

Universidad de Cartagena

# Informe – JSP

Seminario de actualización

Amaury Rafael Ortega Camargo Y Ramiro Jose Verbel de la Rosa

## Contenido

1	Objetivo.....	2
2	Librerías.....	2
2.1	com.mysql.jdbc.Driver.....	2
2.2	com.mysql.jdbc.Connection .....	2
2.3	com.mysql.jdbc.Statement.....	2
3	Clases .....	3
3.1	Servidor .....	3
3.1.1	DB .....	3
3.1.2	Pc.....	3
3.1.3	Util .....	3
3.1.4	Controladores.ServidorIniciar .....	3
3.1.5	Controladores.ServidorDetener.....	4
4	Páginas web .....	5
4.1	index.jsp .....	5
4.2	iniciado.jsp .....	5
4.3	mensaje.jsp .....	7
5	Implementación .....	7
5.1	Arquitectura .....	7
5.2	Docker .....	8
5.2.1	Apache.....	8
5.2.2	Mysql.....	8
5.2.3	Resumido.....	8
5.2.4	Comandos.....	9
5.2.5	Imagen.....	9
6	Bibliografía .....	9

## 1 Objetivo

Comprender la comunicación de red usando la clase `HttpServlet`, junto a JSP, alojada en las librerías de java con el fin implementar un modelo cliente-servidor el cual proporcione instancias de servidores virtuales que contengan phpMyAdmin y MySQL accesibles mediante red por los clientes.

## 2 Librerías

Debido a que la clase `HttpServlet` se encuentra alojada en las librerías de java, se hace uso de la librería `mysql-connector-java- 5.1.44-bin.jar` la cual provee clases que son empleadas para la conexión a la base de datos encargada de registrar los servidores virtuales asignados a los clientes. Por lo tanto, únicamente se presentan en la clase DB logrando el uso de bases de datos MySQL, esta clase se explica posteriormente en el documento. Específicamente las clases usadas de esta librería son:

### 2.1 `com.mysql.jdbc.Driver`

Esta clase es llamada para establecer una conexión con la base de datos, solo es instanciada una vez logrado que el driver dentro de la JVM esté listo para ser usado por medio del siguiente código:

```
Class.forName("com.mysql.jdbc.Driver");
```

### 2.2 `com.mysql.jdbc.Connection`

Esta clase es usada para instanciar y comenzar una conexión hacia una base de datos, especificando la dirección de red, el puerto y el nombre de la base de datos; además es necesario especificar las credenciales usando usuario y contraseña por medio del siguiente código:

```
Connection conexion = (Connection) DriverManager.getConnection(
    "jdbc:mysql://" + db_ip + ":" + db_port + "/" + db_name,
    user, pass);
```

Esta clase también nos permite preparar sentencias SQL destinadas a dicha conexión usando el método `createStatement`.

```
Statement st = (Statement) conexion.createStatement();
```

Al finalizar el uso de dicha conexión, es posible cerrarla usando el método `close()`.

### 2.3 `com.mysql.jdbc.Statement`

Finalmente, esta clase nos permite ejecutar las sentencias SQL y en caso necesario retorna un objeto de tipo `ResultSet`. Del objeto es posible obtener el contenido de la DB por medio del nombre de las columnas usando el método `getString (String)`. Esto junto nos permite ejecutar cualquier sentencia en nuestra base de datos por medio de la siguiente estructura:

```
// Saca el ultimo valor de la BD para tener los puertos
Query = "SELECT * FROM `registros`ORDER BY idUsuario DESC LIMIT 1";
Statement st = (Statement) conexion.createStatement();
resultSet = st.executeQuery(Query);
while (resultSet.next()) {
    usuario.setPuertoPHP(Integer.parseInt(resultSet.getString("puertoPHP")));
    usuario.setPuertoSQL(Integer.parseInt(resultSet.getString("puertoSQL")));
}
```

## 3 Clases

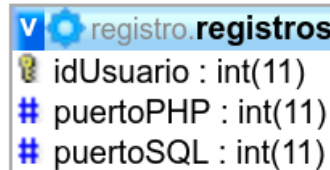
Se implementaron las siguientes clases:

### 3.1 Servidor

En la aplicación del servidor se tienen las siguientes clases:

#### 3.1.1 DB

Permite manejar la base de datos usando la librería mencionada anteriormente y la clase Pc que será explicada posteriormente. La base datos usada está conformada por la siguiente tabla:



registro.registros	
idUsuario	: int(11)
puertoPHP	: int(11)
puertoSQL	: int(11)

Esta clase nos permite trabajar con la base de datos por los siguientes métodos:

##### 3.1.1.1 *Pc insertar()*

Genera una instancia de la clase Pc la cual contiene el id, puertoPHP y puertoSQL correspondientes al nuevo cliente pidiendo el servicio teniendo en cuenta los registros existentes en la base de datos.

##### 3.1.1.2 *void desconectar()*

Detiene la conexión con la base de datos.

##### 3.1.1.3 *void eliminar(Integer id)*

Elimina el registro en la base datos del usuario con el id proporcionado.

##### 3.1.1.4 *void conectar()*

Crea la conexión con la base de datos.

#### 3.1.2 Pc

Clase para unificar la información de un cliente del servidor, contiene un id, puertoPHP y puertoSQL. Debido a la sencillez, esta clase solo tiene constructores y getters y setters.

#### 3.1.3 Util

Esta clase tiene como función, almacenar la dirección ip del servidor con la base de datos como string estático y una variable booleana para activar y desactivar el uso de docker.

#### 3.1.4 Controladores.ServidorIniciar

Esta clase es la encargada de ejecutar el método processRequest cuando el servidor recibe una petición GET o POST en el URL /servidoriniciar; gracias a la sencillez de los HttpServlet, no debemos preocuparnos por hacer la aplicación multi-hilo, de esto se encargará GlassFish. Al recibir una petición se crea una conexión a la base de datos usando la clase DB. Inmediatamente se crea un objeto de la clase Pc usando el método insertar() de la clase DB. Este objeto, que llamaremos usuario, es guardado en la sesión del navegador del cliente y nos desconectamos de la base de datos así:

```

DB base_datos = new DB();
base_datos.conectar();
Pc usuario = base_datos.insertar();

//Guardado de variable en sesion
request.getSession().setAttribute("pc", usuario);
base_datos.desconectar();

```

Procedemos a crear el servidor virtual que será dado al cliente usando la información en el objeto usuario. Esto se hace obteniendo un Shell en el sistema operativo del servidor con Docker instalado. Usando una instancia de la clase Runtime, es posible ejecutar comandos con el método exec, esto puede ser almacenado en un objeto de la clase Process lo que nos permitirá esperar que el comando en cuestión haya sido ejecutado usando el método waitFor(). Y finalmente se re-direcciona la sesión del cliente hacia iniciado.jsp

```

Process proceso;
Runtime shell = Runtime.getRuntime();
// COMANDO DOCKER
// docker run -d --rm -p [PuertoPHP]:80 -p [PuertoSQL]:3306 --name=server[ID] xxdrackleroxx/test
proceso = shell.exec("docker run -d --rm -p " + usuario.getPuertoPHP()
    + ":80 -p " + usuario.getPuertoSQL() + ":3306 --name=server"
    + usuario.getId() + " xxdrackleroxx/test");
proceso.waitFor();

request.getRequestDispatcher("iniciado.jsp").forward(request, response);

```

### 3.1.5 Controladores.ServidorDetener

Esta clase es la encargada de ejecutar el método processRequest cuando el servidor recibe una petición GET o POST en el URL /servidordetener?id=#. Al recibir una petición se obtiene el parámetro id lo cual identificara al cliente en el servidor. Procedemos a crear una conexión a la base de datos usando la clase DB. Esta conexión servirá para eliminar al cliente usando el id obtenido anteriormente usando el método eliminar de la clase DB.

```

Integer id = Integer.parseInt(request.getParameter("id"));

DB base_datos = new DB();
base_datos.conectar();
base_datos.eliminar(id);

```

Procedemos a detener y destruir el servidor virtual usando el id obtenido antes. Esto se hace obteniendo un Shell en el sistema operativo del servidor con Docker instalado. Usando una instancia de la clase Runtime, es posible ejecutar comandos con el método exec para detener y destruir el servidor virtual. Justo después, se guarda en la sesión del cliente, un mensaje diciendo que su servidor ha sido eliminado y finalmente se re-direcciona el navegador del cliente a una página web con este mensaje.

```

Process proceso;
Runtime shell = Runtime.getRuntime();
// COMANDO DOCKER

proceso = shell.exec("docker stop -t 0 server" + id);

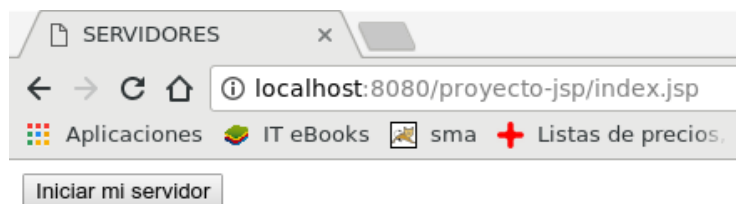
request.getSession().setAttribute("mensaje", "Servidor " + id + " eliminado");
request.getRequestDispatcher("mensaje.jsp").forward(request, response);

```

## 4 Páginas web

### 4.1 index.jsp

Sitio web mostrado en la dirección <http://IP-HOST:PUERTO/proyecto-jsp>.



Aquí se tiene un form, con un único botón, cuya acción es `servidoriniciar`, esta acción llama al controlador `ServidorIniciar` por una petición POST.

```

<form action="servidoriniciar" name="form" method="post">
    <button type="submit">Iniciar mi servidor</button><br>
</form>

```

### 4.2 iniciado.jsp

Una vez creado ejecutado el controlador `ServidorIniciar`, el navegador del cliente muestra esta página. Usando código java es posible obtener la variable usuario de tipo `Pc`, de la sesión del cliente así:

```

<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <%
      Pc usuario = (Pc) request.getSession().getAttribute("pc");
      Integer id = usuario.getId();
      Integer php = usuario.getPuertoPHP();
      Integer sql = usuario.getPuertoSQL();
    %>
    <title>Servidor <%=id%></title>
  </head>

```

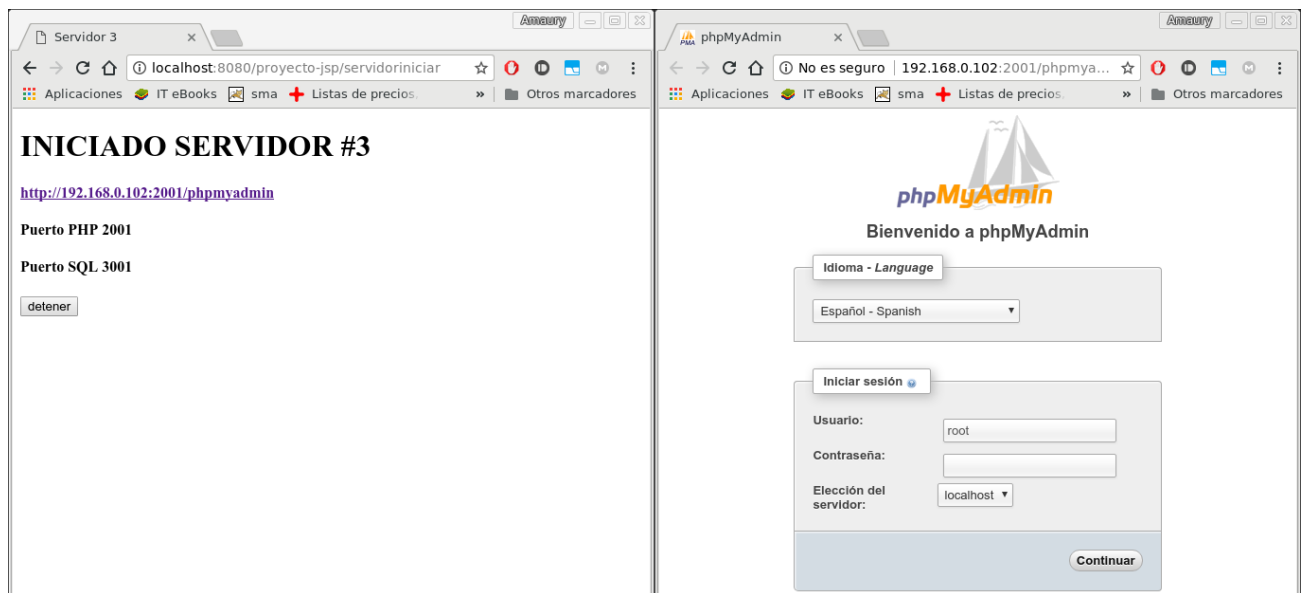
De esta forma podemos mostrar la información correspondiente a cada cliente directamente en la vista con el siguiente código:

```

<h1>INICIADO SERVIDOR #<%=id%></h1>
<h4><a href="http://<%=Util.ip%>:<%=php%>/phpmyadmin">
http://<%=Util.ip%>:<%=php%>/phpmyadmin
</a></h4>
<h4>Puerto PHP <%=php%></h4>
<h4>Puerto SQL <%=sql%></h4>

```

Dando como resultado lo siguiente:



Finalmente se tiene un form con la acción `servidordetener?id=#` donde # es el id del objeto usuario capturado previamente de la sesión.

```

<form action="servidordetener?id=<%=id%>" name="form" method="post">
  <button type="submit">detener</button><br>
</form>

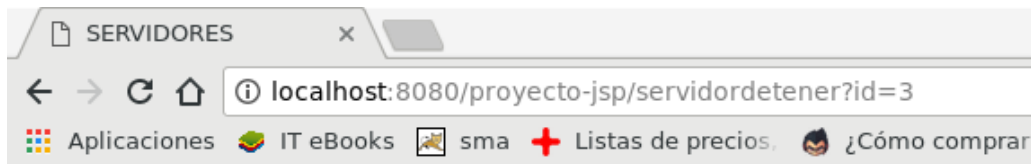
```

### 4.3 mensaje.jsp

Esta página es mostrada cuando el controlador ServidorDetener re-direcciona el navegador del cliente luego de destruir el servidor virtual. El controlador guarda en sesión un mensaje el cual es capturado en código java dentro el JSP así:

```
<h1><%=request.getSession().getAttribute("mensaje")%></h1>
```

Dando como resultado:



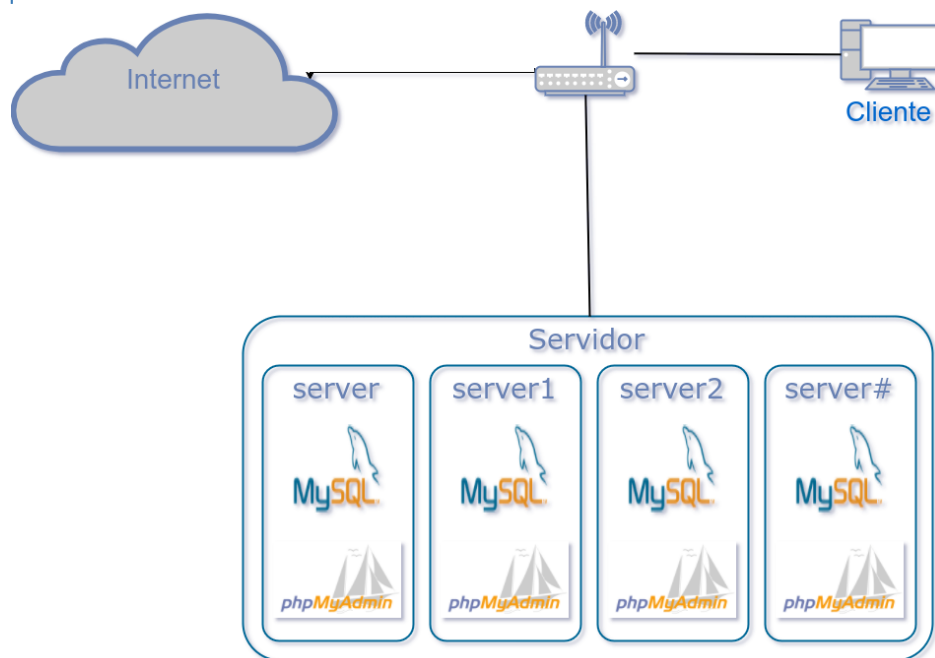
## Servidor 3 eliminado

### Inicio

Además, se deja un link hacia index.jsp para volver a hacer peticiones de servidores virtuales.

## 5 Implementación

### 5.1 Arquitectura





## 5.2 Docker

Docker es una herramienta open-source que nos permite realizar una 'virtualización ligera', con la que poder empaquetar entornos y aplicaciones que posteriormente podremos desplegar en cualquier sistema que disponga de esta tecnología.

Para ello Docker usa LXC (Linux Containers), que es un sistema de virtualización que permite crear múltiples sistemas totalmente aislados entre sí, sobre la misma máquina.

La gran diferencia es que una máquina virtual necesita contener todo el sistema operativo mientras que un contenedor Docker aprovecha el sistema operativo sobre el cual se ejecuta compartiendo el kernel.

Se usó como base la imagen `wnameless/mysql-phpmyadmin:latest` que viene con apache, phpmyadmin pero con configuraciones no funcionales para nuestro proyecto, todo esto bajo Ubuntu 12.04. Los cambios fueron los siguientes:

### 5.2.1 Apache

Se cambia `/etc/phpmyadmin/apache.conf` añadiendo `allow from all`. Y se reinicia el servicio con `service apache2 restart`. Esto para permitir conexiones fuera de la red P2P que se crea entre el host y el contenedor.

### 5.2.2 Mysql

Se cambia `/etc/mysql/my.cnf` modificando `bind-address = 127.0.0.1` a `bind-address = *` usando:

```
sed -i "s/.*bind-address.*/bind-address = 0.0.0.0/" /etc/mysql/my.cnf
```

Esto para permitir conexiones fuera de la red P2P que se crea entre el host y el contenedor. Luego se reinicia el servicio con:

```
mysqladmin shutdown & mysqld_safe
```

Luego, dentro del motor de base de datos se deben dar permisos al usuario root para ser usado por cualquier cliente en la red así:

```
echo "FLUSH privileges;" > sql.sql; echo "CREATE USER 'root'@'%';" >> sql.sql; echo  
"GRANT ALL PRIVILEGES ON * . * TO 'root'@'%' WITH GRANT OPTION  
MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0  
MAX_UPDATES_PER_HOUR 0 MAX_USER_CONNECTIONS 0;" >> sql.sql
```

```
mysqld < sql.sql
```

### 5.2.3 Resumido

Otra forma de hacer lo anterior es con un script creado por nosotros que puede ser usado así:

```
wget https://raw.githubusercontent.com/AmauryOrtega/Sem-  
Update/master/Proyecto-1-Corte/Docker/script.sh && chmod +x script.sh  
&& ./script.sh && rm script.sh
```

#### 5.2.4 Comandos

Es posible trabajar con contenedor de forma manual así:

Iniciar. `docker run -d --rm -p [PuertoPHP]:80 -p [PuertoSQL]:3306 --name=[Nombre] xxdrackleroxx/test`

Detener. `docker stop -t 0 [Nombre]`

#### 5.2.5 Imagen

Debido a la agrupación de los cambios en script BASH, se pudo simplificar nuestra imagen docker así:

FROM `wnameless/mysql-phpmyadmin:latest`

ADD `script.sh script.sh`

RUN `chmod +x script.sh && ./script.sh && rm script.sh`

Para más información, favor revisar, comentar o hacer peticiones en el repositorio de todo el proyecto (Ortega Camargo & Verbel de la Rosa, Repositorio Sem-Update, 2017) o en el repositorio de aprendizaje de Docker que se usó como fuente de información (Ortega Camargo, Repositorio Learning-Docker, 2017).

## 6 Bibliografía

Mestras, J. P. (s.f.). *Java EE – Servlets*. Obtenido de Dep. Ingeniería del Software e Inteligencia Artificial. Universidad Complutense Madrid: <https://www.fdi.ucm.es/profesor/jpavon/web/43-servlets.pdf>

ORACLE. (s.f.). *Class HttpServlet*. Obtenido de Java Platform, Enterprise Edition (Java EE) 7: <http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServlet.html>

Ortega Camargo, A. R. (2017). *Repositorio Learning-Docker*. Obtenido de GitHub.com: <https://github.com/AmauryOrtega/Learning-Docker>

Ortega Camargo, A. R. (s.f.). *Docker Hub*. Obtenido de xxdrackleroxx/test PUBLIC REPOSITORY: <https://hub.docker.com/r/xxdrackleroxx/test/>

Ortega Camargo, A. R., & Verbel de la Rosa, R. J. (2017). *Repositorio Sem-Update*. Obtenido de Github: <https://github.com/AmauryOrtega/Sem-Update>