INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Práctica 1: Servlets Simples.

Unidad de aprendizaje: Web Application Development

Grupo: 3CM6

Alumnos(a):
Ramos Diaz Enrique

Profesor(a):
Montes Casiano Hermes Francisco

26 de febrero 2019

Índice

1. Introducción

En ésta practica vamos a revisar la estructura y los métodos que componen a un servlet simple, así como su comportamiento, objetos que maneja, parámetros, auxiliares para escritura web, etc.

Consta de cinco puntos explicados en el desarrollo, que a medida que avancemos, irán aumentando de complejidad, utilizando más métodos, objetos, procesamiento de información, etc.

Se realizará desde un HolaMundo, obtención de cabeceras del protocolo HTTP en las peticiones, hasta conexiones con una base de datos y operaciones se selección e inserción por medio de formularios HTML.

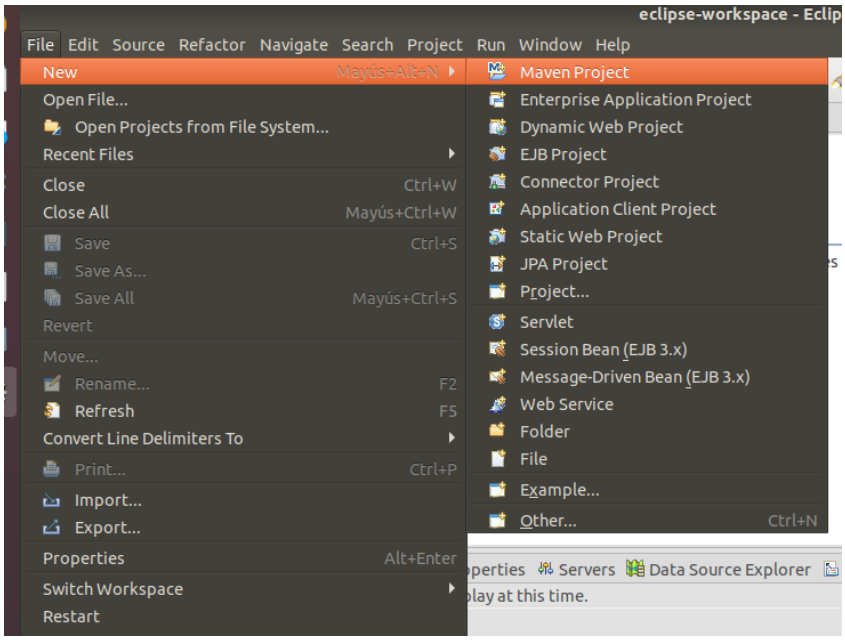
También vamos a revisar la relación de los servlets con otras clases Java, y explorar la capacidad para generar páginas web dinámicas, utilizando un objeto especial que permite escribir variables de Java como si fuera HTML plano.

Por último, hacemos una breve explicación de como crear un proyecto en el IDE Eclipse, y configurar el archivo pom.xml para añadir las dependencias que necesite este desde el servidor Maven.

2. Desarrollo

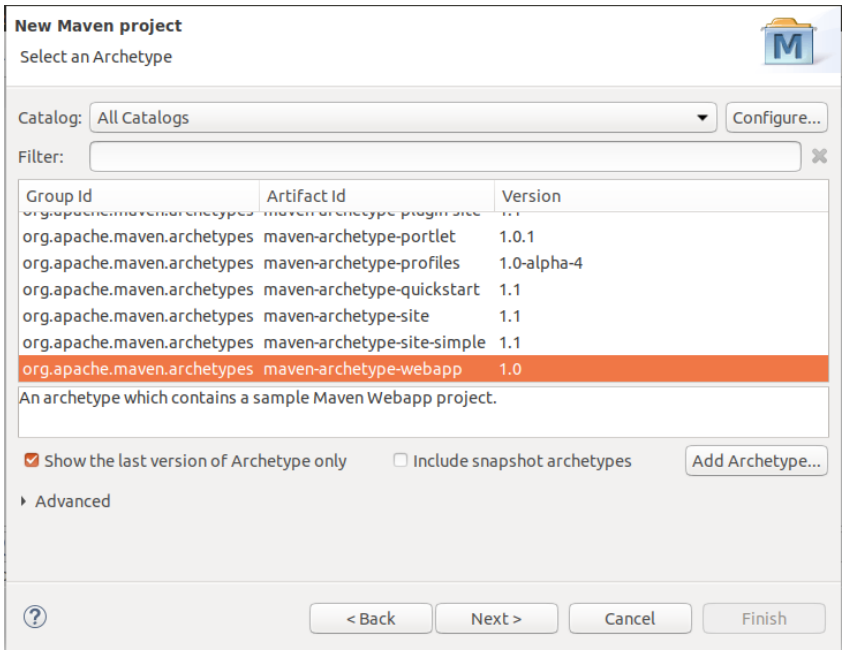
2.1. Creando un proyecto en Eclipse

1. Abrimos el IDE Eclipse, y damos clic en la pestaña File > New > Maven Project

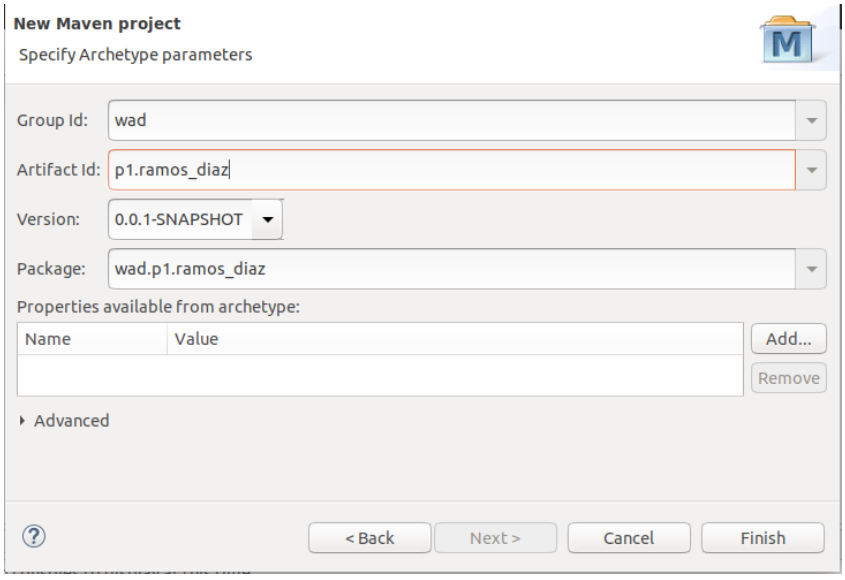


2. Seleccionamos el workspace en donde se guardarán los archivos del proyecto.

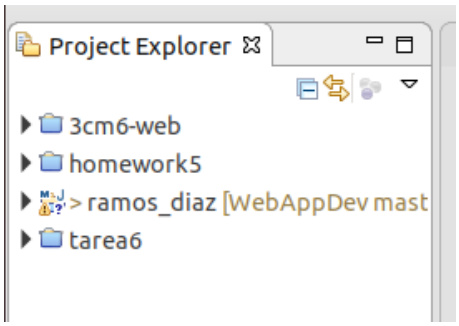
3. Seguimos las indicaciones. Cuando nos pida elegir el arquetipo, seleccionamos ”maven-archetype-webapp”, versión 1.0



4. Posteriormente, ingresamos el Group ID y el Artifact ID que queramos, y damos clic en Finish.



5. Nuestro proyecto aparecerá en el Project Explorer.

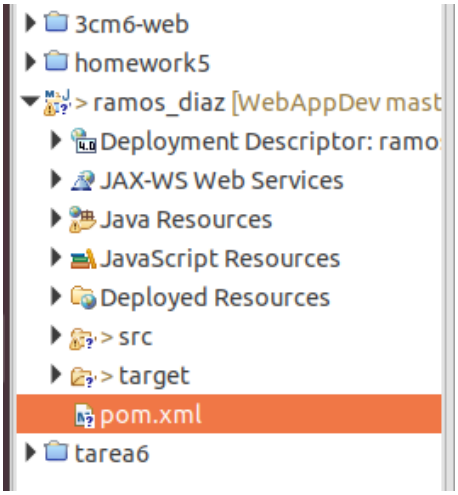


6. Si lo deseamos, podemos crear la carpeta /src/main/java y dentro de ella paquetes, que tendrá todas las clases y servlets del proyecto.

2.2. Dependencias de Maven

Para ésta práctica, necesitamos dos dependencias de Maven: una son las clases Servlet de Java, que poseen todos los métodos y objetos necesarios, y el otro es el conector al gestor de base de datos PostgreSQL 9.4. Ambos son archivos JAR y se cargan en nuestro proyecto de la siguiente manera:

- Dentro de la carpeta de nuestro proyecto, nos dirigimos al archivo **pom.xml** y lo abrimos.



- Vamos a añadir las dependencias copiando y pegando el siguiente texto. Este bloque de código debe de estar dentro de las etiquetas <project> </project>:

```
<dependencies>
   <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
   </dependency>
   <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>
      <scope>provided</scope>
   </dependency>
   <!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->
   <dependency>
       <groupId>org.postgresql</groupId>
       <artifactId>postgresql</artifactId>
       <version>42.2.5</version>
   </dependency>
</dependencies>
```

Ahora entraremos de lleno con el desarrollo de los puntos especificados en ésta práctica.

2.3. Punto 1: Hola Mundo

Creamos un nuevo servlet en nuestro proyecto. Los métodos **doGet()** y **doPost()** son los responsables de atender las peticiones (objetos de tipo request) hechas por un user agent (navegador web, cliente web, etc), procesarlas, y generar una respuesta a éste (objetos de tipo response). En la mayoría de los puntos vamos a utilizar únicamente el método doGet().

Establecemos el tipo de contenido del objeto response como HTML plano, y creamos un objeto PrintWriter. Por medio de éste, utilizando el método println(), establecemos la estructura de una página de HTML genérica, y escribimos el "Hola Mundo". El navegador interpretará esto y lo desplegará en pantalla.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
  ↪  throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<body>");
    out.println("<h2>Hello world</h2>");
    out.println("<br>");
    out.println("<a href='index.jsp'><button>Home</button></a>");
    out.println("</body>");
    out.println("</html>");
    out.close();
}
```

2.4. Punto 2: Cabeceras del protocolo HTTP

Una vez listo nuestro objeto response y nuestro objeto PrintWriter en el método doGet(), leemos los nombres las cabeceras del protocolo HTTP de la peticion, utilizando el método getHeaderNames() del objeto request, y guardándolas en un objeto Enumeration<String>.

Dentro de un ciclo, vamos revisando si el objeto Enumeration tiene elementos restantes en su colección, y por cada nombre de la cabecera atrapado, obtenemos su contenido por medio del método getHeader(nombre), enviando este como parámetro. Imprimimos ambos valores en la estructura HTML.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
  ↪  throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    Enumeration<String> e = request.getHeaderNames();
    out.println("<html>");
    out.println("<body>");
    while (e.hasMoreElements()) {
        String name = (String)e.nextElement();
        String value = request.getHeader(name);
        out.println(name + " ====> " + value);
        out.println("<br>");
    }
    out.println("<br>");
    out.println("<a href='index.jsp'><button>Home</button></a>");
```

```
15      out.println("</body>");
16      out.println("</html>");
17      out.close();
18  }
```

2.5. Punto 3: Contador Local de peticiones GET

Aquí dejaremos de lado el método doGet() y nos enfocaremos en el método **service()**. Declaramos una variable local llamada contador inicializada en cero.

Luego, verificamos que la petición recibida sea a través del método GET, y vamos incrementando nuestro contador. Luego, preparamos el response y el PrintWriter como de costumbre para desplegar el valor en el navegador.

Nota: Como cada vez que se solicita una petición GET en éste servlet se crea y se inicializa nuevamente el contador a cero, éste nunca aumentará sin importar las peticiones que se reciban. En el siguiente punto abordamos esto.

```
1  protected void service(HttpServletRequest request, HttpServletResponse response)
     ↪  throws ServletException, IOException {
2     int counter = 0;
3     if ((request.getMethod()).equals("GET")) {
4        counter++;
5        response.setContentType("text/html");
6        PrintWriter out = response.getWriter();
7        out.println("<html>");
8        out.println("<body>");
9        out.println("<h2>Number of GET Requests: " + counter + "</h2>");
10       out.println("<br>");
11       out.println("<a href='index.jsp'><button>Home</button></a>");
12       out.println("</body>");
13       out.println("</html>");
14       out.close();
15    }
16 }
```

2.6. Punto 4: Contador Global de peticiones GET

Como vimos, en el punto anterior el valor del contador permanece en uno, ya que cada vez que se invoca el método service() por medio de GET, éste se vuelve a crear continuamente e inicializar en cero.

Para solucionar esto, debemos de declarar el contador como un objeto global, afuera de todos los métodos pero dentro de la clase.

Al enviar peticiones al servlet, el contador irá aumentando.

```
1  public class Punto4 extends HttpServlet {
2     int counter = 0;
3
```

```
4     protected void service(HttpServletRequest request, HttpServletResponse
        ↪  response) throws ServletException, IOException {
5       if ((request.getMethod()).equals("GET")) {
6           counter++;
7           response.setContentType("text/html");
8           PrintWriter out = response.getWriter();
9           out.println("<html>");
10          out.println("<body>");
11          out.println("<h2>Number of GET Requests: " + counter + "</h2>");
12          out.println("<br>");
13          out.println("<a href='index.jsp'><button>Home</button></a>");
14          out.println("</body>");
15          out.println("</html>");
16          out.close();
17      }
18    }
19  }
```

2.7. Punto 5: Despliegue y registro de usuarios en una base de datos

2.7.1. Instalación de PostgreSQL y carga de la base de datos

Antes que nada debemos de instalar el gestor de base de datos PostgreSQL siguiendo las instrucciones de éste link: `https://askubuntu.com/questions/633919/how-install-postgresql-9-4`

Asignamos una contraseña al usuario postgres y creamos la base de datos **homework-6**:

```
# passwd postgres
# exit
$ su - postgres
$ pqsl
postgres =# alter user postgres with password 'postgres';
postgres =# \q
$ createdb homework-6
$ psql homework-6
```

Cargamos el siguiente script en la base de datos:

```
1  -- Tables:
2
3  create table users (id_user int4 not null, tx_login varchar(30) not null unique,
     ↪  tx_password varchar(100) not null, primary key (id_user));
4  create table role (id_role serial not null, nb_role varchar(50) not null unique,
     ↪  ds_role varchar(250) not null, st_valid bool not null, primary key
     ↪  (id_role));
5  create table account (id_account serial not null, id_role int4 not null, id_user
     ↪  int4 not null, fh_begin date not null, fh_end date, primary key
     ↪  (id_account));
6  create table person (id_person serial not null, tx_first_name varchar(30) not
     ↪  null, tx_last_name_a varchar(30) not null, tx_last_name_b varchar(30) not
     ↪  null, tx_curp varchar(18) not null unique, fh_birth date not null, primary
     ↪  key (id_person));
```

```
7  create table access (id_access int4 not null, nu_attempt int4 not null,
     ↪ fh_failed timestamp(0), fh_lock timestamp(0), primary key (id_access));
8  create table type_contact (id_type serial not null, nb_type varchar(50) not null
     ↪ unique, ds_type varchar(200) not null, st_valid bool not null, primary key
     ↪ (id_type));
9  create table person_contacto (id_persona_contacto serial not null, id_persona
     ↪ int4 not null, id_tipo int4 not null, tx_contact varchar(50), primary key
     ↪ (id_persona_contacto));
10 create table contact (id_person int4 not null, id_contact int4 not null, primary
     ↪ key (id_person, id_contact));
11 alter table account add constraint FKaccount713322 foreign key (id_role)
     ↪ references role;
12 alter table account add constraint FKaccount999795 foreign key (id_user)
     ↪ references users;
13 alter table access add constraint FKaccess801659 foreign key (id_access)
     ↪ references users;
14 alter table person_contacto add constraint FKperson_con849379 foreign key
     ↪ (id_persona) references person;
15 alter table person_contacto add constraint FKperson_con408257 foreign key
     ↪ (id_tipo) references type_contact;
16 alter table users add constraint FKusers311802 foreign key (id_user) references
     ↪ person;
17 alter table contact add constraint FKcontact249289 foreign key (id_person)
     ↪ references person;
18 alter table contact add constraint FKcontact337911 foreign key (id_contact)
     ↪ references person;
19
20 -- Inserts:
21 -- Roles
22
23 insert into role(nb_role, ds_role, st_valid) values ('Técnico','Director técnico
     ↪ de futbol',true);
24 insert into role(nb_role, ds_role, st_valid) values ('Jugador','Jugador
     ↪ profesional de futbol',true);
25
26 -- Personas
27 insert into person(tx_first_name, tx_last_name_a, tx_last_name_b, tx_curp,
     ↪ fh_birth) values ('Ricardo', 'Ferreti', 'Oliveria', 'MOCH870812HGRX00',
     ↪ to_date('01/01/1950','dd/MM/yyyy'));
28 insert into person(tx_first_name, tx_last_name_a, tx_last_name_b, tx_curp,
     ↪ fh_birth) values ('Rafael', 'Márquez', 'Álvarez', 'MOCH870812HGRX01',
     ↪ to_date('01/01/1960','dd/MM/yyyy'));
29 insert into person(tx_first_name, tx_last_name_a, tx_last_name_b, tx_curp,
     ↪ fh_birth) values ('Javier', 'Hernández', 'Balcázar', 'MOCH870812HGRX02',
     ↪ to_date('01/01/1970','dd/MM/yyyy'));
30 insert into person(tx_first_name, tx_last_name_a, tx_last_name_b, tx_curp,
     ↪ fh_birth) values ('José Andrés', 'Guardado', 'Hernández',
     ↪ 'MOCH870812HGRX03', to_date('01/01/1980','dd/MM/yyyy'));
31 insert into person(tx_first_name, tx_last_name_a, tx_last_name_b, tx_curp,
     ↪ fh_birth) values ('Cristiano Ronaldo', 'Dos Santos', 'Aveiro',
     ↪ 'MOCH870812HGRX04', to_date('01/01/1990','dd/MM/yyyy'));
32
33 -- Usuarios
34 insert into users(id_user, tx_login, tx_password) values
     ↪ (1,'tuca_ferreti@gmail.com', 'prueba123');
```

```
35 insert into users(id_user, tx_login, tx_password) values
    ↪  (2,'rafael_marquez@gmail.com', 'prueba123');
36 insert into users(id_user, tx_login, tx_password) values
    ↪  (3,'chicharito_hernandez@gmail.com', 'prueba123');
37 insert into users(id_user, tx_login, tx_password) values
    ↪  (4,'andres_guardado@gmail.com', 'prueba123');
38 insert into users(id_user, tx_login, tx_password) values
    ↪  (5,'cristiano_ronaldo@gmail.com', 'prueba123');
39
40 -- Cuentas
41 insert into account(id_role, id_user, fh_begin, fh_end) values (1, 1,
    ↪  to_date('01/01/2018','dd/MM/yyyy'), to_date('31/12/1950','dd/MM/yyyy'));
42 insert into account(id_role, id_user, fh_begin, fh_end) values (2, 2,
    ↪  to_date('01/01/2018','dd/MM/yyyy'), to_date('31/12/2018','dd/MM/yyyy'));
43 insert into account(id_role, id_user, fh_begin, fh_end) values (2, 3,
    ↪  to_date('01/01/2018','dd/MM/yyyy'), to_date('31/12/2018','dd/MM/yyyy'));
44 insert into account(id_role, id_user, fh_begin, fh_end) values (2, 4,
    ↪  to_date('01/01/2018','dd/MM/yyyy'), to_date('31/12/2018','dd/MM/yyyy'));
45 insert into account(id_role, id_user, fh_begin, fh_end) values (2, 5,
    ↪  to_date('01/01/2018','dd/MM/yyyy'), null);
46
47 -- Acceso
48 insert into access(id_access, nu_attempt, fh_failed, fh_lock) values
    ↪  (1,0,null,null);
49 insert into access(id_access, nu_attempt, fh_failed, fh_lock) values
    ↪  (2,0,null,null);
50 insert into access(id_access, nu_attempt, fh_failed, fh_lock) values
    ↪  (3,0,null,null);
51 insert into access(id_access, nu_attempt, fh_failed, fh_lock) values
    ↪  (4,0,null,null);
52 insert into access(id_access, nu_attempt, fh_failed, fh_lock) values
    ↪  (5,0,null,null);
53
54 -- Tipo contacto
55 insert into type_contact(nb_type, ds_type, st_valid) values
    ↪  ('Telephone','Telephone',true);
56 insert into type_contact(nb_type, ds_type, st_valid) values
    ↪  ('Email','Email',true);
57 insert into type_contact(nb_type, ds_type, st_valid) values ('Mobile
    ↪  phone','Mobile phone',true);
58 insert into type_contact(nb_type, ds_type, st_valid) values
    ↪  ('Facebook','Facebook',true);
59 insert into type_contact(nb_type, ds_type, st_valid) values
    ↪  ('Twitter','Twitter',true);
60
61 --Contactos
62 insert into person_contacto(id_persona, id_tipo, tx_contact) values
    ↪  (1,1,'5555555555');
63 insert into person_contacto(id_persona, id_tipo, tx_contact) values
    ↪  (2,2,'aaaa.bbbb@cccc.com');
64 insert into person_contacto(id_persona, id_tipo, tx_contact) values
    ↪  (3,3,'5555555555');
65 insert into person_contacto(id_persona, id_tipo, tx_contact) values
    ↪  (4,4,'Facebook');
```

```
66 insert into person_contacto(id_persona, id_tipo, tx_contact) values
    ↪  (5,5,'@Twitter');
```

2.7.2. Método GET. Leer registros

Nos apoyamos de una clase llamada Conexión, que es la encargada de realizar la conexión con la base de datos **homework-6**, de hacer consultas e inserciones SQL, y de cerrar la conexión.

```
1  public class Conexion {
2     String url;
3     String user;
4     String db;
5     String driver;
6     String password;
7     Connection con;
8     SimpleDateFormat sdf, sdf2;
9
10    public Conexion(String driver, String url, String db, String user, String
        ↪  password) {
11       this.driver = driver;
12       this.url = url;
13       this.db = db;
14       this.user = user;
15       this.password = password;
16       this.con = null;
17       sdf = new SimpleDateFormat("dd-MM-yyyy");
18       sdf2 = new SimpleDateFormat("yyyy-MM-dd");
19    }
20
21    public void conectarBD() {
22       try {
23             Class.forName(driver);
24             con = DriverManager.getConnection(url + db,  user, password);
25             con.setAutoCommit(false);
26         }catch (Exception e) {
27             System.out.println("Ocurrio un error : "+e.getMessage());
28             System.exit(1);
29         }
30         System.out.println("La conexión se realizo sin problemas");
31    }
32
33    public void cerrarConexion() {
34       try {
35          con.close();
36          System.out.println("Se cierra la conexion...");
37       }catch(Exception e) {
38          e.printStackTrace();
39          System.exit(1);
40       }
41    }
42 }
```

Se crea un objeto Conexión en el constructor del servlet.

Primero, se abre la conexión a la base de datos. Invocando al método mostrarUsuarios(out), enviándole como parámetro el objeto PrintWriter, con ayuda de un objeto Statement y su método executeQuery(), al cual le pasamos como parámetro nuestra consulta SQL, recibe todos los registros de la tabla **person** y los logins asociados de la tabla **users**, que son guardados en un objeto ResultSet.

Utilizando los métodos getters de éste objeto, vamos ir imprimiendo en una tabla HTML genérica (con ayuda del PrintWriter) la información de cada registro, mientras existan en el ResultSet.

Ya por último cerramos el objeto Statement y la conexión a la base de datos.

```
public void mostrarUsuarios(PrintWriter out) {
   try {
      Statement stmt = null;
      stmt = con.createStatement();
        ResultSet rsPerson = stmt.executeQuery(
              "SELECT p.*, u.tx_login " +
              "FROM person p, users u " +
              "WHERE u.id_user = p.id_person;");

        while (rsPerson.next()) {
            out.println("<tr>");
         out.println("<td>" + rsPerson.getString("tx_first_name") + "</td>");
         out.println("<td>" + rsPerson.getString("tx_last_name_a") + "</td>");
         out.println("<td>" + rsPerson.getString("tx_last_name_b") + "</td>");
         out.println("<td>" + rsPerson.getString("tx_curp") + "</td>");
         out.println("<td>" + sdf.format(rsPerson.getDate("fh_birth")) +
           ↪  "</td>");
         out.println("<td>" + rsPerson.getString("tx_login") + "</td>");
         out.println("</tr>");
        }

        rsPerson.close();
        stmt.close();
        System.out.println("Usuarios mostrados en tabla");
   }catch(Exception e) {
      e.printStackTrace();
      System.exit(1);
   }
}
```

Método doGet() del servlet:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
  ↪  throws ServletException, IOException {
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();

   out.println("<html>");
   out.println("<head>");
   out.println("<meta charset='UTF-8'>");
   out.println("<title>Table X</title>");
   out.println("</head>");
   out.println("<body>");
   out.println("<h2>Users</h2>");
```

```
12      out.println("<table border='2'>");
13      out.println("<tr>");
14      out.println("<th>First Name (job title)</th>");
15      out.println("<th>Last Name</th>");
16      out.println("<th>Second Last Name</th>");
17      out.println("<th>CURP</th>");
18      out.println("<th>Birthday</th>");
19      out.println("<th>Nickname</th>");
20      out.println("</tr>");
21
22      c.conectarBD();
23      c.mostrarUsuarios(out);
24      c.cerrarConexion();
25
26      out.println("</table>");
27      out.println("<br>");
28      out.println("<a href='punto5-2.jsp'><button>New</button></a>");
29      out.println("<a href='index.jsp'><button>Home</button></a>");
30      out.println("</body>");
31      out.println("</html>");
32      out.close();
33  }
```

2.7.3. Método POST. Insertar registros

Para éste punto creamos un formulario HTML con los campos de:

- Nombre
- Primer Apellido
- Segundo Apellido
- CURP
- Fecha de nacimiento - Cumpleaños
- Login
- Contraseña
- Verificar Contraseña

Que utiliza el método POST y envía los datos al servlet.

```
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <meta charset="UTF-8">
5           <title>New User</title>
6       </head>
7       <body>
8           <h2>New User</h2>
9           <form method="POST" action="Punto5">
10              <label>First name:</label>
11              <input type="text" name="firstName" id="firstName" required>
12              <br>
13              <label>Last name:</label>
14              <input type="text" name="lastName" id="lastName" required>
15              <br>
```

```
16          <label>Second last name:</label>
17          <input type="text" name="secondName" id="secondName" required>
18          <br>
19          <label>CURP:</label>
20          <input type="text" name="curp" id="curp" required>
21          <br>
22          <label>Birthday:</label>
23          <input type="date" name="birthday" id="birthday" required>
24          <br>
25          <label>Login:</label>
26          <input type="email" name="login" id="login" required>
27          <br>
28          <label>Password:</label>
29          <input type="password" name="password" id="password" required>
30          <br>
31          <label>Confirm Password:</label>
32          <input type="password" name="confirmPassword" id="confirmPassword"
              ↪  required>
33          <br>
34          <a href="Punto5"><button type="button">Cancel</button></a>
35          <input type="submit" value="Save">
36      </form>
37   </body>
38 </html>
```

En el método doPost(), primero obtenemos los valores de los parámetros ingresados en el formulario.

Verificamos que ambas contraseñas ingresadas sean las mismas, si lo son, abrimos la conexión a la base de datos e invocamos al método registrarUsuarios() del objeto Conexión, enviando como parámetros los valores obtenidos del formulario.

Aquí creamos un objeto Statement otra vez, y le enviamos como parámetro la instrucción SQL para ingresar estos valores recibidos correspondientes en la tabla **person** y **users**, y ejecutamos esta con el método executeUpdate().

Cerramos el objeto Statement y la conexión a la base de datos.

NOTA: Cabe mencionar que todos los métodos de la clase Conexión están dentro de bloques try-catch, para manejar las excepciones y errores que pudiesen surgir.

```
1  public void registrarUsuarios(String firstName, String lastName, String
     ↪  secondName, String curp, String birthday, String login, String password) {
2     try {
3        Statement stmt2 = null;
4        boolean registrar = true;
5        stmt2 = con.createStatement();
6          ResultSet rsLogins = stmt2.executeQuery(
7                "SELECT tx_login " +
8                "FROM users u;");
9
10         while(rsLogins.next()) {
11            if(login.equals(rsLogins.getString("tx_login"))) {
12               registrar = false;
13            }
```

```
14          }
15
16          if(registrar) {
17             Statement stmt = null;
18           int nextId = 1;
19           stmt = con.createStatement();
20           Date birth = new Date((sdf2.parse(birthday)).getTime());
21
22           String addPerson =
23                 "INSERT INTO person (tx_first_name, tx_last_name_a,
                    ↪  tx_last_name_b, tx_curp, fh_birth) " +
24                 "VALUES ('" + firstName + "', '" + lastName + "', '" + secondName
                    ↪  + "', '" + curp + "', '" + birth + "');";
25
26           stmt.executeUpdate(addPerson);
27
28           String getId = "SELECT id_person " +
29                   "FROM person " +
30                   "WHERE tx_curp = '" + curp + "';";
31
32           ResultSet rsId = stmt.executeQuery(getId);
33
34           if(rsId.next())
35              nextId = rsId.getInt("id_person");
36
37           String addUser =
38                 "INSERT INTO users (id_user, tx_login, tx_password) " +
39                 "VALUES (" + nextId + ", '" + login + "', '" + password + "');";
40
41           stmt.executeUpdate(addUser);
42             stmt.close();
43             con.commit();
44             System.out.println("Registro insertado correctamente");
45          }
46          else {
47             System.out.println("El usuario ya existe. No se registrará la
                  ↪  información.");
48             stmt2.close();
49          }
50       }catch(Exception e) {
51          e.printStackTrace();
52          System.exit(1);
53       }
54    }
```

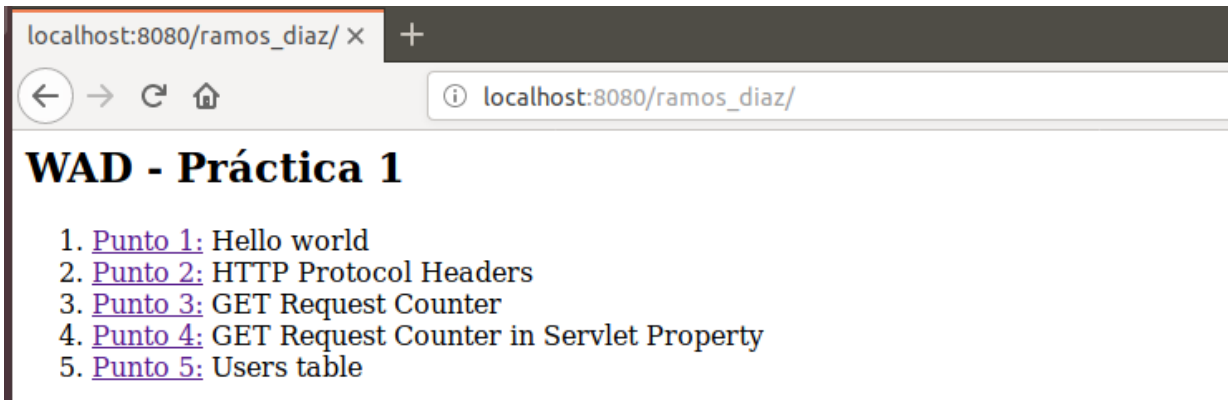
Por último y muy importante, invocamos al método doGet() al final y le enviamos como parámetros los objetos response y request, para desplegar en el navegador el nuevo registro de la base de datos junto a los anteriores existentes.

Método doPost() del servlet:

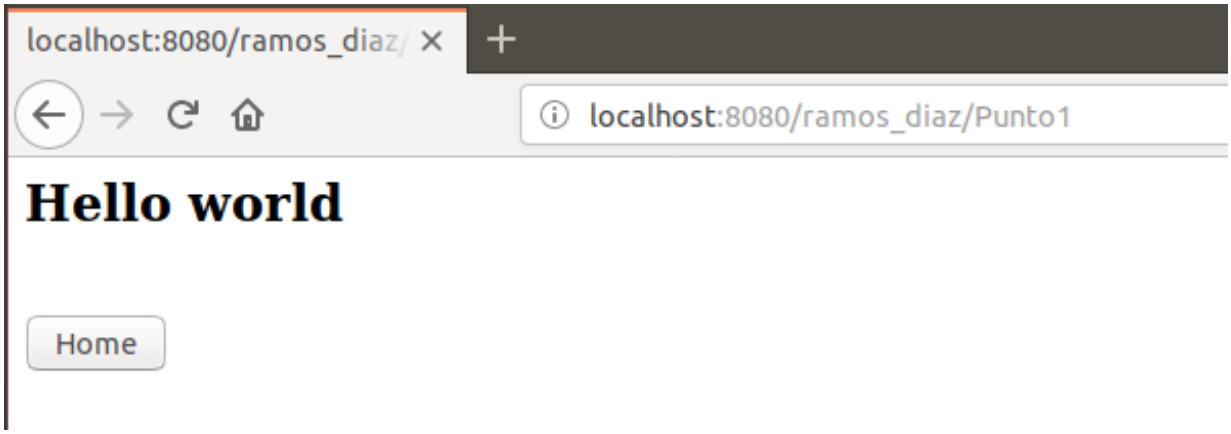
```
1  protected void doPost(HttpServletRequest request, HttpServletResponse response)
     ↪  throws ServletException, IOException {
2     String firstName = request.getParameter("firstName");
```

```
3       String lastName = request.getParameter("lastName");
4       String secondName = request.getParameter("secondName");
5       String curp = request.getParameter("curp");
6       String birthday = request.getParameter("birthday");
7       String login = request.getParameter("login");
8       String password = request.getParameter("password");
9       String confirmPassword = request.getParameter("confirmPassword");
10
11      if(confirmPassword.equals(password)) {
12          c.conectarBD();
13          c.registrarUsuarios(firstName, lastName, secondName, curp, birthday,
              ↪ login, confirmPassword);
14          c.cerrarConexion();
15      }
16      else
17          System.out.println("Las contraseñas no coinciden. Intente de nuevo");
18
19      doGet(request, response);
20  }
```

3. Pruebas



3.1. Punto 1: Hola Mundo

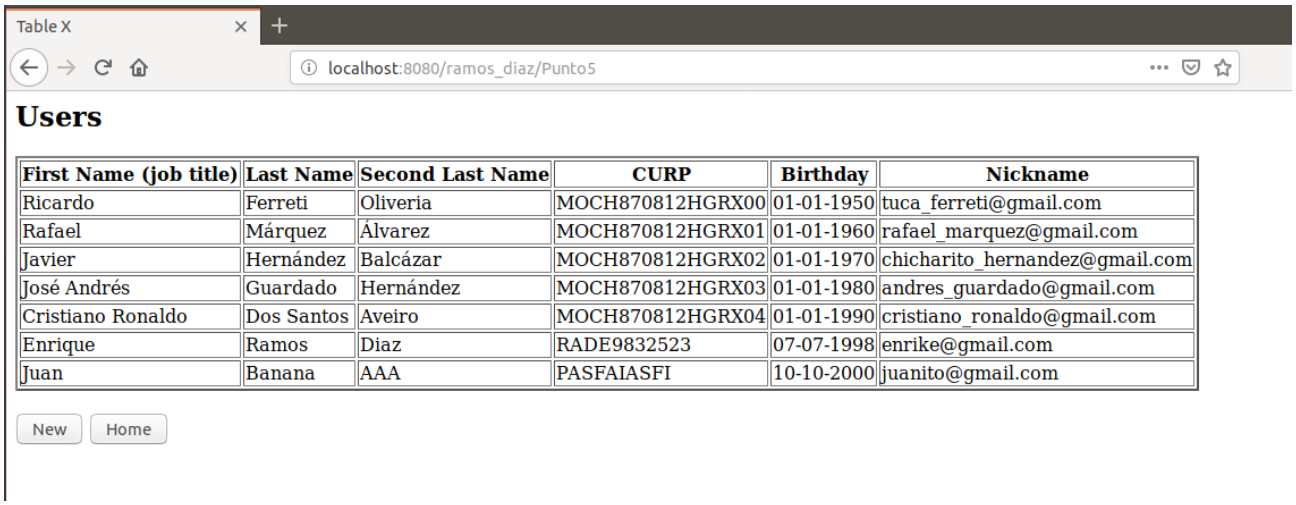


3.2. Punto 2: Cabeceras del protocolo HTTP

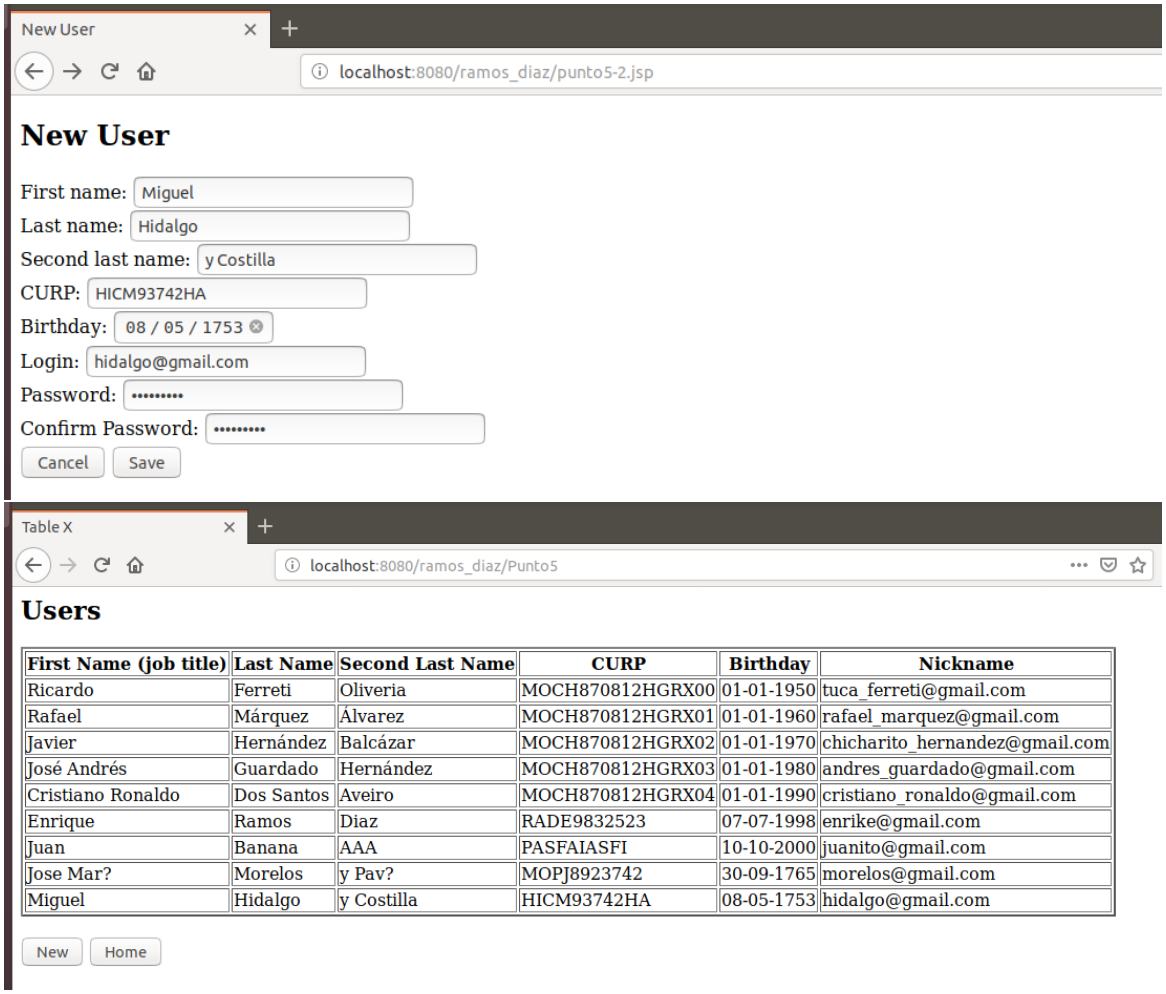
3.3. Punto 3: Contador de peticiones GET

3.4. Punto 4: Contador Global de peticiones GET

3.5. Punto 5-1: Despliegue de usuarios en una tabla



3.6. Punto 5-2: Formulario de registro y despliegue de nuevos usuarios



4. Conclusiones

Al codificar un servlet y tener dudas de cómo funcionan sus objetos, métodos o métodos de sus objetos, una buena práctica es ir a la documentación de Java y leer las especificaciones de estos.

En este práctica, por ejemplo, se trabajo más que nada con los objetos request y response, y algunos de los métodos que se revisaron fueron: getParameters(), getWriter() y setContentType().

Más que nada esta práctica nos ayuda a familiarizarnos con la estructura que compone a un servlet, el funcionamiento de sus dos principales métodos doGet() y doPost(), su comportamiento, características y diferencias. Esto con el fin de saber cuál método usar en diversas situaciones que puedan presentarse al trabajar con proyectos reales.

Hay que tener en cuenta todas las dependencias que vamos a necesitar si es que requerimos de librerías externas, como es el caso del conector al gestor de base de datos PostgreSQL, y el paquete javax.sevlet, que es básico e indispensable para que los servlets funcionen.

El procesamiento de la información de las peticiones es un aspecto relativamente ”ajeno” a los servlets, pues este se realiza con lenguaje Java puro, sin involucrar al apartado web en él; el navegador web únicamente nos servirá como entrada y salida de información (request y response respectivamente).

5. Bibliografía

[1] (2019) H. F. Montes Casiano. WEB APPLICATION DEVELOPMENT 2018 - 2019 2. Accessed february 2019. [Online]. Available: `http://www.comunidad.escom.ipn.mx/hermesm/cursos/WAD20192.xhtml`