Manejo de sesiones

¡Comencemos!

¿Qué aprenderás?

- Diferenciar cookie y sesión.
- Aplicar variables de sesión en entorno de prueba.

Introducción

Las cookies nacieron en 1994, como una nueva funcionalidad del navegador Netscape. La compañía de telecomunicaciones MCI estaba desarrollando una aplicación de comercio electrónico y les solicitó una solución técnica para recordar los productos que los usuarios añadían al carrito de compras, mientras seguían navegando por el catálogo, sin la necesidad de almacenar datos en los servidores del proveedor.

El concepto de cookie se compara a la analogía de los 'rastros de galleta': cuando visitamos un sitio, dejamos un rastro de lo que hacemos, una especie de huellas digitales o 'migajas' que permite que los navegadores recuerden algunas de nuestras preferencias.

¿Qué son las cookies?

- Son pequeños archivos de datos (texto) creados y enviados por un sitio web y almacenados en el navegador del usuario. De esta forma, el sitio web puede consultar la actividad previa del navegador y detectar las páginas visitadas por el usuario.
- Está formada por una clave que le da nombre y un valor asociado, que se puede crear, modificar o borrar tanto en el cliente como en el servidor y que una vez creada se envía en cada petición HTTP, de cada archivo, como parámetro de este protocolo.

Es importante mencionar que una cookie no puede borrar ni leer información del ordenador de los internautas, ni tampoco identifica a una persona, sino que reconoce una combinación de computador-navegador-usuario.

Si entramos a nuestro navegador, podremos ver aquellas cookies y su información almacenada por los sitios que frecuentemente visitamos.
Debemos ir a *configuración > configuración del sitio > datos de sitios y cookies*.

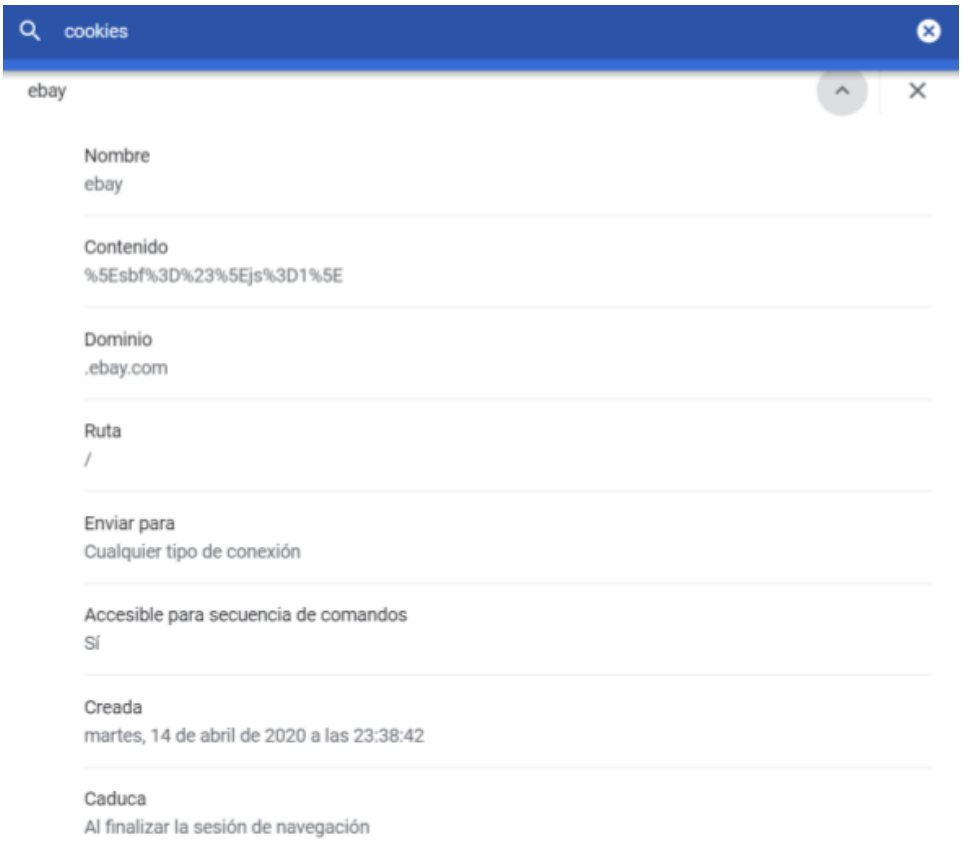

Imagen 1. Cookies en el navegador.
Fuente: Desafío Latam

¿Cuál es la utilidad de las cookies?

El protocolo HTTP técnicamente no mantiene el estado en la aplicación, esto quiere decir que olvidará los datos de los request y de los response a medida que el usuario interactúe con la página. Esto es un grave problema teniendo en cuenta la naturaleza de un sistema web, ya que no tendría sentido estar preguntando al usuario a cada rato como se llama, o que parámetros envió en los formularios.

Es por esto que las aplicaciones web tienen que utilizar una serie de técnicas que permitan mantener el estado entre cliente y servidor en el transcurso del tiempo. Cuando hablamos de "mantener el estado" nos referimos a guardar los datos del usuario y su información relacionada, para que independiente de la navegación del sitio, se siga reconociendo.

Algunas ventajas y desventajas en el uso de cookies son:

Ventajas	Desventajas
Navegación rápida, sencilla y personalizada para el usuario.	Solo pueden almacenar texto.
Conocer los hábitos de navegación del usuario.	Son dependientes del navegador, esto significa que si el cliente deshabilita las cookies, ya no será posible utilizarlas.
Recordar preferencias de uso.	No se puede almacenar más de 4 Kb de información.

Tabla 1. Ventajas y desventajas de las cookies
Fuente: Desafío Latam

Aspectos legales en el uso de cookies

Si bien las cookies pueden dar una gran visión de las actividades y preferencias de los usuarios, también existe el riesgo de identificarlas sin su consentimiento explícito.

Algunos de los problemas de privacidad que se han observado en relación a las cookies tiene relación con qué se está registrando y quién lo está registrando, por lo que en 2018 la Comisión Europea hizo efectivo el Reglamento General de Protección de Datos (RGPD) que controla la forma en la que empresas y otras organizaciones tratan datos personales: **https://ec.europa.eu/justice/smedataprotect/index_en.htm.**

Imagen 2. RGPD.
Fuente: Desafío Latam

En este reglamento se establecen, entre otras cosas, políticas relacionadas al uso de cookies. Un sitio debe contar con los siguientes requisitos:

- Política de cookies transparente.
- Resumen y responsabilidad por las cookies en la web.
- Solicitud de consentimiento a través de un ícono informativo.
- Posibilidad de retirar el consentimiento en cualquier momento.
- Renovación de consentimiento.
- Los consentimientos deben ser almacenados.

Cookies y sesiones

Una sesión es una serie de comunicaciones entre cliente y servidor, en la que se realiza un intercambio de información. Su ciclo de vida comienza cuando un usuario se conecta por primera vez a un sitio web y su finalización ocurre en uno de los siguientes casos:

- Cuando se abandona el sitio web.
- Después de un tiempo de inactividad.
- Cuando se cierra o reinicia el navegador.

La principal diferencia entre las sesiones y las cookies es que la información almacenada con una sesión se guarda en el lado del servidor y con una cookie, en el cliente. Además, las sesiones permiten almacenar mayor información, pero su ciclo de vida es mucho menor.

Sesiones y servlets

Los servlets implementan su propio mecanismo de manejo de sesiones. La imagen siguiente gráfica como trabaja la sesión la tecnología java con servlets:

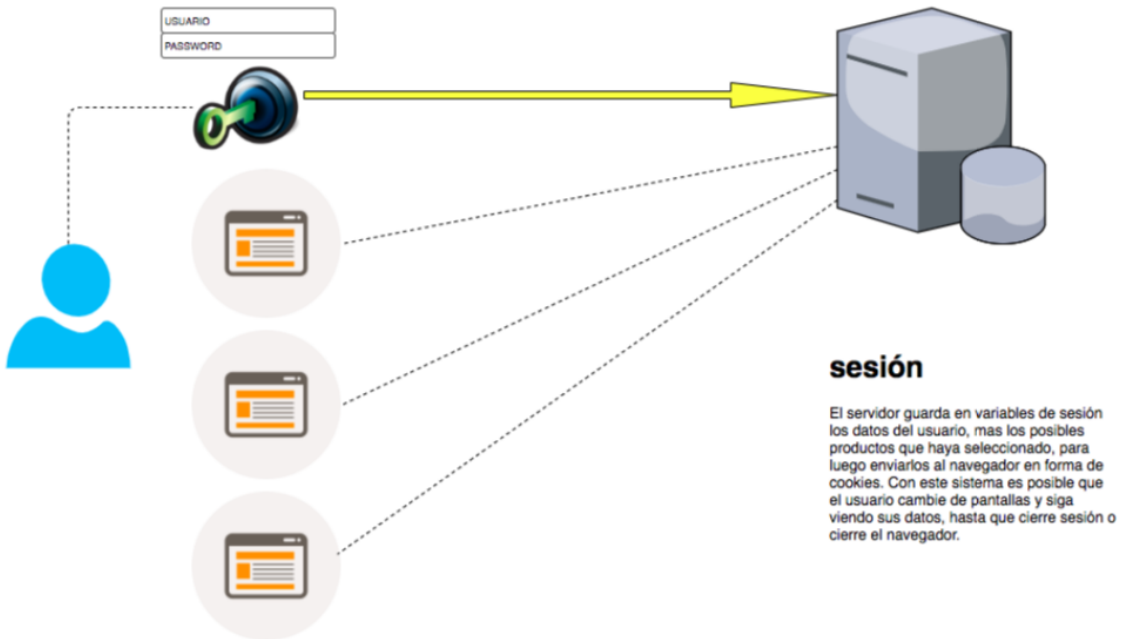

Imagen 3. Uso de sesiones.
Fuente: Desafío Latam

La biblioteca de java nos provee de las clases HttpSession, la cual nos permite manejar y almacenar variables de sesión en torno a las acciones del usuario. Cuando hablamos de variables de sesión, estamos diciendo que esta clase puede almacenar cualquier tipo de elemento, incluidas las clases de la aplicación (entidades de negocio).

Vamos a analizar un ejemplo sencillo del manejo de sesión y el almacenamiento de variables asociados a un usuario.

La siguiente clase es solamente una entidad de negocio de tipo *EntradasConcierto*, la cual representa la venta de tickets para un evento musical y nos servirá para simular la compra:

```
public class EntradasConcierto{
  private String titularEntrada;
  private String nombreArtista;
  private String ubicacion;
  private Date fechaConcierto;
  public String getTitularEntrada() {
        return titularEntrada;
  }
  public void setTitularEntrada(String titularEntrada) {
        this.titularEntrada = titularEntrada;
  }
  public String getNombreArtista() {
        return nombreArtista;
  }
  public void setNombreArtista(String nombreArtista) {
        this.nombreArtista = nombreArtista;
  }
  public String getUbicacion() {
        return ubicacion;
  }
  public void setUbicacion(String ubicacion) {
        this.ubicacion = ubicacion;
  }
  public Date getFechaConcierto() {
        return fechaConcierto;
  }
  public void setFechaConcierto(Date fechaConcierto) {
        this.fechaConcierto = fechaConcierto;
  }
}
```

Esta clase representa a la entidad de negocio que simula una entrada al concierto. Cuenta con un constructor vacío por defecto y los parámetros de *titularEntrada*, *nombreArtista*, *ubicacion* y *fechaConcierto*.

A continuación, vamos a crear el primer servlet, que recibirá el nombre de la persona que comprará el ticket.

```
    @WebServlet("/creaSesion")
    public class EjemploSesion extends HttpServlet {
      private static final long serialVersionUID = 1L;
      protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws IOException{
            String usuario = request.getParameter("nombre");
            EntradaConcierto entrada = new EntradasConcierto();
            HttpSession sesion = request.getSession(true);
            PrintWriter writer = response.getWriter();
            entrada.setNombreArtista("Iron Maiden");
            entrada.setFechaConcierto(new Date());
            entrada.setUbicacion("Cancha");
            sesion.setAttribute("datosCompra", entrada);
            writer.println("<html><body>");
            writer.println("<h1>Entradas para
"+entradas.getNombreArtista()+"</h1>");
            writer.println("<h1>A nombre de
"+entradas.getTitularEntrada()+"</h1>");
            writer.println("</body></html>");
      }
    }
```

La línea sombreada cuenta con la implementación de la clase HttpSession con su objeto *session*. Este objeto recibe la sesión que está contenida en el request mediante el método *getSession(true)*.

Se utiliza el objeto previamente creado de nombre sesión y, mediante el método *setAttribute()*, enviamos los datos de compra y el objeto entrada que contiene los datos del registro del evento. Con esto la sesión ya contiene el objeto entrada y los puede mantener activos.

Al llamar al servlet mediante la dirección:

http://localhost:8080/proyectoSesiones/creaSesion?nombre=Juan%20Soto

En el servlet ocurre lo siguiente:

1. El *servlet* recibe el request mediante el parámetro *HttpServletRequest request*.
2. Obtenemos el valor que fue enviado por el usuario, correspondiente al dueño de la entrada mediante el método *request.getParameter("usuario")* y lo guardamos en la variable usuario.
3. Creamos un objeto entrada de la clase *EntradasConcierto*.
4. Generamos la sesión mediante el request que llega desde el cliente.
5. Creamos atributos para dar valor al objeto entrada.

Finalmente guardamos el objeto entrada en el objeto de sesión mediante el método *setAtribute*.

La pantalla nos mostrará un mensaje con el dueño de las entradas, que nosotros mismos generamos al llamar al *servlet creaSession*:

Por el momento, solo creamos una variable de sesión en la cual guardamos un objeto de tipo *EntradaConcierto* que contiene el nombre del propietario más datos propios de la *entrada*. *Ahora* es tiempo de crear un segundo *servlet*, que al ejecutarlo va a rescatar de la sesión los valores previamente guardados. La idea es que llamemos al próximo *servlet* desde otro navegador para comprobar que realmente estamos guardando valores en la sesión.

El código es el siguiente:

Este *servlet* se encarga de capturar los datos de la sesión que guardamos en la pantalla anterior. Lo hace utilizando la clase *HttpSession*:

```
    @WebServlet("/obtieneSesion")
    public class ObtieneSesion extends HttpServlet {
      protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws IOException{
            HttpSession misesion = request.getSession();
            EntradasConcierto entradas =
(EntadasConcierto)misesion.getAttribute("datosCompra");
            PrintWriter writer = response.getWriter();
            writer.println("<html><body>");
            writer.println("<h1>Entradas para
"+entradas.getNombreArtista()+"</h1>");
            writer.println("<h1>A nombre de
"+entradas.getTitularEntrada()+"</h1>");
            writer.println("</body></html>");
      }
    }
```

La variable *misesion* ahora contiene los datos de la entrada que guardamos en el primer servlet. Ahora solo es cuestión de crear un objeto de tipo *EntradasConcierto* para luego obtener el objeto previamente guardado en la sesión.

Tenemos los datos contenidos en el objeto entradas y desde aquí podemos obtener los valores que estaban guardados en la sesión, para luego, mediante el método *writer.println*, **dibujarlos** en pantalla.

Es tiempo de llamar al *servlet obtieneSesion* con la *url*:

- http://localhost:8080/proyectoSesiones/obtieneSesion

Veremos que logró recuperar los valores ingresados en *servlet EjemploSesion*.

Imagen 4. *EjemploSesion*.
Fuente: Desafío Latam

En resumen, en el *servlet EjemploSesion* enviamos el nombre de la persona que está comprando la entrada:

http://localhost:8080/proyectoSesiones/creaSesion?nombre=Juan%20Soto

Luego desde otro navegador, llamamos al *servlet ObtieneSesion* encargado de recuperar los valores que guardamos desde *EjemploSesion*.

http://localhost:8080/proyectoSesiones/obtieneSesion

Trabajo Práctico con Sesiones

Aplicación de variables de sesión en entorno de prueba

Para llevar a cabo la implementación de sesiones, vamos a crear un pequeño sitio, en donde tendremos un *login* que pedirá usuario y password mediante un formulario en un *JSP*.

Tendremos un *servlet* que se encargará de procesar la petición y realizar una validación básica, la cual si es afirmativa redirigirá a un segundo *JSP* con el nombre del usuario logueado.

Esta pantalla *JSP* también tendrá un botón de cerrar sesión, que se encargará de eliminar la sesión del sistema y redirigir a la pantalla de *login*.

Con este ejemplo podremos ver como trabajan los *JSP* junto a los *Servlets* y las sesiones.

Este proyecto cuenta con *bootstrap* y *css*, los cuales están ya configurados en el head de los *jsp*, solo debes pegar los archivos en la carpeta css ubicada en el directorio *WebContent*, tal como se muestra en la imagen:

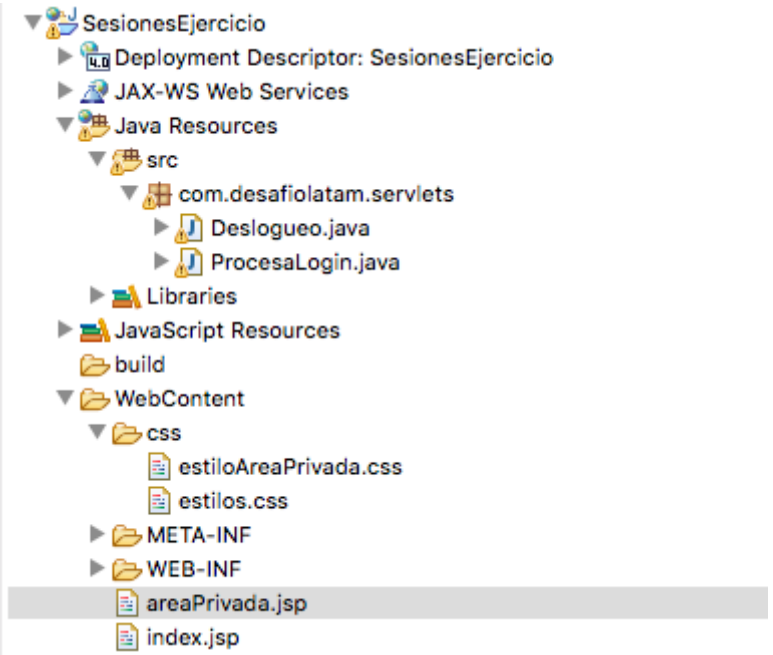

Imagen 5. Estructura de carpetas.
Fuente: Desafío Latam

La pantalla de login tendrá este formato:

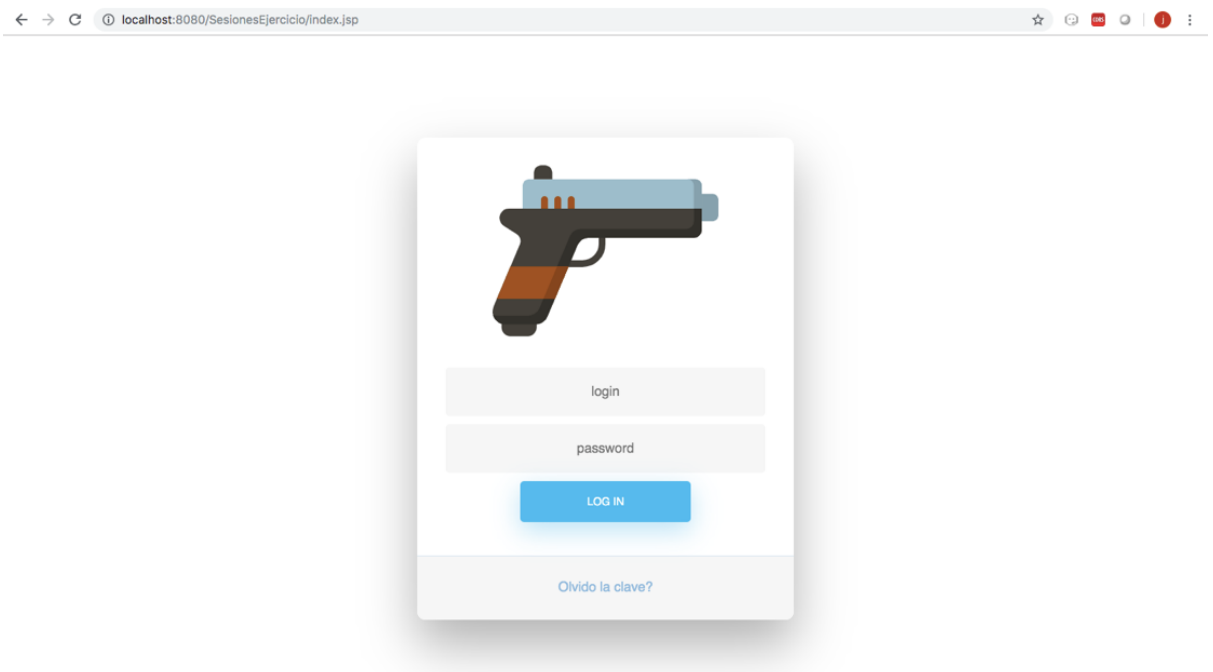

Imagen 6. Vista Login.
Fuente: Desafío Latam

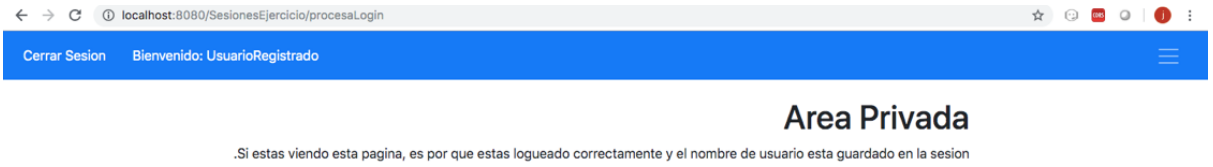

Imagen 7. Login correcto.
Fuente: Desafío Latam

Primero veamos el código del servlet de procesamiento:

```
    @WebServlet("/procesaLogin")
    public class ProcesaLogin extends HttpServlet {
      private final String LOGIN = "UsuarioRegistrado";
      private final String PASS = "admin";

      protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws IOException, ServletException{
            String requestLogin = "";
            String requestPass = "";
            //Generamos variable out para crear código HTML
            PrintWriter writer = response.getWriter();
            //Obtenemos valores del request desde el formulario
            requestLogin = request.getParameter("login");
            requestPass = request.getParameter("pass");

            //Comparamos los valores enviados desde el formulario, si
alguno de ellos
            //no coincide, generamos una alerta con un mensaje, mediante
la variable out
            if(!LOGIN.contentEquals(requestLogin)) ||
!PASS.contentEquals(requestPass){
                  out.println("<script type=\"text/javascript\">");
                  out.println("alert('Usuario o password
incorrecto');");
                  out.println("location='index.jsp';");
                  out.println("</script>");
            }else {
                  //creamos la sesion si el usuario existe
                  HttpSession sesionUsuario = request.getSession(true);
                  sesionUsuario.setAttribute("Nombre", requestLogin);
                  RequestDispatcher rd =
request.getRequestDispatcher("areaPrivada.jsp");
                  rd.forward(request, response);
            }
      }
    }
```

Este *servlet* en términos prácticos se ubica en lo que se conoce como la capa de negocio, y se encarga de obtener los datos que el usuario ingresó por el *jsp* para validar y dar acceso a las siguientes páginas, además de generar una variable de sesión que se mantendrá en todo momento hasta que el usuario cierre la sesión.

El formulario que recibe los datos del usuario se llama *login.jsp*, y consta de un formulario sencillo de *usuario* y *password* más un botón que gatilla la llamada al *servlet*.

```
    <%@page language="java" contentType="text/html"
pageEncoding="UTF-8"%>
    <!DOCTYPE html>
    <html>
    <head>
      <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
      <title>Login Servlet</title>
      <link
href="//maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
rel="stylesheet" id="bootstrap-css">
      <link href="css/estilos.css" rel="stylesheet">
      <script
src="//maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"></sc
ript>
      <script
src="//cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></scri
pt>
    </head>
    <body>
      <div class="wrapper fadeInDown">
        <div id="formContent">
          <div class="fadeIn first">
            <img
src="https://image.flaticon.com/icons/svg/1548/1548286.svg" id="icon"
alt="User Icon" />
          </div>
          <form action="procesaLogin" method="POST">
            <input type="text" id="login" class="fadeIn second"
name="login" placeholder="login">
            <input type="text" id="password" class="fadeIn third"
name="pass" placeholder="password">
            <input type="submit" class="fadeIn fourth" value="Log In">
          </form>

          <div id="formFooter">
```

```
            <a class="underlineHover" href="#">Olvidó la clave?</a>
          </div>
        </div>
      </div>
    </body>
    </html>
```

Para términos estéticos se agrega la librería de *Bootstrap*. Para más detalles de este *framework* visitar su página *Bootstrap*.

Notar que en el formulario existe la etiqueta *action* que tiene como valor la palabra *procesalogin*. Ese es el nombre del *servlet* que recibe los datos que el usuario ingresa en el formulario *(@WebServlet("/procesaLogin"))*.

Para continuar con el flujo, necesitamos otro archivo *java* que se encargue de cerrar la sesión que el usuario inició. Para esto, generamos otro *servlet* que rescatará desde la sesión la variable que contiene el nombre del usuario logueado. Este *servlet* dispondrá del nombre para mostrarlo en el *jsp* de acceso privado (no se puede acceder a él si no se está logueado).

```
    @WebServlet("/logout")
    public class Deslogueo extends HttpServlet {
      protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws IOException, ServletException{
            //Se crea variable de sesion y se recibe desde el request
            HttpSession sesion = request.getSession();
            PrintWriter out = response.getWriter();
            out.println("<script type=\"text/javascript\">");
            out.println("alert('Ha cerrado sesion correctamente');");
            out.println("</script>");
            //el médito invalidate destruye la sesion
            sesion.invalidate();
            //el metodo sendRedirect redirige al formulario de ingreso
            response.sendRedirect("index.jsp");
            return;
      }
    }
```

El *servlet* es sencillo de entender. Se genera una variable de tipo *HttpSession* para guardar en él la sesión que está contenida en el request. Luego solamente hay que cerrar la sesión con el método *invalidate()*. Para informar al usuario el cierre de sesión se genera una alerta

mediante el objeto de tipo *PrintWriter*. Al final la secuencia termina cuando se redirige al usuario a la página *index.jsp*.

A continuación se muestra el *jsp* de nombre *areaPrivada.jsp*. Este código tiene más *html*, que por el momento debemos obviar.

```
    <%@page language="java" contentType="text/html"
pageEncoding="UTF-8"%>
    <!DOCTYPE html>
    <html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Area Privada</title>
    <link

href="//maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
      rel="stylesheet" id="bootstrap-css">
    <link href="css/estilos.css" rel="stylesheet">
    <script

src="//maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"></sc
ript>
    <script

src="//cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></scri
pt>
    </head>
    <body>
      <%
            HttpSession misesion = request.getSession();
        String nombre;
        nombre = (String)misesion.getAttribute("Nombre");
      %>
      <nav class="navbar navbar-expand navbar-dar bg-primary">
            <a href="#menu-toggler" id="menu-toggle"
class="navbar-brand"><span
                  class="navbar-toggler-icon"></span></a>
            <button class="navbar-toggler" type="button"
data-toggle="collapse"
                  data-target="#navbarsExample02"
aria-controls="navbarExample02"
                  aria-expanded="false" aria-label="Toggle navigation">
                  <span class="navbar-toggler-icon"></span>
            </button>
```

```
            <div class="collapse navbar-collapse" id="navbarsExample02">
                  <ul class="navbar-nav mr-auto">
                        <li class="nav-item active"><a class="nav-link"
href="#">Bienvenido:
                                    <% nombre %><span
class="sr-only">(current)</span>
                        </a></li>
                        <li class="nav-item active"><a class="nav-link"
href="logout">Cerrar
                                    Sesion<span
class="sr-only">(current)</span>
                        </a></li>
                  </ul>
                  <form class="form-inline my-2 my-md-0"></form>
            </div>
      </nav>
      <div id="wrapper" class="toggled">
            <!--  Page Content -->
            <div id="page-content-wrapper">
                  <div class="container-fluid">
                        <h1>Area Privada</h1>
                        <p>Si estás viendo esta página, es porque estás
logueado
                              correctamente y el nombre de usuario está
guardado en la sesión.</p>
                  </div>
            </div>
      </div>
      <script src="js/jquery.min.js"></script>
      <script src="js/bootstrap.bundle.min.js"></script>
      <script>
            $(function() {
                  $("#menu-toggle").click(function(e) {
                        e.preventDefault();
                        $("#wrapper").toggleClass("toggled");
                  });

                  $(window).resize(function(e) {
                        if ($(window).width() <= 768) {
                              $("#wrapper").removeClass("toggled");

                        } else {
                              $("#wrapper").addClass("toggled");
                        }
```

```
                });
            });
        </script>
    </body>
    </html>
```

Importante: Después de la etiqueta **<body>** vemos los tags **<% %>** en los cuales se ejecuta el código *java*.

```
    <%
      HttpSession misesion = request.getSession();
      String nombre;
      nombre = (String) misesion.getAttribute("Nombre");
    %>
```

Generamos un objeto de tipo *HttpSession*, que recibe desde el request la sesión generada. Creamos una variable nombre de tipo *String*, para luego guardar en ella el valor del atributo que está guardado en la sesión. Con esto tenemos el nombre de usuario guardado y listo para ser usado; en este caso lo usaremos solo para mostrarlo en la pantalla.

Tenemos el valor de la sesión en la variable nombre y, como queremos que se visualice en la pantalla, debemos mostrarla mediante las etiquetas jsp:

```
    Bienvenido: <%= nombre %>
```

Con esa sentencia el nombre se despliega al lado del texto bienvenido.

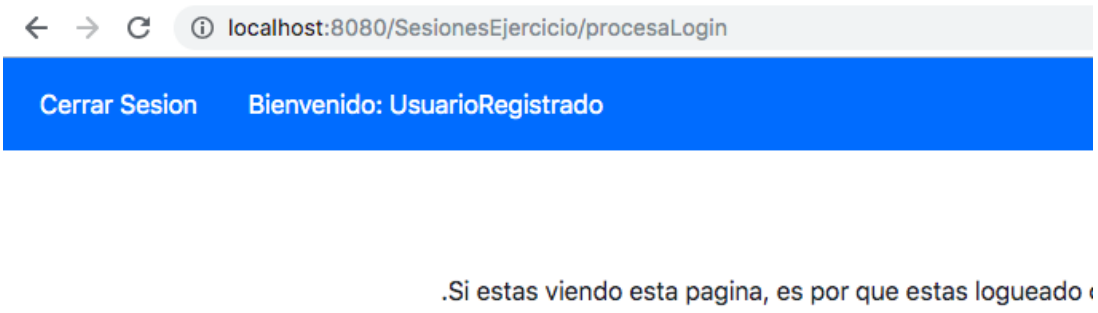

Imagen 8. Inicio de sesión.
Fuente: Desafío Latam