msg;
188.  }
189.
190.  /**
191.   * OPERACIÓN opcional 1
192.   * Añadir un nuevo miembro del comité de programa pasándolo como
193.   * parámetro de formulario
194.   *
195.   * @param dni DNI/Pasaporte del nuevo Ponente
196.   * @param nom Nombre del nuevo Ponente
197.   * @param ape Apellidos del nuevo Ponente
198.   * @param afi Afiliación del nuevo Ponente
199.   * @param pais País del nuevo Ponente
200.   * @return Mensaje de error o éxito al añadir al nuevo ponente
201.   */
202.  @POST
203.  @Path("/nuevoPonenteForm")
204.  @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
205.  @Produces(MediaType.TEXT_PLAIN)
206.  public String createDoctorForm(@FormParam("dni") String dni,
207.  @FormParam("nom") String nom, @FormParam("ape") String ape,
208.  @FormParam("afi") String afi, @FormParam("pais") String pais) {
209.      String msg;
210.
211.      try {
212.          if(dni == null || dni.isEmpty()) {
213.              msg = "Tienes que indicar un DNI";
214.          } else if(!dni.matches("\\d+")) {
215.              msg = "Tienes que introducir un número para el DNI";
216.          } else if(!bbdd.containsKey(dni)) {
217.              Ponente p = new Ponente(dni, nom, ape, afi, pais);

```

```
218.         bbdd.put(dni, p);
219.         msg = "Ponente añadido correctamente";
220.     } else
221.         msg = "ERROR: Ya existe un ponente con DNI: " + dni;
222.     } catch (Exception e) {
223.         msg = "ERROR (Excepción): " + e.getMessage();
224.     }
225.
226.     return msg;
227. }
228.
229. /**
230.  * OPERACIÓN opcional 3
231.  * Devuelve el nº de registros de la estructura Map.
232.  *
233.  * @return nº de ponentes
234.  */
235. @GET
236. @Path("/contarPonentes1")
237. @Produces(MediaType.TEXT_PLAIN)
238. public int countDoctors() {
239.     return bbdd.size();
240. }
241. }
```



## Anexo C: Clase Servicio2 (con MySQL)

```
1. package pnetSW;
2.
3. import java.sql.Connection;
4. import java.sql.DriverManager;
5. import java.sql.PreparedStatement;
6. import java.sql.ResultSet;
7. import java.sql.SQLException;
8. import java.sql.Statement;
9. import java.util.HashMap;
10. import java.util.Map;
11. import java.util.concurrent.atomic.AtomicInteger;
12.
13. import javax.ws.rs.Consumes;
14. import javax.ws.rs.DELETE;
15. import javax.ws.rs.FormParam;
16. import javax.ws.rs.GET;
17. import javax.ws.rs.POST;
18. import javax.ws.rs.PUT;
19. import javax.ws.rs.Path;
20. import javax.ws.rs.PathParam;
21. import javax.ws.rs.Produces;
22. import javax.ws.rs.core.MediaType;
23.
24.
25. /**
26.  * @author      Andrés Martínez <andres.martinezgavira@alum.uca.es>
27.  * @author      Antonio Ruiz <antonio.ruizrondan@alum.uca.es>
28.  * @version     1.0
29.  * @since      1.0
30.  */
31. @Path("/oftalmologia2")
32. public class Servicio2 {
33.     private Connection conexion = null;
34.     private Statement comando = null;
35.     private PreparedStatement ps = null;
36.     private ResultSet registro = null;
37.     private String consulta;
38.     private static AtomicInteger totalPonentes = new AtomicInteger(total());
39.
40.     /**
41.      * Establece conexión a SGBD MySQL, con las siguientes características.
42.      *
43.      * Nombre BBDD:      'pnet'
```

```

44.  * Usuario:          'pnet'
45.  * Contraseña:       'pnet'
46.  * IP Servidor MySQL: '127.0.0.1'
47.  * Puerto MySQL:     '3306'
48.  *
49.  * @return objeto Connection
50.  */
51.  private static Connection MySQLConnect() {
52.      Connection con = null;
53.      try {
54.          Class.forName("com.mysql.jdbc.Driver");
55.          String user = "pnet";
56.          String pass = "pnet";
57.          String name = "pnet";
58.          String host = "127.0.0.1";
59.          int port = 3306;
60.          String url = String.format("jdbc:mysql://%s:%d/%s?useSSL=false", host, port, name);
61.          con = DriverManager.getConnection(url, user, pass);
62.
63.          if (con != null) {
64.              System.out.println("Conexión a base de datos " + url + " ... Ok");
65.          }
66.      } catch (Exception ex) {
67.          System.err.println(ex);
68.      }
69.
70.      return con;
71.  }
72.
73.
74.  /**
75.   * OPERACIÓN 1: Obtener en JSON todos los miembros del comité de
76.   * programa (DNI, nombre, apellidos, afiliación, nacionalidad. Ej:
77.   * 1, Andrés, Martínez Gavira, Hospital Puerta del Mar, España)
78.   *
79.   * @return Mapa (diccionario), cuya clave será el DNI del ponente y
80.   * el valor es el Ponente que tiene ese dni.
81.   */
82.  @GET
83.  @Path("/todosPonentes")
84.  @Produces({"application/json"})
85.  public Map<String, Ponente> readAllDoctors() {
86.      Map<String, Ponente> bbdd = new HashMap<>();
87.
88.      try {
89.          conexion = MySQLConnect();
90.          consulta = "SELECT * FROM ponentes";
91.          comando = conexion.createStatement();
92.          registro = comando.executeQuery(consulta);
93.          System.out.println(consulta);
94.
95.          while(registro.next()) {
96.              bbdd.put(registro.getString(1),
97.                  new Ponente(registro.getString(1), registro.getString(2),
98.                      registro.getString(3), registro.getString(4),
99.                      registro.getString(5)));
100.          }
101.      } catch (Exception ex) {
102.          System.err.println(ex);
103.      } finally {

```

```

104.         cerrarTodo();
105.     }
106.
107.     return bbdd;
108. }
109.
110.
111. /**
112.  * OPERACIÓN 2: Obtener en modo texto los datos de un miembro
113.  * concreto del comité de programa. El miembro a buscar se podrá
114.  * especificar con su nombre (DNI) como parámetro del path.
115.  *
116.  * @param key Entero que representa el dni del ponente a obtener
117.  * @return Mensaje de error o éxito al obtener a un ponente
118.  */
119. @GET
120. @Path("/obtenerPonente/{dni}")
121. @Produces(MediaType.TEXT_PLAIN)
122. public String readOneDoctor(@PathParam("dni") String key) {
123.     String msg;
124.     try {
125.         conexion = MySQLConnect();
126.         consulta = "SELECT * FROM ponentes WHERE dni = ?";
127.         ps = conexion.prepareStatement(consulta);
128.         ps.setString(1, key);
129.         registro = ps.executeQuery();
130.         System.out.println(ps.toString());
131.         registro.next();
132.         msg = "DNI: " + registro.getString(1);
133.         msg += "\nNOMBRE: " + registro.getString(2);
134.         msg += "\nAPELLIDOS: " + registro.getString(3);
135.         msg += "\nAFILIACIÓN: " + registro.getString(4);
136.         msg += "\nPAÍS: " + registro.getString(5);
137.     } catch (SQLException ex) {
138.         msg = "ERROR: No existe un ponente con DNI: " + key;
139.     } catch (Exception ex) {
140.         msg = "ERROR (Excepción): " + ex.getMessage();
141.     } finally {
142.         cerrarTodo();
143.     }
144.
145.     return msg;
146. }
147.
148.
149. /**
150.  * OPERACIÓN 3: Actualizar la afiliación y nacionalidad de afiliación
151.  * de un miembro (pasando como parámetro del path el nombre (DNI) y
152.  * los nuevos datos como JSON)
153.  *
154.  * @param key Dni del ponente a actualizar
155.  * @param p Ponente serializado desde JSON a Ponente gracias a
156.  * la notación XmlRootElement sobre la clase Ponente
157.  * @return Mensaje de error o éxito al actualizar a un ponente
158.  */
159. @PUT
160. @Path("/modificarPonente/{dni}")
161. @Consumes(MediaType.APPLICATION_JSON)
162. @Produces(MediaType.TEXT_PLAIN)
163. public String updateDoctor(@PathParam("dni") String key, Ponente p) {

```

```

164.     String msg;
165.
166.     try {
167.         conexion = MySQLConnect();
168.         consulta = "SELECT COUNT(*) FROM ponentes WHERE dni = ?";
169.         ps = conexion.prepareStatement(consulta);
170.         ps.setString(1, key);
171.         registro = ps.executeQuery();
172.         registro.next();
173.         if(registro.getInt(1) > 0) {
174.             consulta = "UPDATE ponentes SET nombre = ?, apellidos = ?";
175.             consulta += ", afiliacion = ?, pais = ? WHERE dni = ?";
176.             ps = conexion.prepareStatement(consulta);
177.             ps.setString(1, p.getNombre());
178.             ps.setString(2, p.getApellidos());
179.             ps.setString(3, p.getAfiliacion());
180.             ps.setString(4, p.getPais());
181.             ps.setString(5, key);
182.             ps.executeUpdate();
183.             System.out.println(ps.toString());
184.             msg = "Ponente actualizado correctamente";
185.         } else
186.             msg = "ERROR: No existe un ponente con DNI: " + key;
187.         } catch (Exception e) {
188.             msg = "ERROR (Excepción): " + e.getMessage();
189.         } finally {
190.             cerrarTodo();
191.         }
192.
193.     return msg;
194. }
195.
196.
197. /**
198.  * OPERACIÓN 4: Borrar un miembro del comité de programa (pasando su
199.  * nombre (DNI) como parámetro del path)
200.  *
201.  * @param key Entero que representa el dni del ponente a eliminar
202.  * @return Mensaje de error o éxito al eliminar a un ponente
203.  */
204. @DELETE
205. @Path("/eliminarPonente/{dni}")
206. @Produces(MediaType.TEXT_PLAIN)
207. public String deleteDoctor(@PathParam("dni") String key){
208.     String msg;
209.
210.     try {
211.         conexion = MySQLConnect();
212.         consulta = "SELECT COUNT(*) FROM ponentes WHERE dni = ?";
213.         ps = conexion.prepareStatement(consulta);
214.         ps.setString(1, key);
215.         registro = ps.executeQuery();
216.         registro.next();
217.         if(registro.getInt(1) > 0) {
218.             consulta = "DELETE FROM ponentes WHERE dni = ?";
219.             ps = conexion.prepareStatement(consulta);
220.             ps.setString(1, key);
221.             ps.executeUpdate();
222.             totalPonentes.decrementAndGet();
223.             msg = "Eliminado correctamente ponente con DNI: " + key;

```

```

224.         System.out.println(ps.toString());
225.     } else
226.     {
227.         msg = "ERROR: No existe un ponente con DNI: " + key;
228.     } catch (Exception e) {
229.         msg = "ERROR (Excepción): " + e.getMessage();
230.     } finally {
231.         cerrarTodo();
232.     }
233.
234.     return msg;
235. }
236.
237. /**
238.  * OPERACIÓN 5: Añadir un nuevo miembro del comité de programa
239.  * pasándolo como JSON
240.  *
241.  * @param p Ponente serializado desde JSON a Ponente gracias a la
242.  *          notación XmlRootElement indicada sobre la clase Ponente
243.  * @return Mensaje de error o éxito al añadir al nuevo ponente
244.  */
245. @POST
246. @Path("/nuevoPonente")
247. @Consumes(MediaType.APPLICATION_JSON)
248. @Produces(MediaType.TEXT_PLAIN)
249. public String createDoctor(Ponente p){
250.     String msg;
251.
252.     try {
253.         conexion = MySQLConnect();
254.         consulta = "SELECT COUNT(*) FROM ponentes WHERE dni = ?";
255.         ps = conexion.prepareStatement(consulta);
256.         ps.setString(1, p.getDni());
257.         registro = ps.executeQuery();
258.         registro.next();
259.         if(p.getDni() == null || p.getDni().isEmpty()) {
260.             msg = "Tienes que indicar un DNI";
261.         } else if(!p.getDni().matches("\\d+")) {
262.             msg = "Tienes que introducir un número para el DNI";
263.         } else if(registro.getInt(1) == 0) {
264.             consulta = "INSERT INTO ponentes (dni, nombre, apellidos, afiliacion, pais)";
265.             consulta += "VALUES (?, ?, ?, ?, ?)";
266.             ps = conexion.prepareStatement(consulta);
267.             ps.setString(1, p.getDni());
268.             ps.setString(2, p.getNombre());
269.             ps.setString(3, p.getApellidos());
270.             ps.setString(4, p.getAfiliacion());
271.             ps.setString(5, p.getPais());
272.             ps.executeUpdate();
273.             totalPonentes.incrementAndGet();
274.             System.out.println(ps.toString());
275.             msg = "Ponente añadido correctamente";
276.         } else
277.         {
278.             msg = "ERROR: Existe un ponente con DNI: " + p.getDni();
279.         } catch (Exception e) {
280.             msg = "ERROR (Excepción): " + e.getMessage();
281.         } finally {
282.             cerrarTodo();
283.         }

```

```

284.         return msg;
285.     }
286.
287.
288.     /**
289.     * OPERACIÓN opcional 1
290.     * Añadir un nuevo miembro del comité de programa pasándolo como
291.     * parámetro de formulario
292.     *
293.     * @param dni    DNI/Pasaporte del nuevo Ponente
294.     * @param nom    Nombre del nuevo Ponente
295.     * @param ape    Apellidos del nuevo Ponente
296.     * @param afi    Afiliación del nuevo Ponente
297.     * @param pais   País del nuevo Ponente
298.     * @return       Mensaje de error o éxito al añadir al nuevo ponente
299.     */
300.     @POST
301.     @Path("/nuevoPonenteForm")
302.     @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
303.     @Produces(MediaType.TEXT_PLAIN)
304.     public String createDoctorForm(@FormParam("dni") String dni,
305.                                     @FormParam("nom") String nom, @FormParam("ape") String ape,
306.                                     @FormParam("afi") String afi, @FormParam("pais") String pais) {
307.
308.         String msg;
309.
310.         try {
311.             conexion = MySQLConnect();
312.             consulta = "SELECT COUNT(*) FROM ponentes WHERE dni = ?";
313.             ps = conexion.prepareStatement(consulta);
314.             ps.setString(1, dni);
315.             registro = ps.executeQuery();
316.             registro.next();
317.             if(dni == null || dni.isEmpty()) {
318.                 msg = "Tienes que indicar un DNI";
319.             } else if(!dni.matches("\\d+")) {
320.                 msg = "Tienes que introducir un número para el DNI";
321.             } else if(registro.getInt(1) == 0) {
322.                 consulta = "INSERT INTO ponentes (dni, nombre, apellidos, afiliacion, pais) ";
323.                 consulta += "VALUES (?, ?, ?, ?, ?)";
324.                 ps = conexion.prepareStatement(consulta);
325.                 ps.setString(1, dni);
326.                 ps.setString(2, nom);
327.                 ps.setString(3, ape);
328.                 ps.setString(4, afi);
329.                 ps.setString(5, pais);
330.                 ps.executeUpdate();
331.                 totalPonentes.incrementAndGet();
332.                 System.out.println(ps.toString());
333.                 msg = "Ponente añadido correctamente";
334.             } else
335.                 msg = "ERROR: Ya existe un ponente con DNI: " + dni;
336.         } catch (Exception e) {
337.             msg = "ERROR (Excepción): " + e.getMessage();
338.         } finally {
339.             cerrarTodo();
340.         }
341.
342.         return msg;
343.     }

```

```
344.
345.
346.    /**
347.     * OPERACIÓN opcional 3
348.     * Devuelve el nº total de ponentes.
349.     *
350.     * @return nº de ponentes
351.     */
352.    @GET
353.    @Path("/contarPonentes2")
354.    @Produces(MediaType.TEXT_PLAIN)
355.    public String countDoctors() {
356.        return totalPonentes.toString();
357.    }
358.
359.
360.    /**
361.     * Devuelve el total de registros contenidos en la tabla ponentes.
362.     *
363.     * @return nº filas de la tabla ponentes. -1 si error.
364.     */
365.    private static int total() {
366.        Connection con = null;
367.        Statement sta = null;
368.        ResultSet res = null;
369.        int t = -1;
370.
371.        try {
372.            con = MySQLConnect();
373.            String consulta = "SELECT COUNT(*) FROM ponentes";
374.            sta = con.createStatement();
375.            res = sta.executeQuery(consulta);
376.            res.next();
377.            t = res.getInt(1);
378.        } catch (Exception e) {
379.            System.err.println(e.getMessage());
380.        } finally {
381.            try {
382.                if (res != null) {
383.                    res.close();
384.                }
385.
386.                if (sta != null) {
387.                    sta.close();
388.                }
389.
390.                if (con != null) {
391.                    con.close();
392.                }
393.            } catch (Exception ex) {
394.                System.err.println(ex.getMessage());
395.            }
396.        }
397.
398.        return t;
399.    }
400.
401.
402.    /**
403.     * Cierra resultset, conexión, etc...
```

```
404.      */
405.      private void cerrarTodo() {
406.          try {
407.              if (registro != null) {
408.                  registro.close();
409.              }
410.
411.              if (ps != null) {
412.                  ps.close();
413.              }
414.
415.              if (comando != null) {
416.                  comando.close();
417.              }
418.
419.              if (conexion != null) {
420.                  conexion.close();
421.              }
422.          } catch (Exception ex) {
423.              System.err.println(ex.getMessage());
424.          }
425.      }
426.
427.  }
```



## Anexo D: Funciones jQuery de invocación de servicios

```
1.  /*
2.  * @fileoverview Invocaciones a Los servicios web REST implementados en
3.  *               Servicio.java y Servicio2.java
4.  *
5.  * @version     1.1
6.  *
7.  * @author      Andrés Martínez <andres.martinezgavira@alum.uca.es>
8.  * @author      Antonio Ruiz <antonio.ruizrondan@alum.uca.es>
9.  * @copyright   Antonio&Andrés
10. *
11. * History
12. *
13. * v1.1 - Se mejoró pudiendo consultar a diferentes servicios que implementan
14. *       Las mismas operaciones.
15. *
16. * La primera versión se hizo solamente para Servicio.java, el cual mantenía
17. * una estructura estática en memoria de Los datos (con un Map).
18. */
19.
20.
21. //Variables globales para construir la url
22. var protocolo = window.location.protocol + "://";
23. var host = window.location.host; //en producción indicar IP pública o dominio
24. var servicio = "/ServicioWeb/";
25. var url = protocolo + host + servicio;
26.
27.
28. /**
29. * Solicita a Los servicios web implementados que Le envíe todos Los ponentes que
30. * actualmente almacenan. Estos datos Los recibe del servicio en formato json.
31. *
32. * @param {String} u Representa el Servicio al que queremos invocar, es decir
33. *                   "oftalmologia": Servicio.java;
34. *                   "oftalmologia2": Servicio2.java
35. * @returns {String} Devuelve una cadena de texto en formato html conteniendo dichos
36. *                   datos
37. */
38. function todosPonentes(u) {
39. $.ajax({
40.   type: "GET",
41.   //url: "http://localhost:8080/ServicioWeb/oftalmologia/todosPonentes",
42.   //url: "http://localhost:8080/ServicioWeb/" +u+ "/todosPonentes",
43.   // mejor lo de abajo, así puede accederse desde otros equipos, no solo localhost.
44.   url: url + u + "/todosPonentes",
45.   dataType: "json",
46.   cache: false,
47.   success: function(data) {
48.     var html = "<table><tr><th>DNI</th><th>Nombre</th><th>Apellidos</th>";
```

```

49.     html += "<th>Afiliación</th><th>País</th></tr>";
50.     $.each(data, function(k, v) {
51.         html += "<tr>";
52.         html += "<td>" + v.dni + "</td>" + "<td>" + v.nombre + "</td>";
53.         html += "<td>" + v.apellidos + "</td>" + "<td>" + v.afiliacion + "</td>";
54.         html += "<td>" + v.pais + "</td>";
55.         html += "</td>";
56.     });
57.     html += "</table>";
58.     $("#consola").html(html);
59. },
60. error:function(res) {
61.     $("#consola").html("ERROR: " + res.statusText);
62. }
63. });
64. }
65.
66.
67. /**
68.  * Solicita a Los servicios web implementados que Le envíe un ponente concreto,
69.  * especificado por su dni (#dni2). Este dato lo recibe del servicio en formato texto.
70.  *
71.  * @param {String} u Representa el Servicio al que queremos invocar, es decir
72.  *                     "oftalmologia": Servicio.java;
73.  *                     "oftalmologia2": Servicio2.java
74.  * @returns {String} Devuelve una cadena de texto en formato html con dicho dato
75.  */
76. function obtenerPonente(u) {
77.     var dni = $("#dni2").val();
78.     $.ajax({
79.         type: "GET",
80.         //url: "http://localhost:8080/ServicioWeb/oftalmologia/obtenerPonente/" + dni,
81.         //url: "http://localhost:8080/ServicioWeb/" + u + "/obtenerPonente/" + dni,
82.         url: url + u + "/obtenerPonente/" + dni,
83.         dataType: "text",
84.         cache: false,
85.         success: function(data) {
86.             var html = "<pre>" + data + "</pre>";
87.             $("#consola").html(html);
88.         },
89.         error:function(res) {
90.             $("#consola").html("ERROR: " + res.statusText);
91.         }
92.     });
93. }
94.
95.
96.
97.
98.
99. /**
100.  * Solicita a Los servicios web implementados que actualice un ponente, especificado
101.  * por su dni (#dni3). Este dato lo envía al servicio en formato json y recibe una
102.  * confirmación exitosa o no en formato texto.
103.  *
104.  * @param {String} u Representa el Servicio al que queremos invocar, es decir
105.  *                     "oftalmologia": Servicio.java;
106.  *                     "oftalmologia2": Servicio2.java
107.  * @returns {String} Devuelve una cadena de texto de confirmación
108.  */

```

```

109. function modificarPonente(u) {
110.     var dni = $("#dni3").val();
111.     var nom = $("#nom3").val();
112.     var ape = $("#ape3").val();
113.     var afi = $("#afi3").val();
114.     var pais = $("#pais3").val();
115.     $.ajax({
116.         type: "PUT",
117.         //url: "http://localhost:8080/ServicioWeb/oftalmologia/modificarPonente/" + dni,
118.         //url: "http://localhost:8080/ServicioWeb/" +u+ "/modificarPonente/" + dni,
119.         url: url + u + "/modificarPonente/" + dni,
120.         contentType: "application/json",
121.         dataType: "text",
122.         cache: false,
123.         data: JSON.stringify({
124.             "dni": dni, "nombre": nom, "apellidos": ape,
125.             "afiliacion": afi, "pais": pais
126.         }),
127.         success: function(data) {
128.             $("#consola").html(data);
129.         },
130.         error: function(res) {
131.             $("#consola").html("ERROR: " + res.statusText);
132.         }
133.     });
134. }
135.
136.
137. /**
138.  * Solicita a Los servicios web implementados que elimine un ponente según
139.  * su dni (#dni4). Recibe una confirmación exitosa o no en formato texto.
140.  *
141.  * @param {String} u Representa el Servicio al que queremos invocar, es decir
142.  *                     "oftalmologia": Servicio.java;
143.  *                     "oftalmologia2": Servicio2.java
144.  * @returns {String} Devuelve una cadena de texto de confirmación
145.  */
146. function eliminarPonente(u) {
147.     var dni = $("#dni4").val();
148.     $.ajax({
149.         type: "DELETE",
150.         //url: "http://localhost:8080/ServicioWeb/oftalmologia/eliminarPonente/" + dni,
151.         //url: "http://localhost:8080/ServicioWeb/" +u+ "/eliminarPonente/" + dni,
152.         url: url + u + "/eliminarPonente/" + dni,
153.         dataType: "text",
154.         cache: false,
155.         success: function(data) {
156.             $("#consola").html(data);
157.         },
158.         error: function(res) {
159.             $("#consola").html("ERROR: " + res.statusText);
160.         }
161.     });
162. }
163.
164.
165. /**
166.  * Solicita a Los servicios web implementados que añada un ponente, especificando
167.  * todos sus campos. Este dato lo envía al servicio en formato json y recibe una
168.  * confirmación exitosa o no en formato texto.

```

```

169.      *
170.      * @param {String} u Representa el Servicio al que queremos invocar, es decir
171.      *                   "oftalmologia": Servicio.java;
172.      *                   "oftalmologia2": Servicio2.java
173.      * @returns {String} Devuelve una cadena de texto de confirmación
174.      */
175.      function nuevoPonente(u) {
176.          var dni = $("#dni5").val();
177.          var nom = $("#nom5").val();
178.          var ape = $("#ape5").val();
179.          var afi = $("#afi5").val();
180.          var pais = $("#pais5").val();
181.          $.ajax({
182.              type: "POST",
183.              //url: "http://localhost:8080/ServicioWeb/oftalmologia/nuevoPonente",
184.              //url: "http://localhost:8080/ServicioWeb/" +u+ "/nuevoPonente",
185.              url: url + u + "/nuevoPonente",
186.              contentType: "application/json",
187.              dataType: "text",
188.              cache: false,
189.              data: JSON.stringify({
190.                  "dni": dni, "nombre": nom, "apellidos": ape,
191.                  "afiliacion": afi, "pais": pais
192.              }),
193.              success: function(data) {
194.                  $("#consola").html(data);
195.              },
196.              error: function(res) {
197.                  $("#consola").html("ERROR: " + res.statusText);
198.              }
199.          });
200.      }
201.
202.
203.      /**
204.      * Solicita a los servicios web implementados que añada un ponente, especificando
205.      * todos sus campos mediante el identificador de un formulario html. Recibe una
206.      * confirmación exitosa o no en formato texto.
207.      *
208.      * @returns {String} Devuelve una cadena de texto de confirmación
209.      */
210.      function nuevoPonenteForm() {
211.          var frm = $('#idForm');
212.          $.ajax({
213.              dataType: "text",
214.              type: frm.attr('method'),
215.              url: frm.attr('action'),
216.              data: frm.serialize(),
217.              cache: false,
218.              success: function(data) {
219.                  $("#consolaForm").html(data);
220.              },
221.              error: function(res) {
222.                  $("#consolaFormError").html("ERROR: " + res.statusText);
223.              }
224.          });
225.      }
226.
227.
228.      /**

```

```
229.      * Solicita al servicio web Servicio que le envíe el número actual de
230.      * ponentes que almacena.
231.      *
232.      * @returns {String} Devuelve una cadena que representa dicho número
233.      */
234.      function contarPonentes1() {
235.          $.ajax({
236.              type: "GET",
237.              //url: "http://localhost:8080/ServicioWeb/oftalmologia/contarPonentes1",
238.              url: url + "oftalmologia/contarPonentes1",
239.              cache: false,
240.              success: function(data) {
241.                  $("#numPonentes").html(data);
242.              }
243.          });
244.      }
245.
246.
247.      /**
248.      * Solicita al servicio web Servicio2 que le envíe el número actual de
249.      * ponentes que almacena.
250.      *
251.      * @returns {String} Devuelve una cadena que representa dicho número
252.      */
253.      function contarPonentes2() {
254.          $.ajax({
255.              type: "GET",
256.              //url: "http://localhost:8080/ServicioWeb/oftalmologia2/contarPonentes2",
257.              url: url + "oftalmologia2/contarPonentes2",
258.              cache: false,
259.              success: function(data) {
260.                  $("#numPonentes").html(data);
261.              }
262.          });
263.      }
```

## Anexo E: DDL y DML de Ponentes (pnet.sql)

```
1. CREATE DATABASE IF NOT EXISTS `pnet` /*!40100 DEFAULT CHARACTER SET utf8 */;
2. USE `pnet`;
3. -- MySQL dump 10.13 Distrib 5.7.17, for Win64 (x86_64)
4. --
5. -- Host: 127.0.0.1 Database: pnet
6. --
7. -- Server version 5.7.20-Log
8.
9. /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
10. /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
11. /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
12. /*!40101 SET NAMES utf8 */;
13. /*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
14. /*!40103 SET TIME_ZONE='+00:00' */;
15. /*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
16. /*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
17. /*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
18. /*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
19.
20. --
21. -- Table structure for table `ponentes`
22. --
23.
24. DROP TABLE IF EXISTS `ponentes`;
25. /*!40101 SET @saved_cs_client = @@character_set_client */;
26. /*!40101 SET character_set_client = utf8 */;
27. CREATE TABLE `ponentes` (
28.   `dni` varchar(45) NOT NULL,
29.   `nombre` varchar(45) DEFAULT NULL,
30.   `apellidos` varchar(45) DEFAULT NULL,
31.   `afiliacion` varchar(45) DEFAULT NULL,
32.   `pais` varchar(45) DEFAULT NULL,
33.   PRIMARY KEY (`dni`),
34.   UNIQUE KEY `dni_UNIQUE` (`dni`)
35. ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
36. /*!40101 SET character_set_client = @saved_cs_client */;
37.
38. --
39. -- Dumping data for table `ponentes`
40. --
41.
42. LOCK TABLES `ponentes` WRITE;
43. /*!40000 ALTER TABLE `ponentes` DISABLE KEYS */;
44. INSERT INTO `ponentes` VALUES
  ('1','Andrés','Martínez','UCA','Spain'),('2','Antonio','Ruiz','Cádiz
  FC','Spain'),('3','Manolo','el del Bombo','Su casa','Spain');
45. /*!40000 ALTER TABLE `ponentes` ENABLE KEYS */;
46. UNLOCK TABLES;
47. /*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;
48.
49. /*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
50. /*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
51. /*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
52. /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
```

```
53. /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;  
54. /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;  
55. /*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```