Portada

**Índice**

[Práctica 1: Métodos GET y POST 3](#_Toc35823487)

[Resultados 4](#_Toc35823488)

[Conclusiones 5](#_Toc35823489)

[Práctica 2: Formulario con servlet 6](#_Toc35823490)

[Resultados 7](#_Toc35823491)

[Conclusiones 10](#_Toc35823492)

[Práctica 3: Registro de laptops prestadas a maestros 11](#_Toc35823493)

[Resultados 14](#_Toc35823494)

[Conclusiones 15](#_Toc35823495)

[Práctica 4: Ejemplo Cookies 16](#_Toc35823496)

[Resultados 19](#_Toc35823497)

[Conclusiones 20](#_Toc35823498)

[Práctica 5: Sesiones y filtros 21](#_Toc35823499)

[Desarrollo 21](#_Toc35823500)

[Sesiones 21](#_Toc35823501)

[Filtros 26](#_Toc35823502)

[Resultados 27](#_Toc35823503)

[Sesiones 27](#_Toc35823504)

[Filtros 29](#_Toc35823505)

[Conclusiones 29](#_Toc35823506)

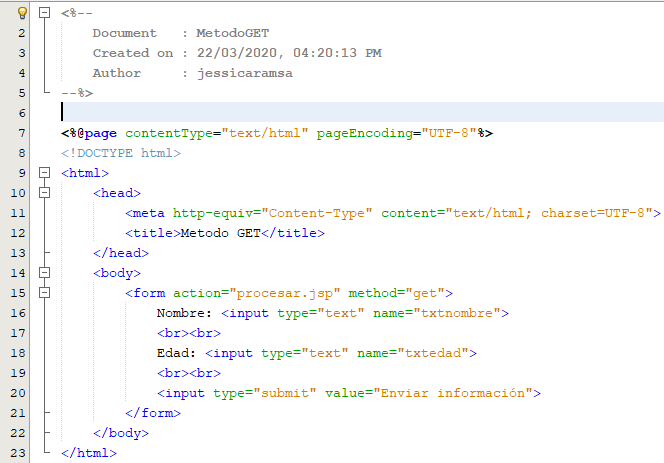
[Práctica 6: Uso de scriptlets, declaraciones, expresiones y clases. 30](#_Toc35823507)

[Resultados 37](#_Toc35823508)

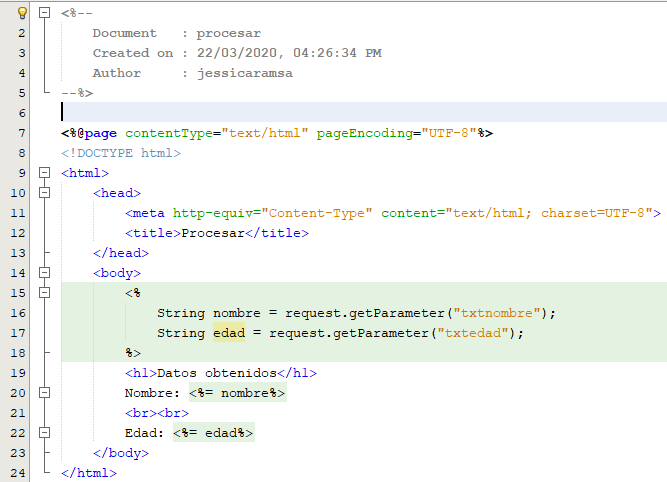
[Conclusiones 39](#_Toc35823509)

# Práctica 1: Métodos GET y POST

Para demostrar cómo se comportan ambos métodos tendremos un formulario simple que nos permitirá alternar entre un método y otro, la única diferencia radica en el método que se utiliza en el formulario *get o post*.



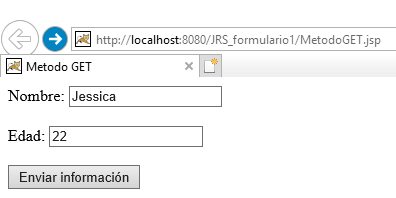
Para la muestra de los datos, contaremos con un jsp en común el cual se llama *procesar*, cabe mencionar que este JSP no hace distinción entre cual método se está utilizando para mandarlo llamar.



## Resultados

**Método GET**

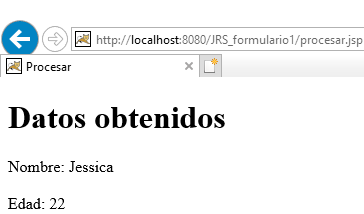
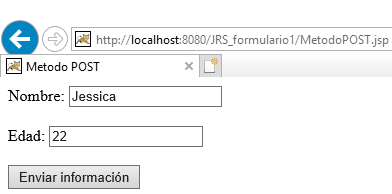
Con este método se observa que los parámetros capturados se envían mediante la URL, visibles para todos.





**Método POST**

Cambiando el método por POST vemos la diferencia que los datos capturados ya no son visibles para el usuario.



## Conclusiones

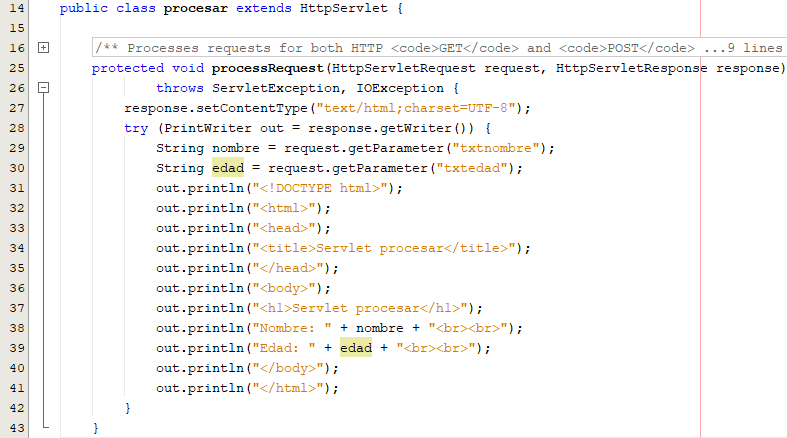
El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET solo deben recuperar datos.

En cambio, el método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.

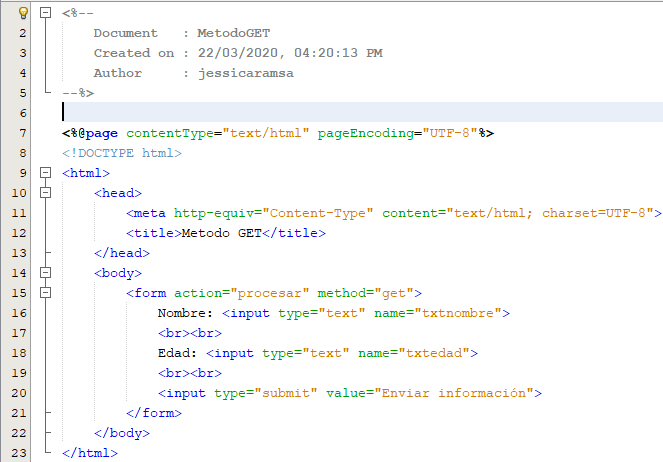
De acuerdo al método HTTP que elijamos para tratar la información será la forma en como enviaremos o recibiremos la información del servidor.

# Práctica 2: Formulario con servlet

Tendremos los mismos archivos que en la práctica pasada con la diferencia de que ahora reemplazaremos el JSP *procesar* por un servlet que contendrá la lógica para visualizar los datos capturados en los JSP del método GET y POST.



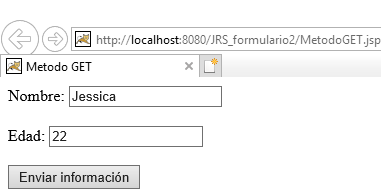
Cambiaremos el *action* de los JSP para que utilicen el servlet que acabamos de crear.



## Resultados

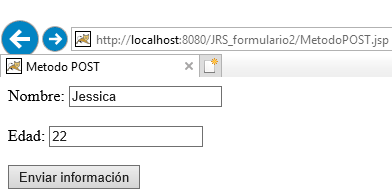
Vemos como para ambos casos el formulario sigue funcionando correctamente.

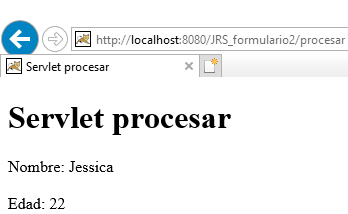
**Método GET**



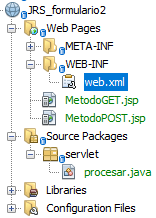


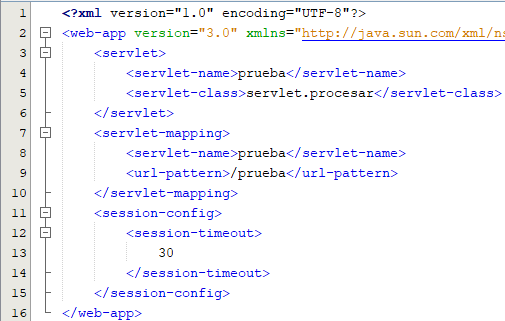
**Método POST**



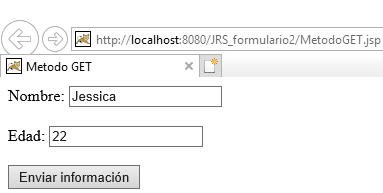


Los servlets utilizan un nombre que puede ser utilizado como *action* en los formularios de los JSP, pero si queremos cambiarlo por otro nombre debemos ir a editar el archivo *web.xml* de la carpeta *WEB-INF* que se encuentra a su vez en la carpeta *Web Pages* de nuestro directorio principal. Cambiamos en todos los lugares que diga procesar por prueba para cambiar el nombre que utilizará nuestro servlet, exceptuando la etiqueta *servlet-class* la cual indica el origen de nuestro archivo servlet.





Observemos que la URL ha cambiado por el nombre *prueba*.



## Conclusiones

Si bien para realizar la navegación y lógica de negocio podemos hacer uso indistintamente ya sea de un Servlet o de un JSP, hay marcadas diferencias que nos harán realizar algunas operaciones con uno o con otro dependiendo de que sea lo que queremos lograr en nuestra aplicación web.

Un JSP nos proporciona una manera sencilla de realizar operaciones pequeñas como pasar datos de una página a otra sin meternos mucho en el código. En cambio, un Servlet nos provee de una mayor funcionalidad para desarrollar la lógica de negocio de una manera más maleable frente a diferentes situaciones en las que debamos utilizar cosas más complejas.

# Práctica 3: Registro de laptops prestadas a maestros

Se necesita que se realice un registro de las laptops que los maestros ingresan al Tecnológico, se deberán capturar los siguientes campos: número de serie de la máquina, marca, modelo, color, nombre del maestro.

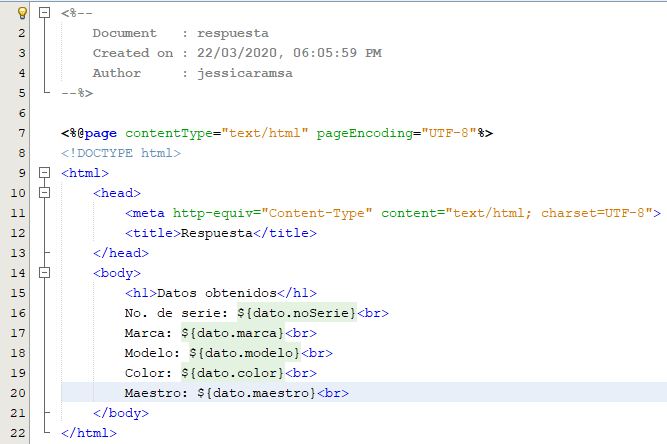
En nuestro archivo *index.jsp* tendremos el formulario de registro para una nueva computadora, especificaremos que el servlet a utilizar será *MiServlet2*.



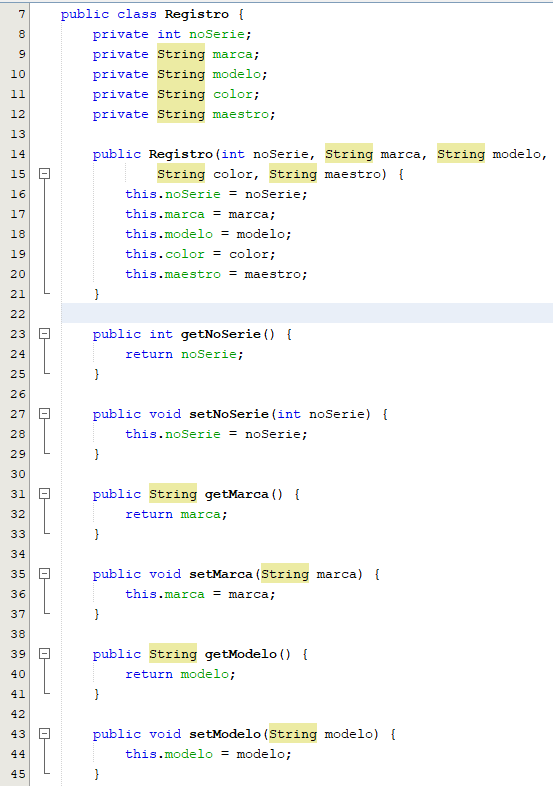
Crearemos un servlet llamado *MiServlet*, el cual se encargará de solicitar los datos capturados y mostrarlos en una nueva página.

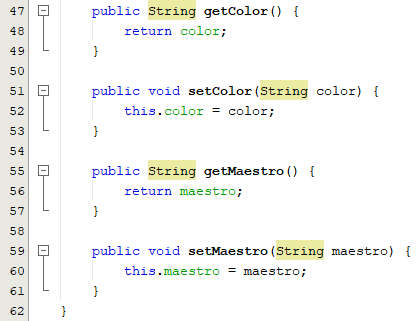


Ahora crearemos una página JSP llamada respuesta, en donde mostraremos los datos.

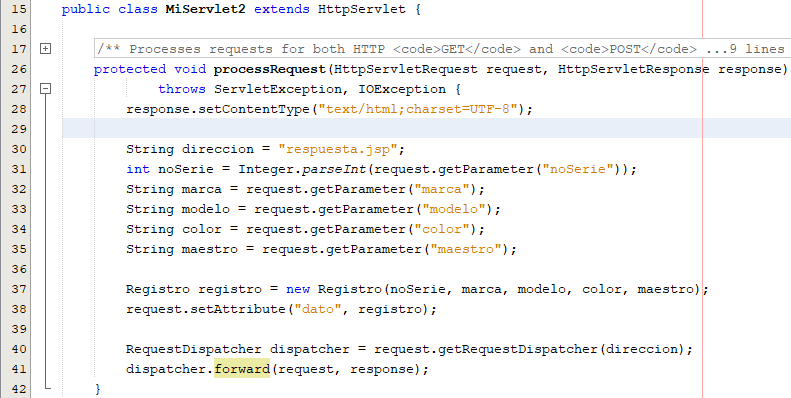


Para realizar un correcto mapeo de datos, crearemos una clase Java llamada *Registro* dentro del paquete *app.dominio* en nuestos *Source Packages*., se declararán los mismos datos a capturar que en el formulario y se agregrarán sus respectivos setter y getter.



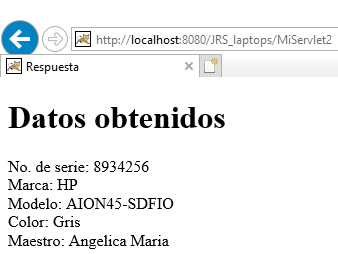
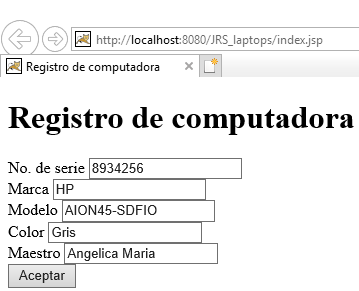


Por último, crearemos un segundo servlet llamado MiServlet2, que se encargará de la lógica para mostrar los datos utilizando nuestra clase Java Registro para realizar el mapeo.



## Resultados

Capturamos los datos de la computadora y enviamos la información, al momento en que carga la página de respuesta vemos que se muestran los datos que habíamos capturado con anterioridad.



Por último, vemos que el árbol de nuestro proyecto queda de la siguiente manera.



## Conclusiones

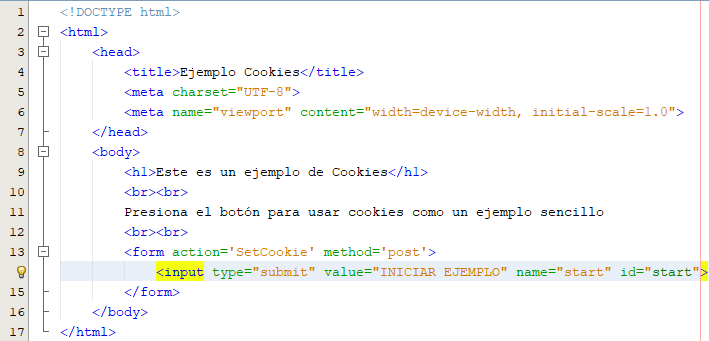
El hacer uso de un Java Bean como lo vimos en el desarrollo de la práctica te ayuda a tener un grado de persistencia cuando estamos tratando entidades en la captura de datos.

El uso de esto Java Bean nos permite hacer uso de datos consistentes, evitando problemas con el tipado de los datos o algún otro problema que se relacione con los datos base utilizados.

# Práctica 4: Ejemplo Cookies

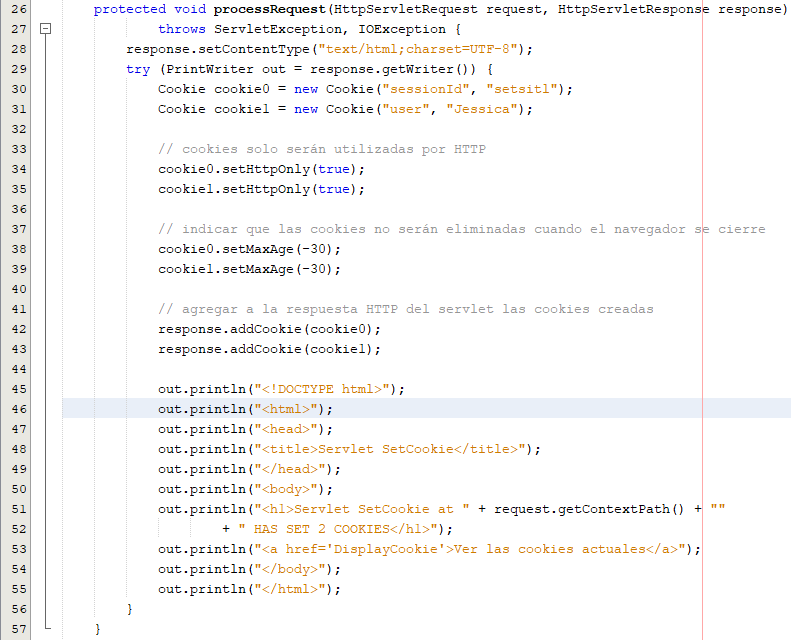
Realizaremos un pequeño ejemplo que nos mostrará el uso de las cookies en el navegador.

Crearemos un nuevo proyecto y en el archivo *index.html* tendremos nuestro formulario para empezar a usar cookies.

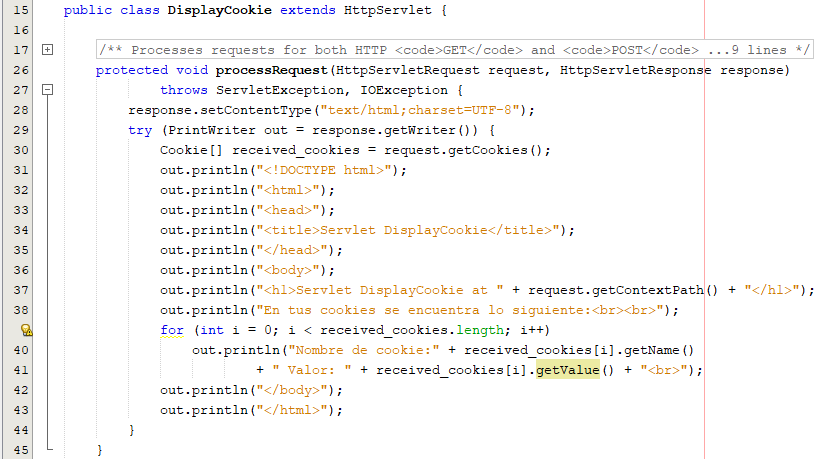


Dentro de la carpeta Source Packages, creamos un nuevo paquete Java llamado *servlets* y ahí creamos dos servlets, uno llamado *SetCookie* y otro *DisplayCookie*.

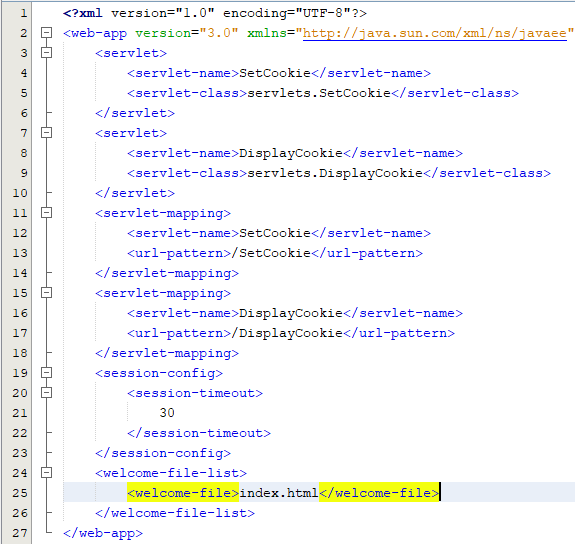
Dentro de SetCookie tendremos el siguiente código que nos permitirá establecer atributos de las cookies que guardaremos y después mostraremos en el navegador.



En el servlet *DisplayCookie* tendremos la lógica para mostrar el nombre y valor de cada cookie guardada.



Por último, editaremos nuestro archivo *web.xml* que se encuentra en la carpeta *WEB-INF* para indicar que nuestra página principal corresponde a *index.html*.

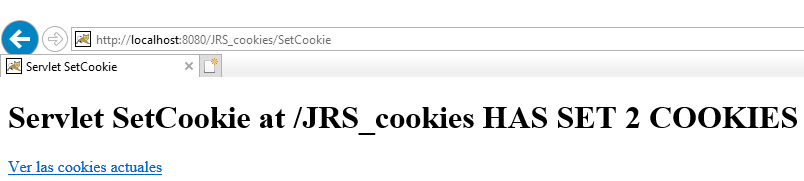


## Resultados

Cuando corremos el proyecto nos aparecerá la página principal *index.html*.



Si presionamos el botón INICIAR EJEMPLO nos redireccionará a la página donde guardaremos las cookies.



Si damos clic en el vínculo para revisar las cookies veremos lo siguiente.



## Conclusiones

Las cookies son archivos creados por un sitio web que contienen pequeñas cantidades de datos y se almacenan en el navegador del usuario, así el sitio puede consultar la actividad del navegador.

Como se vio en esta práctica, se puede identificar al usuario almacenando su historial de actividad, de manera que se le pueda ofrecer contenido más apropiado. Además, se pueden recordar los accesos para saber si ya ha visitado una página o no.

Dicho esto, una cookie puede sernos útil en este tipo de casos, pero por otro lado no es buena práctica depender de las cookies. Esto debido a que en ocasiones los mismos usuarios bloquean el guardado de las cookies en todos los sitios, lo que nos impediría que dichos usuarios tengan la misma experiencia de un usuario que tiene habilitado el guardado de las cookies.

# Práctica 5: Sesiones y filtros

Se realizará un ejemplo sencillo para ejemplificar el uso de sesiones y filtros.

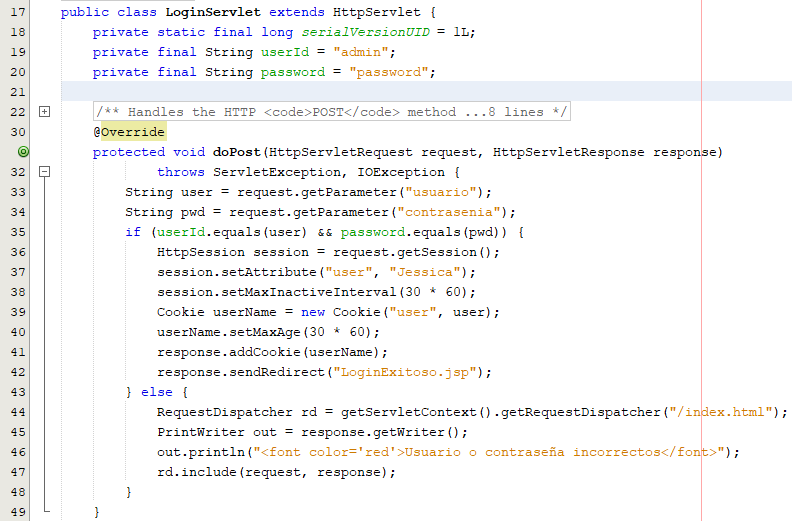
## Desarrollo

### Sesiones

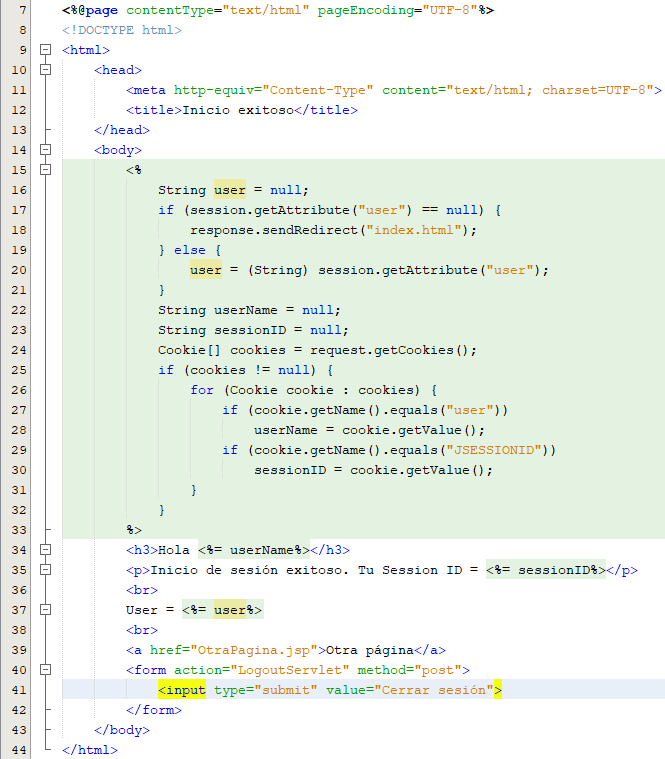
En el archivo HTML llamado *index* tendremos el siguiente formulario.



Dentro de *Source Packages* crearemos un nuevo paquete Java llamado *servlet*. Después crearemos el primer servlet llamado *LoginServlet* en el cual crearemos las cookies y la sesión.



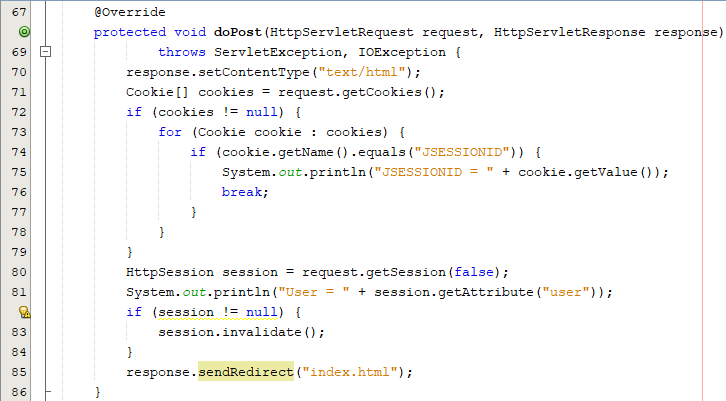
A continuación, se creará el JSP *LoginExitoso*, que se mostrará en caso de que el usuario haya iniciado sesión correctamente.



Se creará una nueva página llamada *OtraPagina.jsp* con el mismo código que la anterior, bloqueando su acceso en caso de no haber una sesión activa.

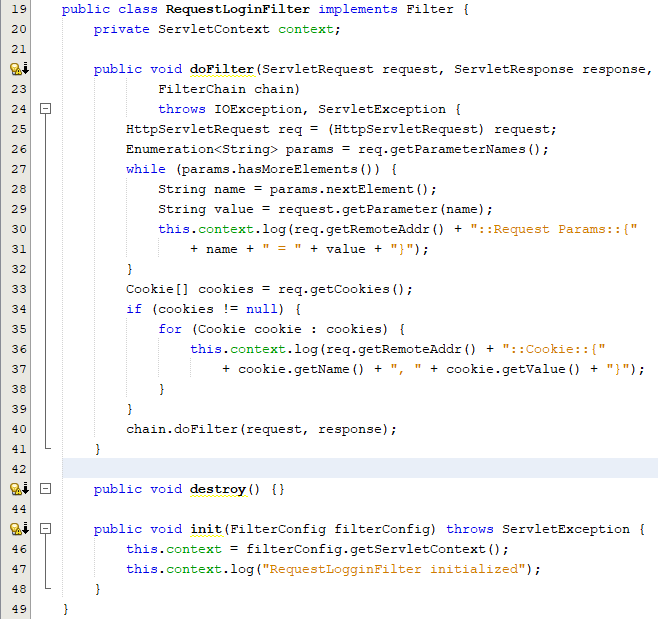


El servlet *LogoutServlet* contiene la lógica para cerrar sesión.

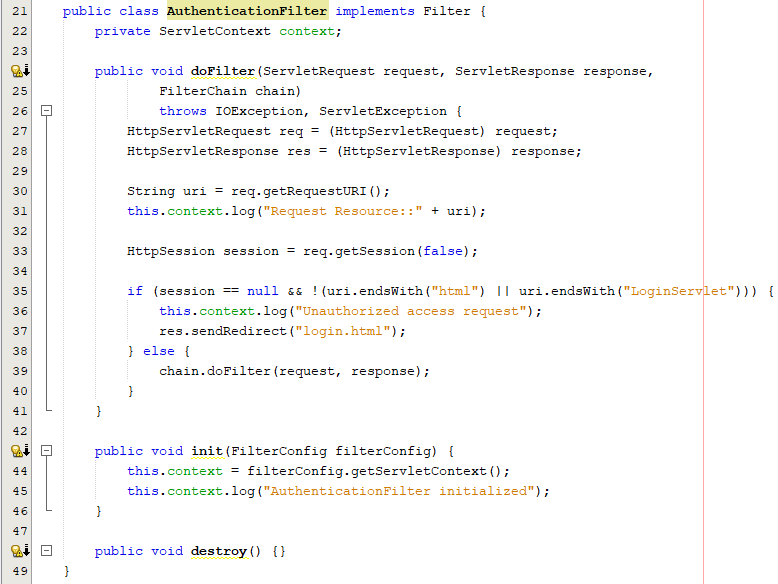


### Filtros

Crearemos un paquete Java llamado *filtros* dentro del *Source Packages*. Dentro crearemos un *Filter* con nombre *RequestLoginFilter*, en la especificación de los *Filter Mappings* crearemos uno nuevo especificando que se realizará mediante la *URL /\**, las *Dispatch Conditions* será mediante *REQUEST*.



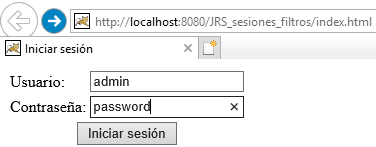
Crearemos un segundo *Filter*, llamado *AuthenticationFilter* que se encargará mostrar datos de la sesión y otros parámetros.



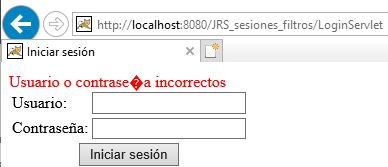
## Resultados

### Sesiones

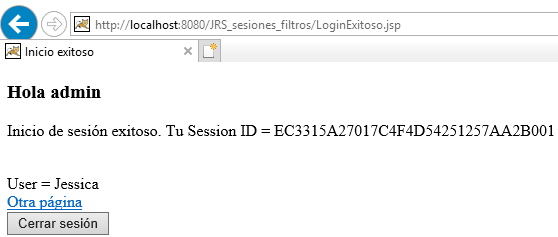
Una vez se ejecute el archivo *index.html* se nos mostrará el inicio de sesión en donde deberemos introducir “*admin*” como usuario y “*password*” como contraseña.



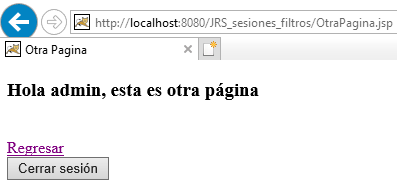
Si introducimos datos incorrectos nos mostrará el mensaje de error.



Cuando hayamos iniciado sesión correctamente nos mostrará la página principal.



Si vamos a la segunda página *Otra página*, veremos que la sesión sigue activa y nos permite la navegación entre las distintas páginas.



Si intentamos cerrar sesión nos regresará a la página del login y nuestra sesión será destruida.

### Filtros

Cuando veamos la consola veremos los mensajes de logs que configuramos en los filtros.

## Conclusiones

Las sesiones son conexiones que se establecen entre un equipo cliente y un servidor. Las páginas web se destruyen y vuelven a crearse a continuación con cada nueva petición al servidor, la información de las páginas no existe más allá del ciclo de vida de una página.

Las sesiones nos facilitan la consulta de información con el servidor ya que se trata directamente la información, haciendo la identificación del usuario y evitando el uso de cookies que tienen desventajas al usarlas por todo nuestra página web.

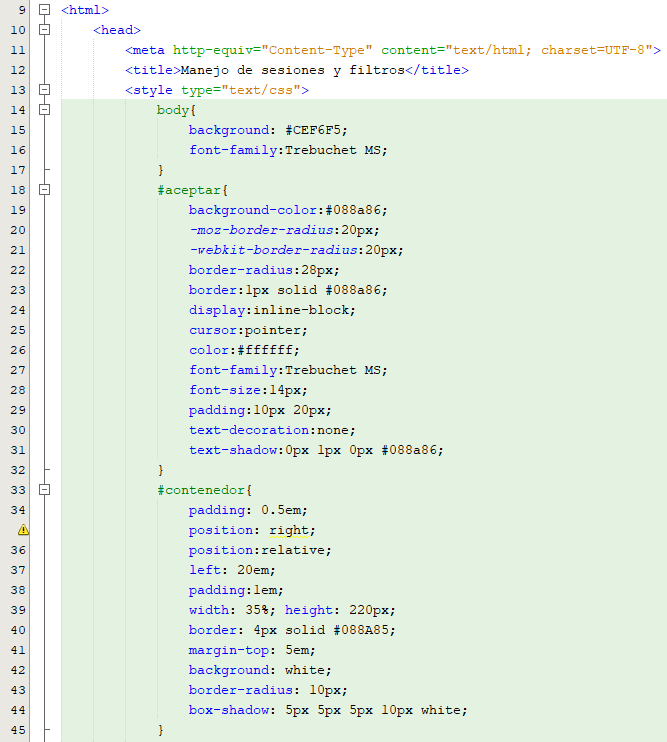
Por otro lado, un filtro (filter) es un objeto que es invocado en el pre-procesamiento y post-procesamiento de una petición realizada por el cliente. Es principalmente utilizado para mejorar el rendimiento de las tareas tales como conversión, logging, compresión, encriptación y desencriptación.

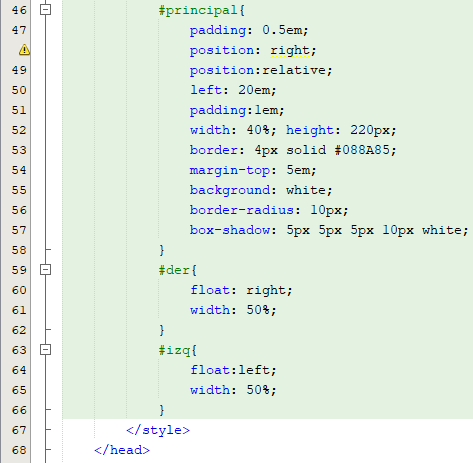
Los filtros nos resultan útiles para ver el log de cada petición realizada por el cliente, con el fin de llevar un registro de las acciones y posibles errores que tenga nuestra aplicación web.

# Práctica 6: Uso de scriptlets, declaraciones, expresiones y clases.

Se realizará una aplicación web que utilice scriptlets, declaraciones y expresiones, para analizar su uso, forma de implementación y ventajas de cada uno de ellos. Crearemos un JSP llamado *Inicio* y otro *Principal*.

Los estilos para *Inicio* son los siguientes.

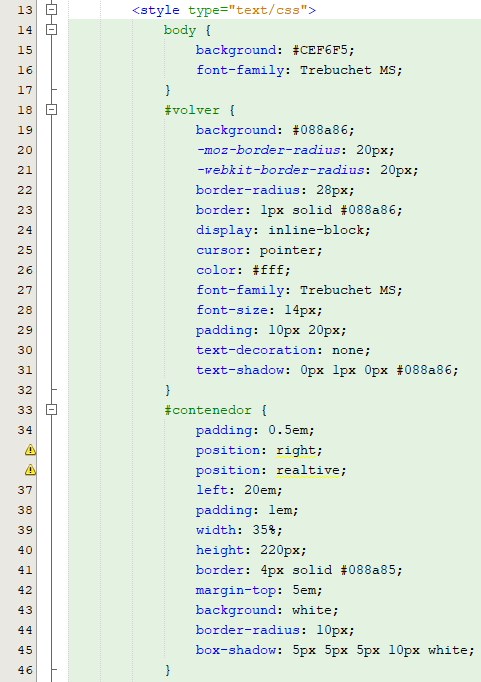


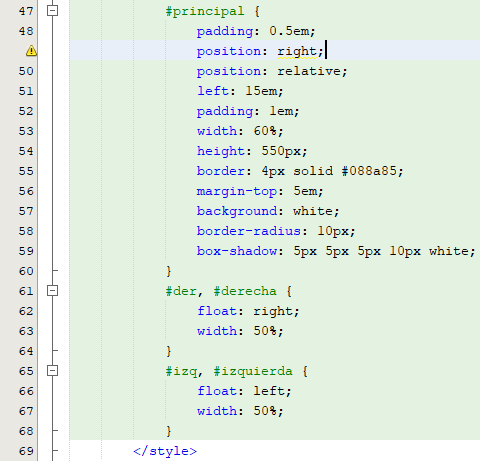


A continuación, veremos el formulario para capturar el nombre y la edad del usuario.



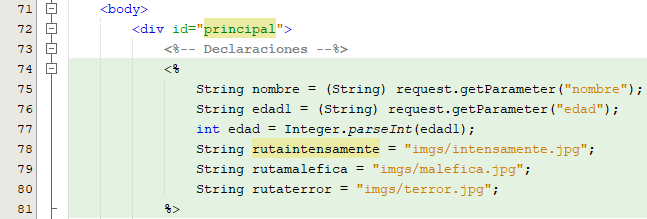
Los estilos para *Principal* son los siguientes.





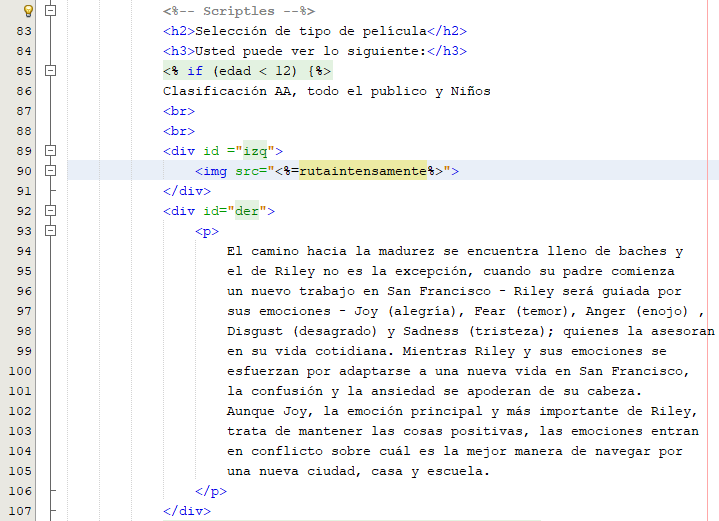
Dentro de la etiqueta *body* haremos uso de declaraciones y scriptlets. Para ejemplificarlo realizaremos un ejemplo en donde mostremos diferentes contenidos de acuerdo a la edad que recibamos.

Haciendo uso de las declaraciones, inicializaremos diferentes variables, para obtener el nombre, edad del usuario y obtener las rutas de las imágenes a mostrar.

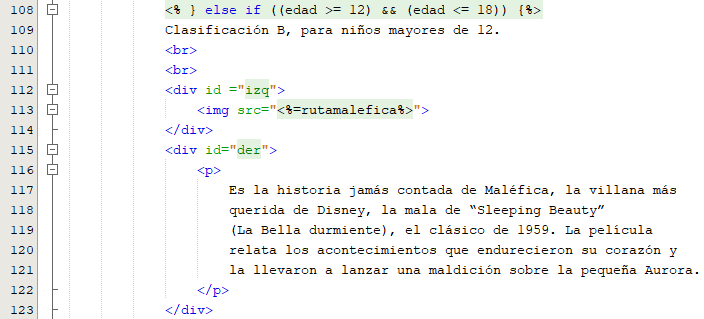


Continuaremos con los scriptlets que nos ayudarán a realizar el cambio de contenido de acuerdo a la edad.

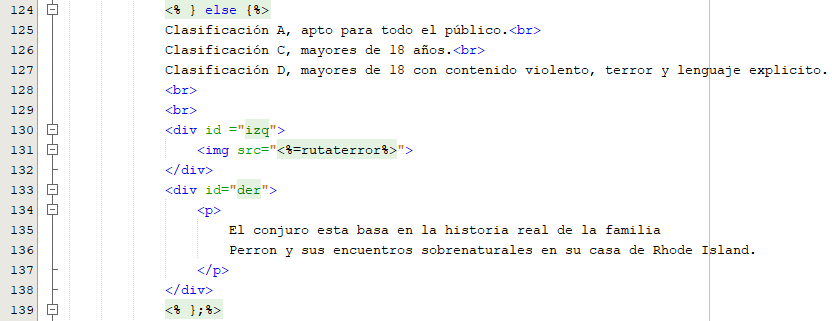
Si la edad es menor a 12 años.



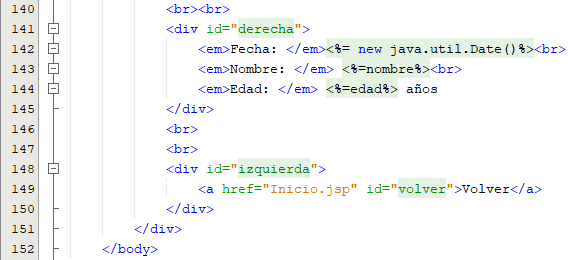
Si la edad es igual o mayor a 12 años, pero menor o igual a 18.



O si es mayor a 18 años.



Por último, mostraremos datos como la fecha actual, nombre y edad del usuario.

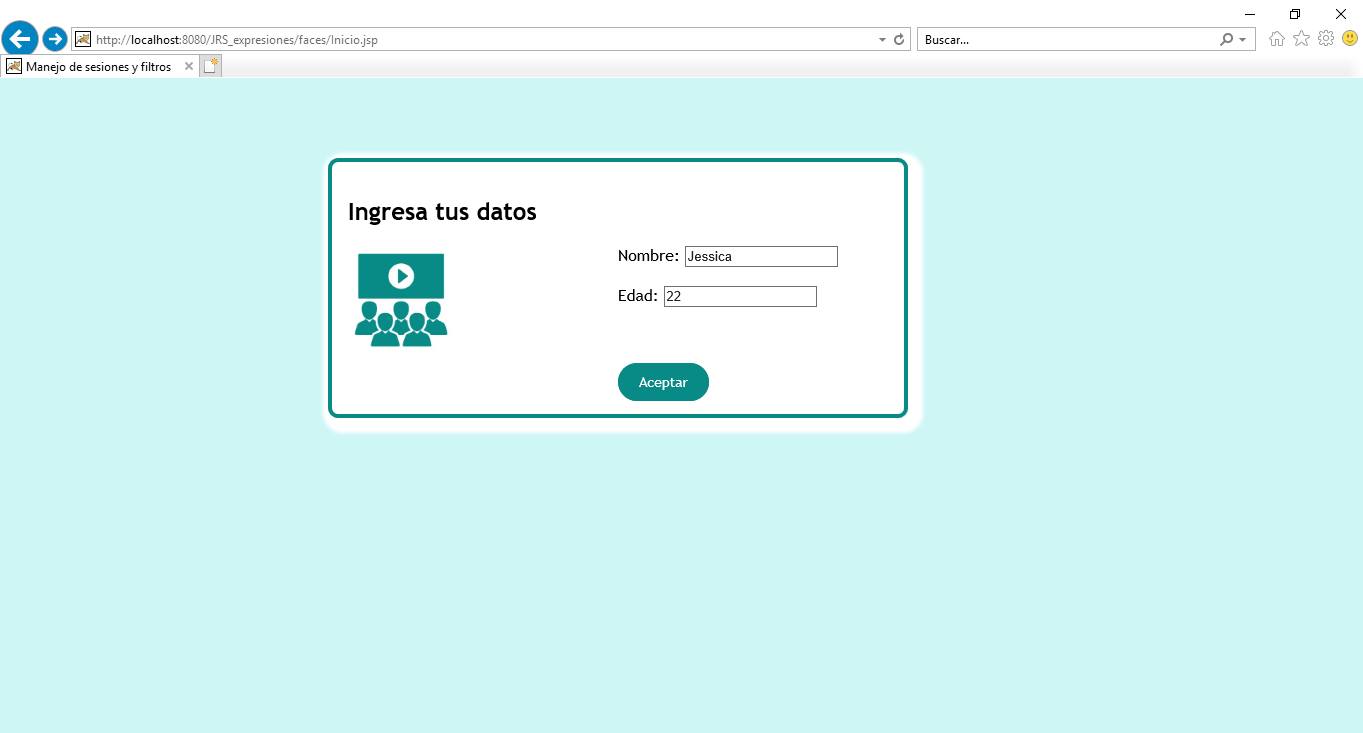


Por último, editamos el archivo *web.xml* que está en la carpeta *WEB-INF* para especificar el archivo inicial será *Inicio.jsp*.

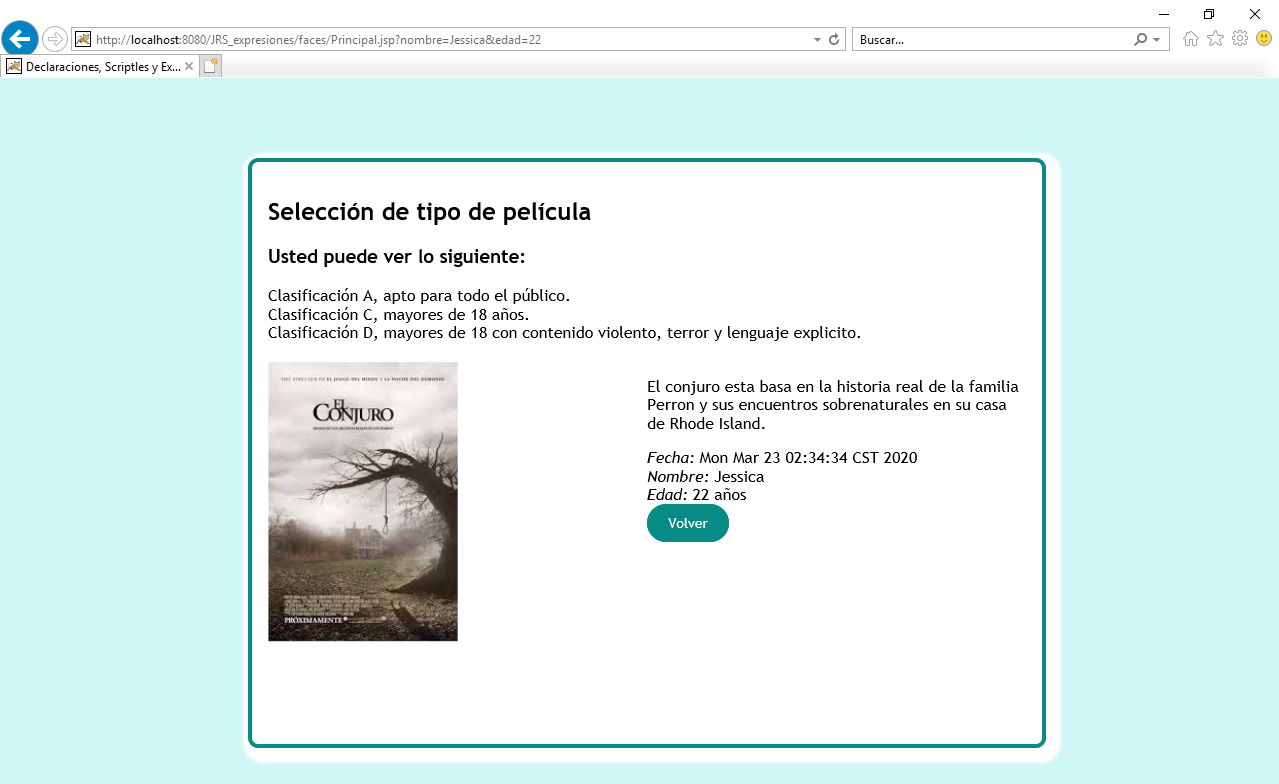


## Resultados

Cuando ejecutamos el proyecto se nos mostrará la página de inicio en donde capturaremos el nombre y la edad.



Al capturar una edad mayor a 18 años se nos muestra el siguiente contenido.



Cuando la edad está en el rango de 12 a 18 años.



Y cuando la edad es menor a 12 años.



## Conclusiones

Directivas, expresiones y scriptlets son diferentes formas en como dentro de un archivo JSP podemos introducir código Java, cada una con diferentes propósitos.

Las directivas ayudan a agregar elementos a un archivo. Si en un archivo JSP incluimos otro JSP podremos introducir ese código sin necesidad de escribir todo en un mismo archivo. Por ejemplo, si tenemos un archivo JSP *index* que tendrá como funcionalidad mostrar la página de inicio de la página web y en otro archivo JSP tenemos el *menú* estructurado, podemos agregar el archivo del menú con una directiva con la siguiente estructura: *<%@include %>*

Las declaraciones son útiles para declarar variables o atributos globales dentro del archivo JSP. Por ejemplo, declaremos la variable *rol* con valor *admin*: *<% String rol = “admin”;%>*

Los scriptlets permiten añadir cualquier código Java, ya sean variables y expresiones. De acuerdo al código de la práctica desarrollada, vimos que podemos que podemos agregar estructuras de control que nos permiten mostrar diferentes cosas de acuerdo a una validación que se realiza con una variable declarada dentro de las declaraciones de un archivo JSP. Por ejemplo, podemos agregar una estructura if para mostrar un mensaje u otro de acuerdo si nuestro rol es *admin* o no:

*<% if (rol == “admin”) {%>*

*<h3>Tienes un rol admin</h3>*

*<% } else {%>*

*<h3>Bienvenido usaurio</h3>*

*<% }%>*

Las expresiones evalúan y convierten la expresión a una cadena. Por ejemplo, si queremos mostrar un número o cualquier expresión simple tendríamos que tener una estructura como la siguiente:

*<%= 27%>*