

Universidad de Cartagena

Informe – JSON

Seminario de actualización

Amaury Rafael Ortega Camargo Y Ramiro Jose Verbel de la Rosa

Contenido

1	Objetivo	3
2	Librerías	3
2.1	javax.servlet.http.HttpServlet	3
2.2	javax.servlet.http.HttpServletRequest	3
2.3	javax.servlet.http.HttpServletResponse	3
2.4	org.apache.http.impl.client.HttpClients;	3
2.5	org.apache.http.impl.client.CloseableHttpClient;	4
2.6	org.apache.http.client.methods.HttpPost;	4
2.7	org.apache.http.client.methods.CloseableHttpResponse;	4
2.8	org.apache.http.HttpEntity;	4
2.9	org.apache.http.util.EntityUtils;	4
2.10	com.mysql.jdbc.Driver	4
2.11	com.mysql.jdbc.Connection	4
2.12	com.mysql.jdbc.Statement	5
3	Clases	5
3.1	Servidor	5
3.1.1	DB	5
3.1.2	Pc	5
3.1.3	Util	6
3.1.4	Controladores.ServidorIniciar	6
3.1.5	Controladores.ServidorDetener	7
4	Cliente	7
4.1	Pc	7
4.2	VentanaPrincipal	7
4.2.1	Inicio	7
4.2.2	Detener	8
5	Implementación	8
5.1	Arquitectura	9
5.2	Docker	9
5.2.1	Apache	9
5.2.2	Mysql	9
5.2.3	Resumido	10

5.2.4	Comandos.....	10
5.2.5	Imagen.....	10
6	Bibliografía	11

1 Objetivo

Comprender la comunicación de red mediante el uso de las clases alojadas en la librería org.apache.http y HttpServlet, usando JSON (JavaScript Object Notation) el cual es un formato de intercambio de información, 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 no es necesario añadirla, por el contrario la clase Gson la cual permite el manejo de JSON está alojada en la librería gson-2.8.2.jar, también se hace uso de la librerías httpclient-4.5.3.jar, httpcore-4.4.7.jar y commons-logging-1.2.jar, todas alojadas en org.apache.http con la finalidad que la aplicación java cliente logre recibir la información enviada mediante método Post del Servlet, además 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 e interfaces usadas en el proyecto son las siguientes:

2.1 javax.servlet.http.HttpServlet

Esta clase es usada como extensión de la clase ServidorIniciar la cual será explicada posteriormente en el documento.

```
public class ServidorIniciar extends HttpServlet {
```

De esta clase se utiliza el método doPost llamado por el servidor el cual permite que un servlet maneje peticiones POST.

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}
```

2.2 javax.servlet.http.HttpServletRequest

Es la interfaz para proporcionar información de solicitud de servlets HTTP, donde el contenedor crea un objeto de este tipo el cual es recibido como parámetro por doPost.

2.3 javax.servlet.http.HttpServletResponse

Es la interfaz para proporcionar funcionalidad específica de HTTP para enviar una respuesta, donde el contenedor crea un objeto de este tipo el cual es recibido como parámetro por doPost.

2.4 org.apache.http.impl.client.HttpClients;

Es usada para crear un CloseableHttpClient instanciado con la configuración predeterminada.

2.5 [org.apache.http.impl.client.CloseableHttpClient;](#)

Permite ejecutar la solicitud HTTP utilizando el contexto predeterminado mediante el método `.execute`.

2.6 [org.apache.http.client.methods.HttpPost;](#)

Se utiliza para solicitar que el servidor de origen acepte la entidad incluida en la solicitud del recurso identificado por la URL.

2.7 [org.apache.http.client.methods.CloseableHttpResponse;](#)

Se utiliza para recibir e interpretar un mensaje de solicitud que un servidor responde con el mensaje de respuesta HTTP

2.8 [org.apache.http.HttpEntity;](#)

Se utiliza para almacenar la entidad del mensaje de respuesta.

2.9 [org.apache.http.util.EntityUtils;](#)

Proporciona métodos necesarios para el manejo de entidades.

2.10 [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");
```

2.11 [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()`.

2.12 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

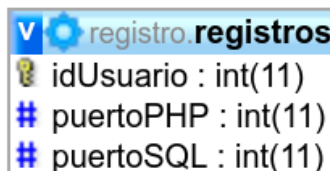
Se implementaron las siguientes clases:

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:



idUsuario	puertoPHP	puertoSQL
-----------	-----------	-----------

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 y nos desconectamos de la base de datos así:

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

```
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();
```

Luego serializamos el objeto usuario en Json, creado una instancia de Gson e invocamos el método toJson el cual recibe como parámetro el objeto a serializar, después mediante response que es el flujo de salida de respuesta HTTP especificamos que el contenido será de tipo JSON y que los caracteres de codificación serán UTF-8, se instancia un objeto de tipo Writer al cual recibe de parte response.getWriter(); un objeto de tipo PrintWriter con el cual podemos enviar datos al cliente, luego de esto se procede a enviar el Json y se cierra la salida.

```
String json = new Gson().toJson(usuario);
response.setContentType("application/json");
response.setCharacterEncoding("UTF-8");
Writer salida = null;
salida = response.getWriter();
salida.write(json);
salida.close();
```

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.

```
Process proceso;
Runtime shell = Runtime.getRuntime();
// COMANDO DOCKER
// docker run -d --rm -p [PuertoPHP]:80 -p [PuertoSQL]:3306
proceso = shell.exec("docker stop -t 0 server" + id);
```

4 Cliente

4.1 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.

4.2 VentanaPrincipal

Esta clase es la GUI que usa el cliente para pedir el servicio del servidor, aquí se tienen 2 métodos que corresponden a 2 botones, `Iniciar` y `Detener`.

4.2.1 Inicio

Se instancia un objeto de `CloseableHttpClient` con la configuración predeterminada, el cual nos permite ejecutar la solicitud HTTP utilizando el contexto predeterminado mediante el método `.execute` el cual recibe como parámetro un objeto de tipo `HttpPost` este se utiliza para solicitar que el servidor de origen acepte la entidad incluida en la solicitud del recurso identificado por la URL.

```
CloseableHttpClient httpclient = HttpClients.createDefault();
HttpPost httpPost = new HttpPost("http://" + ip + ":8080/proyecto-gson/servidoriniciar");
CloseableHttpResponse respuesta = httpclient.execute(httpPost);
```


Luego se obtiene la entidad de mensaje de esta respuesta, verificamos que contenga la entidad si es así usamos EntityUtils que proporciona métodos para tratar con entidades en este caso el método toString que lee el contenido de la unidad y lo devuelve como una cadena, verificamos que esta cadena contenga el Json y procedemos a deserializar nuestro objeto usuario mediante una instancia de Gson e invocamos el método .fromJson el cual recibe el Json y la clase del objeto; luego aseguramos que el contenido de la entidad se consuma completamente y que el flujo de contenido, si existe, se cierre; igualmente se cierra el flujo respuesta y libera todos los recursos del sistema asociados con él.

```
HttpEntity entity = respuesta.getEntity();

if (entity != null) {
    String json = EntityUtils.toString(entity);
    System.out.println(json);
    if (json.equals("ERROR")) {
        System.err.println("[LOG] Error inesperado, porfavor intente mas tarde");
    } else {
        user = new Gson().fromJson(json, Pc.class);
    }
}

EntityUtils.consume(entity);
respuesta.close();
```

4.2.2 Detener

Aquí igual que en inicio, se instancia de CloseableHttpClient con la configuración predeterminada, el cual nos permite ejecutar la solicitud HTTP

```
HttpPost httpPost = new HttpPost("http://" + ip + ":8080/proyecto-gson/servidordetener?id=" + user.getId());
```

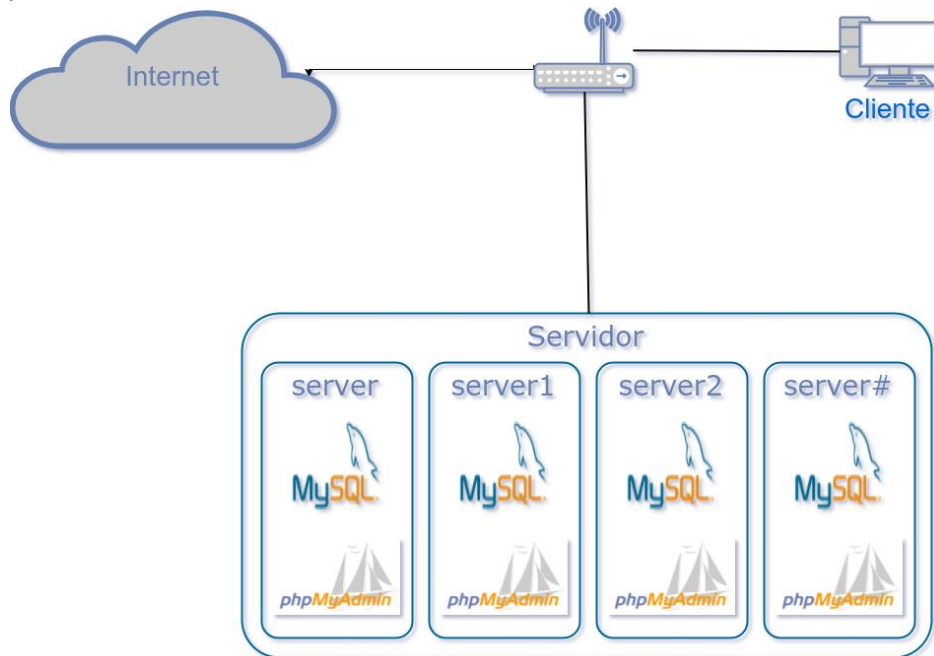
En el cual pasamos el id del usuario capturado previamente en el inicio, llamando a ServidorDetener el cual finaliza y destruye el contenedor.

5 Implementación



Pantalla principal cliente

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:

```
mysqldadmin 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

- Foundation, T. A. (04 de Septiembre de 2017). *Http Components*. Obtenido de Inicio rápido de HttpClient: <https://hc.apache.org/httpcomponents-client-ga/quickstart.html>
- Foundation, T. A. (s.f.). *Apache HttpCore 4.4.7 API*. Obtenido de Package org.apache.http: <https://hc.apache.org/httpcomponents-core-ga/httpcore/apidocs/org/apache/http/>
- Google. (2017). *Repositorio gson*. Obtenido de <https://github.com/google/gson>
- 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>
- Rodríguez, M. A. (17 de Septiembre de 2012). *Jugando con JSON en Java y la librería Gson*. Obtenido de <https://www.adictosaltrabajo.com/tutoriales/gson-java-json/>