# Práctica 1:

Morales Flores Víctor Leonel

25 de Febrero de 2019

# 1 Introducción

Aunque las herramientas tecnológicas para desarrollar aplicaciones web han evolucionado con el tiempo, cualquier framework que trabaje con el lenguaje Java tendrá en el fondo un Servlet.

Un servlet es una clase Java que soporta peticiones HTTP, brindándonos todas las capacidades que tiene Java en la web.

Para counicarnos con el cliente, Java nos proporciona de dos objetos: HTTP Request que contiene las peticiones del cliente y HTTP Response que nos permite responder a la petición de un usuario que se conecte al servicio.

A lo largo de esta práctica se busca aprender cómo invocar un servlet, la diferencia entre los métodos GET y POST, como recuperar las cabeceras del protocolo HTTP desde nuestro objeto request, cómo generar páginas de contenido dinámico y realizar consultas báscias a una Base de Datos para insertar o recuperar datos.

Aunque servlets vuelve el desarrollo de aplicaciones web un proceso tardado, los frameworks terminan por generar un servlet cuando los usamos. Saber cómo funciona un servlet nos permitirá poder entender de forma más transparente el funcionamiento de los frameworks.

# 2 Desarrollo

## 2.1 Iniciando el proyecto

Crearemos un proyecto y crearemos dos paquetes:

El primer paquete será llamad mx.ipn.escom.wad.servlets.practica1 y es donde almacenaremmos todos los servlets que creemos.

El segundo paquete será el que contendrá la clase para la conexión a la base de datos y será llamada: mx.ipn.escom.database

En nuestra carpeta src/main/webapp tendremos todos los archivos HTML, JavaScript o JSPs.

En el archivo index.jsp se colocó un menú para poder acceder a los distintos ejercicios que se propusieron en la práctica. Una vez iniciado el proyecto con nuestro servidor de desarrollo Jetty podremos acceder al proyecto a través de la ruta http://localhost:8080/3cm6-practica1-mfvl/

En nuestro archivo pom.xml agregaremos las siguientes dependencias:

```
<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
        <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
```

```
    <version>42.2.5</version>
</dependency>
```

## 2.2 Hello World!

En este primer servlet lo único que hacemos es programar un servlet que nos mostrará el mensaje "Hello World!" cuando solicitemos el servicio.

Para esta parte de la práctica llamaremos al servlet 'HelloWorld'. En este servlet implementaremos el método GET para que responda a peticiones de este tipo(por el URL).

Lo primero que hay que hacer es establecer el contentType a través de nuestro objeto response con su método setContentType. Posteriormente obtendremos el objeto PrintWriter para poder escribir en el fujo de salida y con este flujo escribiremos la respuesta usando etiquetas HTML:

```
response.setContentType("text/html");
PrintWriter out=response.getWriter();
out.println("<html>");
out.println("<head>");
out.println("<title>");
out.println("WAD:Practica 1.1");
out.println("</title>");
out.println("</head>");
out.println("<body>");
out.println("<h1>Hello world!</h1>");
out.println("</body>");
out.println("</html>");
```

## 2.3 HTTP Protocol Header

El siguiente ejercicio nos permite recuperar los headers que vienen dentro del protocolo HTTP. En él podemos encontrar información como e host y el User-Agent.

Para desarrollar este punto haremos uso del método getHeaderNames() que pertenece a nuestro objeto request y lo almacenaremos en un Ennumeration. Una vez almacenado iteraremos hasta que nuestro objeto no tenga más elementos. En cada iteración obtendremos el valor de nuestro Header a través de nuestro método getHeader(nombre_header) donde le pasaremos como parámetro el nombre de nuestro Header que fue recuperado de nuestro Ennumeration como se muestra en las siguientes líneas de código. Es importante mencionar que esto se colocó dentro del método doGet de nuestro Servlet y en líneas anteriores a las que se muestras se estableció el contentType.

```
Enumeration<String> nombre_headers=request.getHeaderNames();

while(nombre_headers.hasMoreElements())
{
```

```
        String nombreH=nombre_headers.nextElement();
        String value=request.getHeader(nombreH);
        out.println("<li>"+nombreH+": "+value+"</li>");
}
```

## 2.4 Service Counter

Para esta sección se inicializa un contador 'counter' en el método doGet() con el valor de cero. Cada vez que se que hagamos una petición al servlet, el valor de counter incremetará en 1. Sin embargo, al realizar peticiones al servlet el valor nunca incrementa debido a que cada vez que se hace una petición se invoca al método service que siempre está inicializando la variable en 0.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
        int counter=0;

        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        counter++;
        out.println("<html>");
        out.println("<head>");
        out.println("<title>");
        out.println("WAD:Practica 1.3");
        out.println("</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Service counter</h1>");
        out.println("<h4>Counter:"+counter+"</h4>");
        out.println("</body>");
        out.println("</html>");
}
```

Cada vez que actualicemos o hagamos una petición al servlet veremos que la variable no incrementa por más peticiones que le hagamos. Esto se debe al alcance de la variable ya que cada vez que invoquemos el método doGet por una petición counter volverá a valer 0.

## 2.5 Global Counter

A diferencia del ejercicio anterior, esta vez el contador se inicializa como variable de instancia del servlet y ya no en el método doGet(), esto produce que cada vez que se invoque al servlet, el contador no será inicializado por el método y podremos ver cómo es que cada vez que le hagamos una petición al servlet, esta variable incrementará.

```
public class GlobalCounter extends HttpServlet {
        private int counter=0;
        protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
                        throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                counter++;
                out.println("<html>");
                out.println("<head>");
                out.println("<title>");
                out.println("WAD:Practica 1.3");
                out.println("</title>");
                out.println("</head>");
                out.println("<body>");
                out.println("<h1>Global counter</h1>");
                out.println("<h4>Counter:"+counter+"</h4>");
                out.println("</body>");
                out.println("</html>");
        }
}
```

En el código podemos visualizar la clase Servlet completa a diferencia de los otros ejemplos. EL motivo por el que se colocó el código totalmente y no parcialmente es para que se pueda visualizar el alcance de nuestra variable counter para este ejercicio. Posteriormente, en la sección de pruebas se anexa la vista del contador sobre el navegador después de hacer varias peticiones. Como se ve en el código, sólo se asigna 0 una ocasión y después solo se hacen incrementos. Debido a que el servlet se instancía una sola vez cuando se levanta nuestro contenedor, independientemente del lugar donde se haga la petición, el valor de nuestro contador incrementará por el alcance de la variable(variable de instancia).

## 2.6 Method GET and POST

En esta sección usaremos los métodos GET y POST para recibir peticiones.

### 2.6.1 Método GET

El caso de que nos comuniquemos al servlet a través de una petición GET el servlet nos deberá mostrar uan tabla con todos los ussuarios registrados en la base de datos. Para ello, se hará la conexión a la base de datos y se generará el HTML de forma dinámica dependiendo del número de registros que existan en la base de datos.

```
String query="SELECT tx_first_name, tx_last_name_a,tx_last_name_b,"+
            "tx_curp,fh_birth,tx_login FROM person INNER JOIN users "+
```

```
                "ON person.id_person=users.id_user;" ;
ResultSet resultado=null;
try
{
        Conexion con=new Conexion();
        resultado=con.execute(query);
        con.close();
        while(resultado.next())
        {
                out.println("<tr>");
                out.println("<td>"+resultado.getString(1)+"</td>");
                out.println("<td>"+resultado.getString(2)+"</td>");
                out.println("<td>"+resultado.getString(3)+"</td>");
                out.println("<td>"+resultado.getString(4)+"</td>");
                out.println("<td>"+resultado.getString(5)+"</td>");
                out.println("<td>"+resultado.getString(6)+"</td>");
                out.println("</tr>");
        }
}
catch(Exception ex)
{
        System.out.println("Error en la lectura a la BD:"+ex.toString());
}
```

Como se puede apreciar en el código, el número de veces que creará una nueva fila en la tabla dependerá del número de filas que nos regrese la consulta a la base de datos.

Para accesar a las columnas de nuestro resultado podemos hacerlo del método getString() del objeto ResultSet. Para este método el primer elemento se encuentra en la posición 1 a diferencia de otros elementos como los arreglos donde el primer índice es el 0.

Es importante el bloque try...catch... porque tanto en la conexión a la base de datos como en el acceso a los elementos de nuestro ResultSet podríamos generar una excepción.

La conexión y consultas hacia la base de datos se haccen a través del objeto Conexion que posteriormente se mostrará y explicará.

### 2.6.2 Método POST

Si queremos ingresar a un nuevo usuario completaremos en formulario 'form5.html' y tras dar clic en 'Send' el formulario hará una petición POST a nuestro servlet. La implementación de este método consiste en recuperar los valores del formulario a través del método getParameter() de nuestro objeto request.

```
String fname=request.getParameter("name");
String lname=request.getParameter("lname");
String slname=request.getParameter("sname");
```

```
String curp=request.getParameter("curp");
String bd=request.getParameter("birthday");
String login=request.getParameter("login");
String pswd=request.getParameter("pswd");
```

Una vez que hayamos recuperado los datos crearemos nuestras consultas de tipo insert. Crearemos una conexión con la base de datos 'homework-6' que fue proporcionada por el profesor para realizar la tarea 6.

```
String query1="INSERT INTO PERSON (tx_first_name,tx_last_name_a,"+
                "tx_last_name_b,tx_curp,fh_birth) VALUES"+
                "('"+fname+"','"+lname+"'," +
                "'"+slname+"','"+curp+"','"+bd+"')";
                String query2="SELECT id_person FROM PERSON WHERE tx_curp LIKE '"+
                curp+"'";
                try {
                        Conexion con=new Conexion();
                        con.insert(query1);
                        ResultSet r1=con.execute(query2);
                        r1.next();
                        String id=r1.getString(1);
                        String query3="INSERT INTO USERS VALUES("+
                        id+",'"+login+"','"+pswd+"')";
                        con.insert(query3);
                        con.close();
                }
                catch(Exception ex)
                {
                        System.out.println("Error al insertar nuevo usuario:" +
                        ex.toString());
                }
```

Si la conexión se estableció comenzaremos a ejecutar nuestras consultas con nuestro objeto con(perteneciente a la clase Conexion).

Debido al diseño de nuestra base de datos tendremos que hacer 2 consultas de tipo insert y ina consulta de tipo select.

Observando el diagrama de la base de datos veremos que nuestra clase Users de la base de datos tiene una llave foránea que pertenece a la clase Person. Sin embargo, la llave foránea es id_person que es asignada por la base de datos de forma automática.

Por esta razón primero ejecutaremos un 'insert' a la clase Person, después a través de un 'select' obtendremos 'id_person' y el valor de nuestro ResultSet será el que utilizaremos para poder hacer nuesto segundo 'insert' a la clase Users de nuestra base.

### 2.6.3 Clase Conexion

Como se comentó anteriormente, esta clase se creó para hacer las conexiones y consultas hacia la base de datos. A continuación se coloca el código fuente:

```
public class Conexion {
        private Connection con;
        public Conexion()
        {
                try
                {
                        Class.forName("org.postgresql.Driver");
                        String url="jdbc:postgresql://localhost:5432/homework-6";
                        con=DriverManager.getConnection(url,"postgres","postgres");
                }
                catch(Exception ex)
                {
                        System.out.println("Error al conectar:"+
                        ex.toString());
                }

        }
        public ResultSet execute(String query)
        {
                ResultSet res=null;
                try {
                        Statement st=con.createStatement();
                        res=st.executeQuery(query);
                }
                catch(Exception ex)
                {

                }
                return res;
        }
        public void insert(String query)
        {
                try {
                        Statement st=con.createStatement();
                        st.executeUpdate(query);
                }
                catch(Exception ex)
                {
                        System.out.println("Error al insertar:"+
                        ex.toString());
                }
        }
```

```
        public void close()
        {
                try
                {
                        con.close();
                }
                catch(SQLException ex)
                {
                        System.out.println("Error al cerrar la conexión:"+
                        ex.toString());
                }
        }
}
```

En el código fuente se puede notar que al instanciar un objeto de esta clase las cadenas de texto que se encargan de hacer la conexión a una base de datos en específico están fijas, esto quiere decir que siempre que instanciemos un objeto de la clase Conexion, se conectará a nuestra base de datos 'homework-6' con el usuario 'postgres'.

El método execute nos retorna un objeto ResultSet y se debe utilizar para consultas de tipo 'select'.

En el caso del método insert(), en realidad puede ejecutar consultas de tipo insert o delete pero dado el alcance del objetivo de la práctica se nombró 'insert' para facilitar el entendimiento.

Es importante destaca que nuestra Clase tiene una variable de instancia de tipo Connection que es por la cual hacemos todas las ejecuciones de las queries.

El último método que tenemos es el close() que simplemente cerrará la conexión de nuestro objeto de tipo Connection.

# 3 Pruebas

## 3.1 Página principal

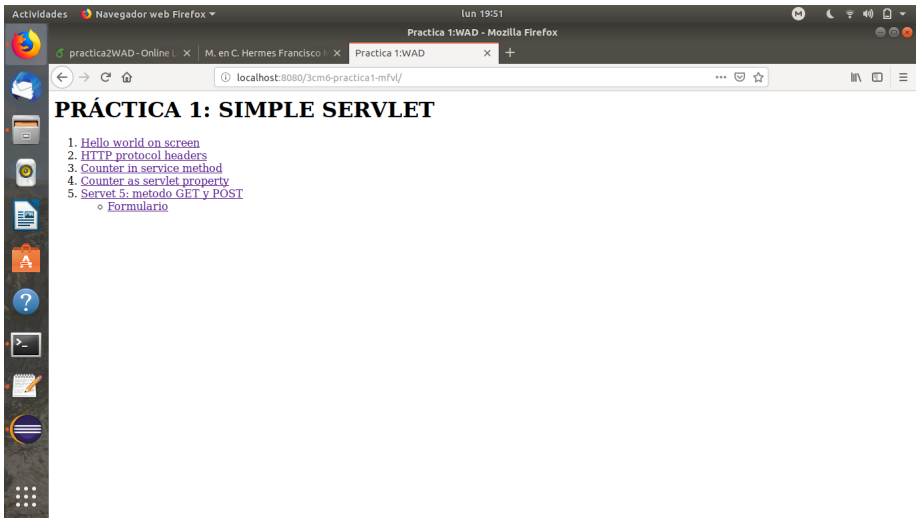


Figure 1: Menú principal para accesar a los distintos ejercicios de la práctica.

## 3.2 Hello World!

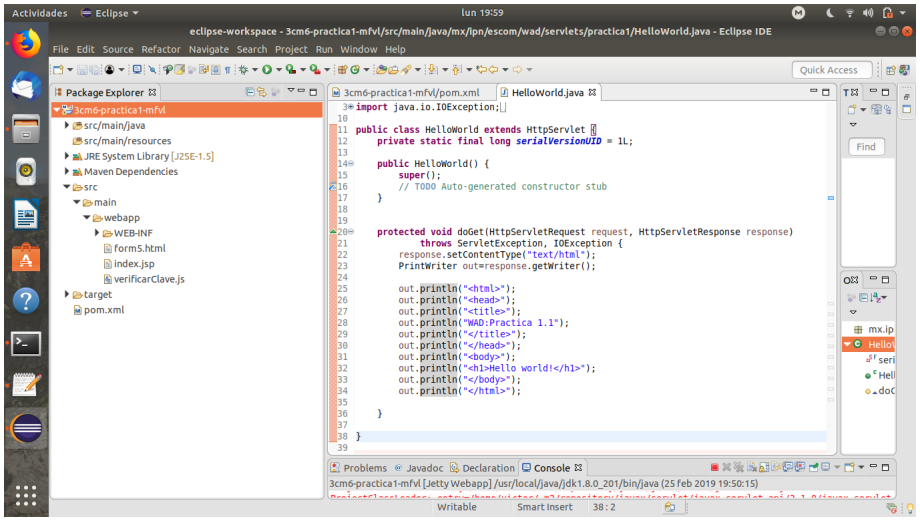


Figure 2: Servlet HelloWorld en ejecución.

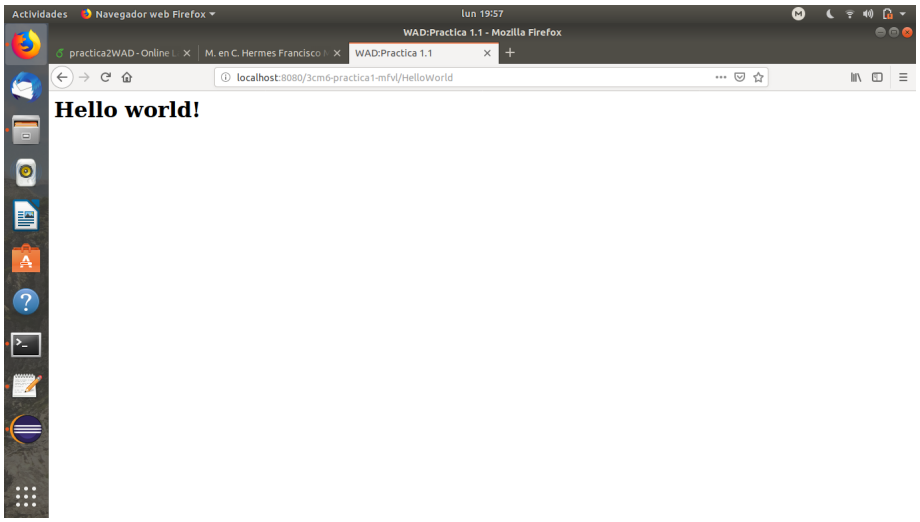


Figure 3: Código fuente del servlet HelloWorld.

## 3.3 HTTP Protocol Header

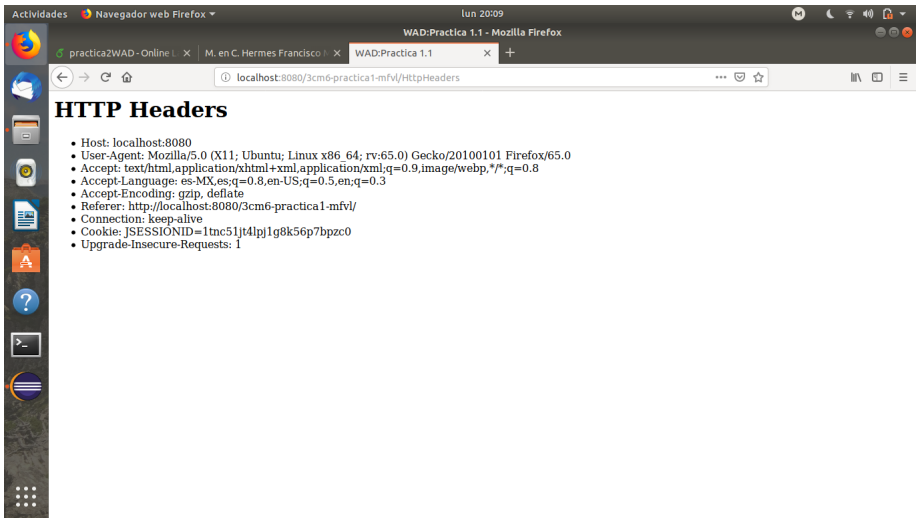


Figure 4: Servlet HttpHeaders respondiendo a la solicitud por el método GET.

## 3.4 HTTP Protocol Header

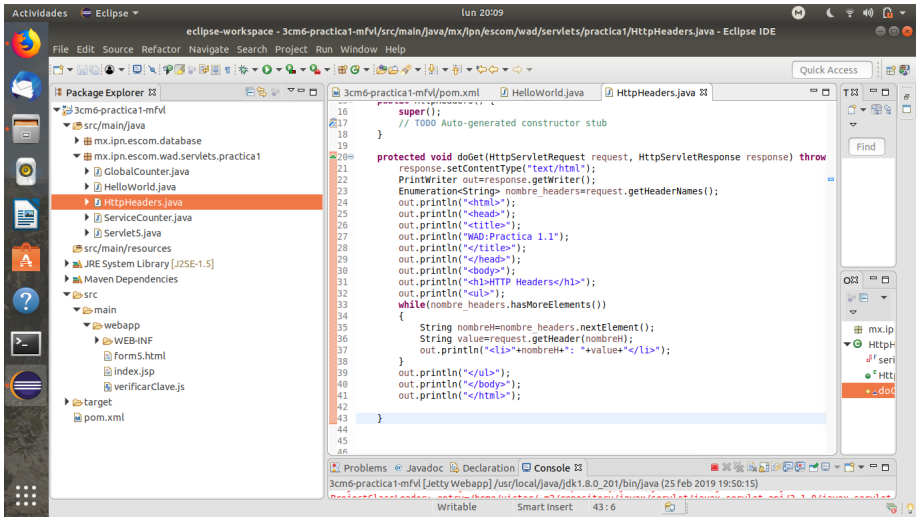


Figure 5: Código fuente del servlet HttpHeaders.

## 3.5 Service Counter

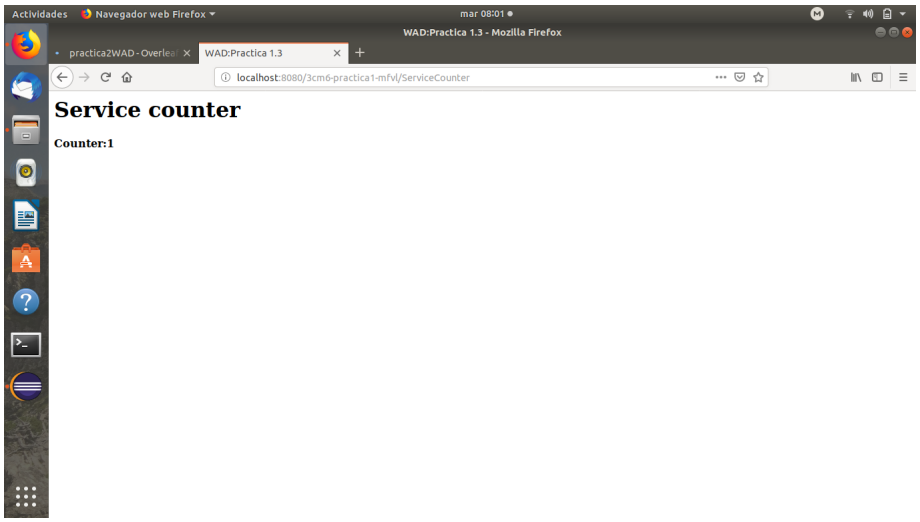


Figure 6: Servlet ServiceCounter sin encremento tras distintas peticiones.

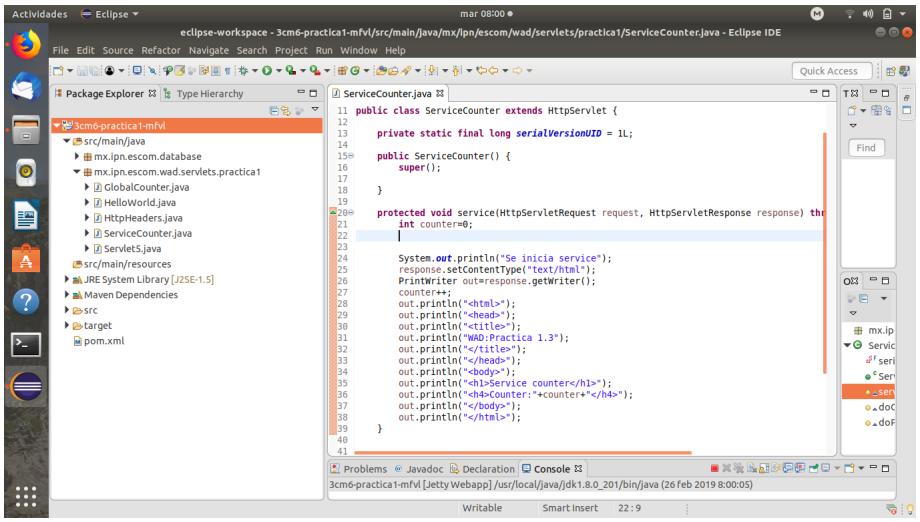


Figure 7: Código fuente del servlet ServiceCounter.

## 3.6 Global Counter

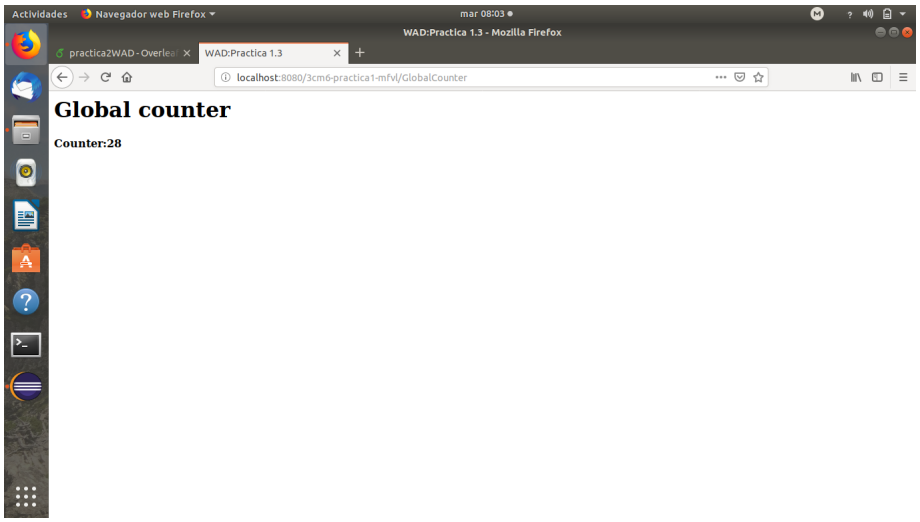


Figure 8: Servlet GlobalCounter tras varias peticiones.

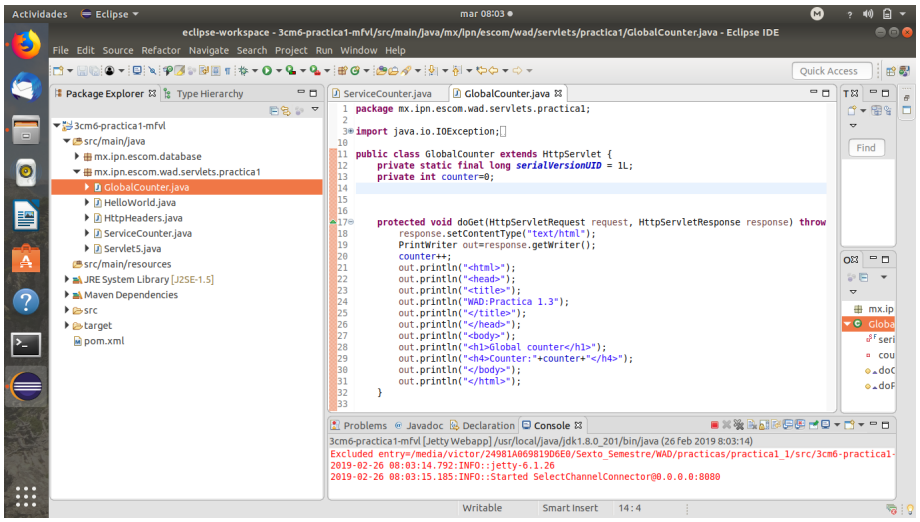


Figure 9: Código fuente del servlet GlobalCounter.

## 3.7 Method GET and POST

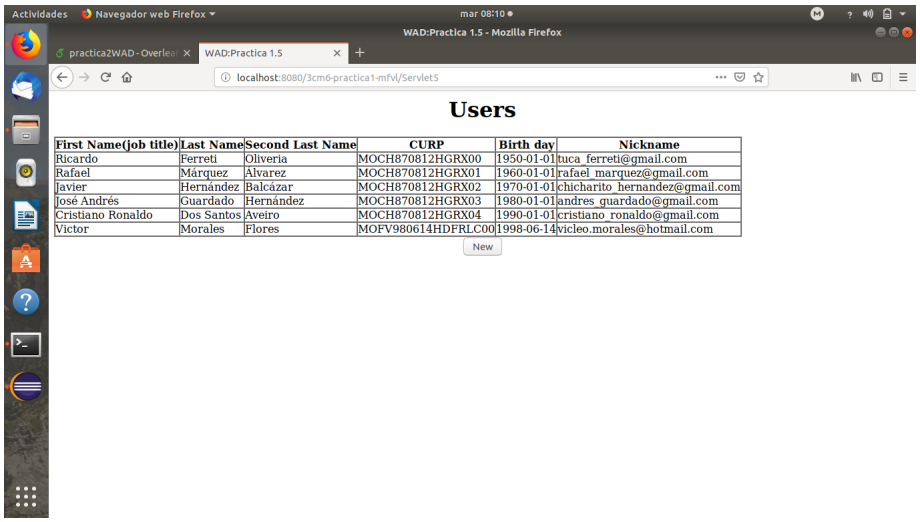


Figure 10: Tabla dinámica a partir de la Base de Datos desplegada por el método doGet().

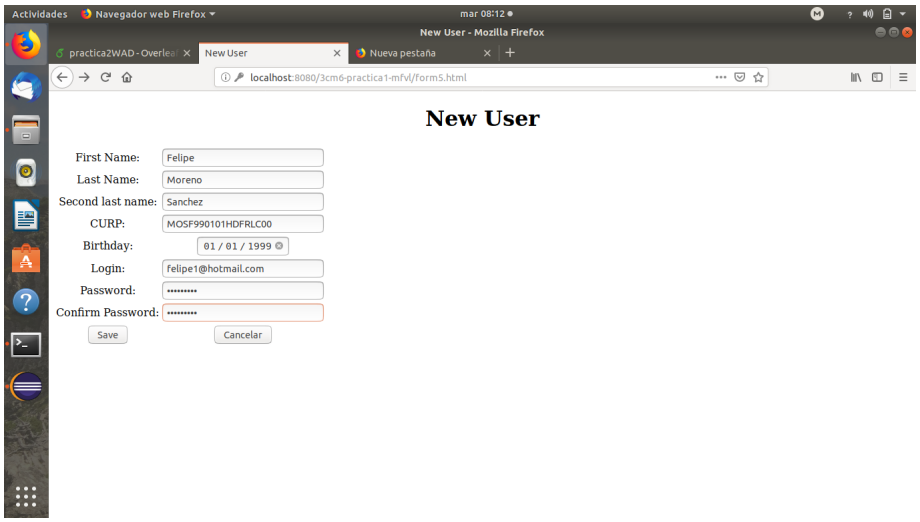


Figure 11: Formulario HTML que invocará al Servlet5 por el método POST

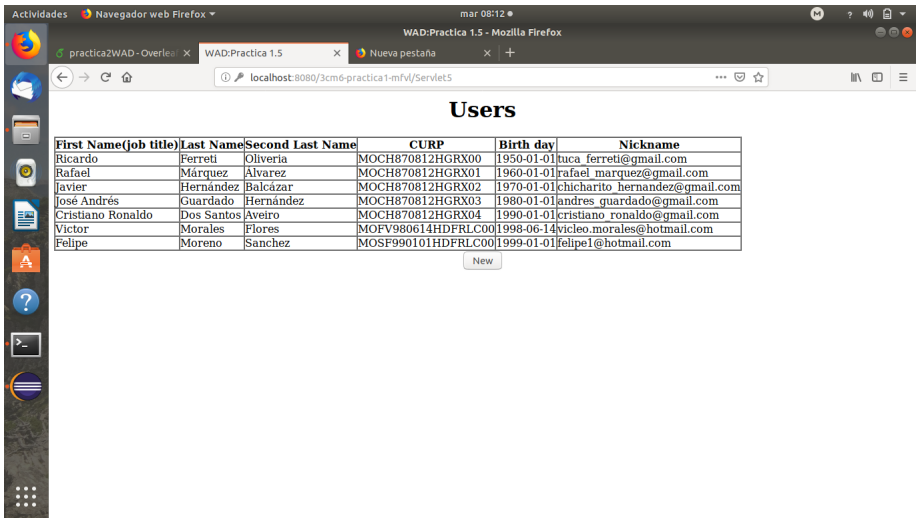


Figure 12: Tabla dinámica tras la inserción de un nuevo registro a la base de datos con el formulario anterior

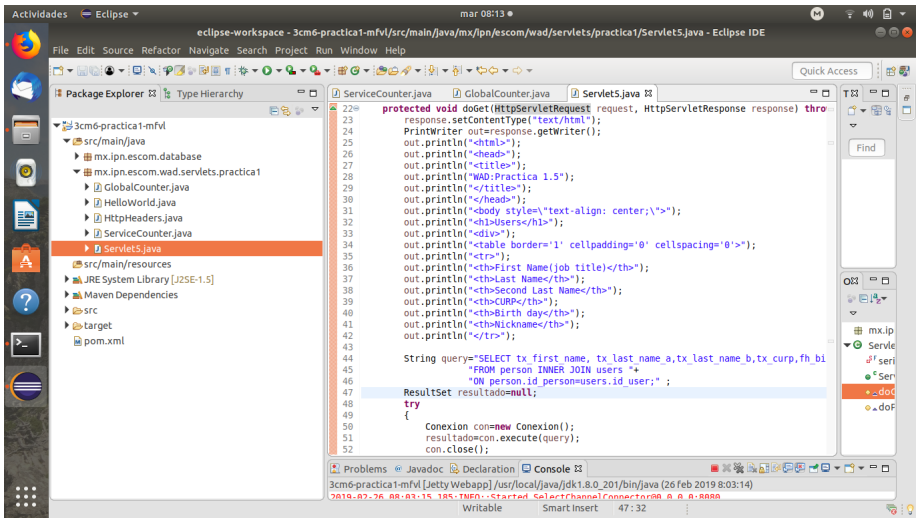


Figure 13: Código fuente del método doGet()

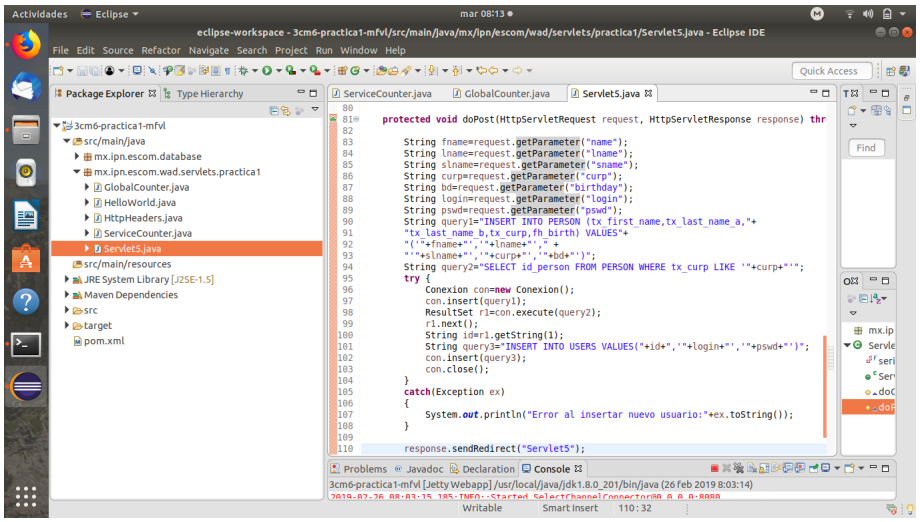


Figure 14: Código fuente del método doPost()

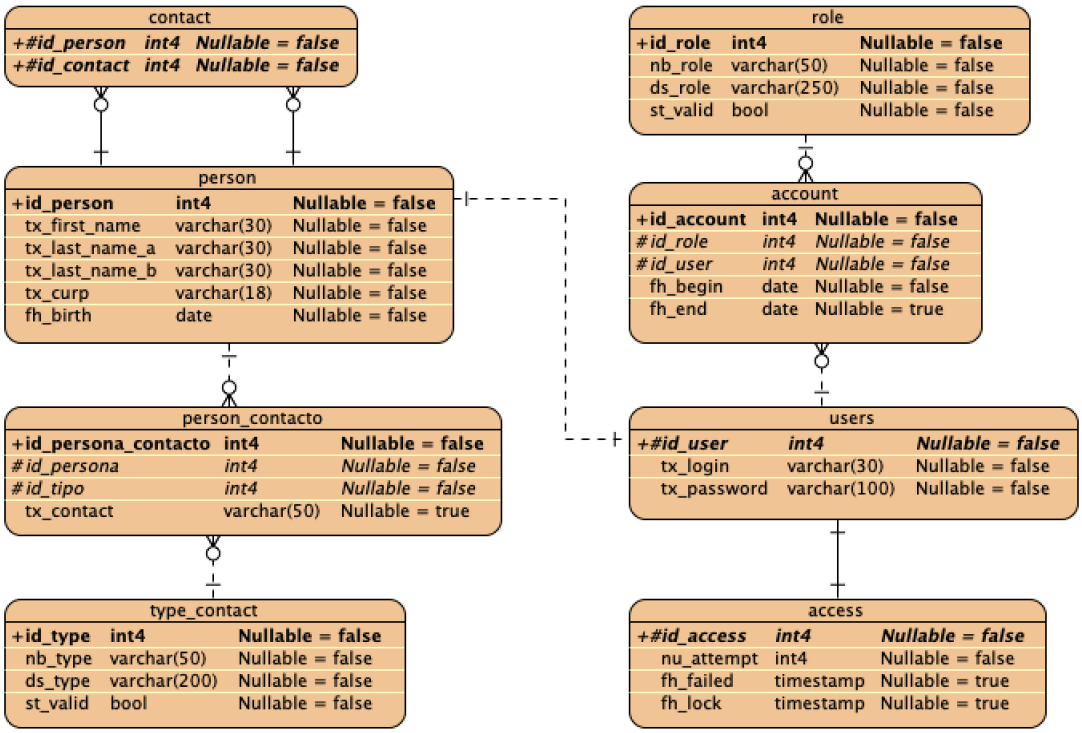


Figure 15: Modelo de la base de datos entregado por el profesor M. en C. Hermes Francisco Montes Casiano

# 4 Conclusiones

El API de Servlets nos proporciona muchos métodos a través de nuestros objetos HttpRequest y HttpResponse para poder obtener información de nuestro cliente.

Cunado un servlet es invocado a través del URL el método que es ejecutado es el método doGet(). Podemos generar distintos comportamientos para un mismo servlet a través de la implementación de los métodos doPost() y doGet() como se vio en el ejercicio cinco.

La elaboración de páginas dinámicas por medio de Servlets genera códigos muy largos y hacer sistemas enteros podría ser una labor muy dificil usando simplemente Servlets.