

Practica 1.

PROTOCOLO HTTP://SERVIDORES_Y_APLICACIONES_WEB



200 OK

INDICE

1. La seguridad en la red. (Wireshark).

1.1	Captura de paquetes con Wireshark.	3
1.1.1	Ejemplo: petición y respuesta de una página básica.	4
1.1.2	Ejemplo: petición y respuesta de una página con elementos multimedia.	5
1.1.3	Ejemplo: petición y respuesta de una página con formulario.	7

2. El protocolo HTTP. (Fiddler y ServidorHTTP)

2.1	Ciclo Petición / Respuesta con Fiddler	9
2.2	Observar la petición del cliente con ServidorHTTP	14
2.3	Cookies con ServidorHTTP	15
2.4	Paso de parámetros con ServidorHTTP	16

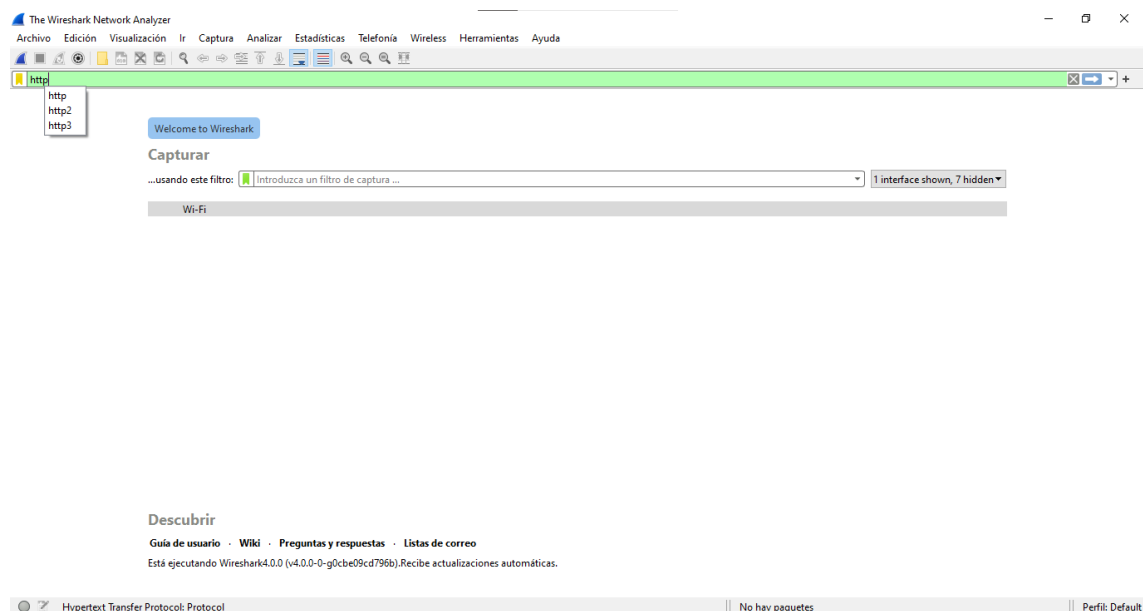
3. Servidores y aplicaciones web. (Tomcat)

3.1	Programas en JSP.	18
3.2	Aplicaciones Web en el cliente.	20
3.3	Páginas estáticas.	21

1.La seguridad en la red. (Wireshark)

1.1 Captura de paquetes con Wireshark.

Para la captura de paquetes con Wireshark deberemos seleccionar nuestro adaptador de red. Luego podremos capturar el tráfico de nuestra red(clicando en el icono azul con forma de aleta). Wireshark nos da la opción de filtrar los protocolos que queramos ver (franja verde de la imagen).



Para ver un poco el funcionamiento del programa, usaremos ejemplos facilitados para esta práctica.

Para ello usaremos un navegador y una pagina con una implementación html básica.

1.1.1 Ejemplo: petición y respuesta de una página básica.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.15	192.168.0.15	TCP	54	51317 → 51317 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2	0.209900	192.168.0.15	209.126.103.48	TCP	54	51317 → 4043 [RST, ACK] Seq=3 Ack=1 Win=256 Len=0
3	0.910079	192.168.0.15	209.126.103.48	TCP	54	51317 → 4043 [RST, ACK] Seq=2 Ack=1 Win=0 Len=0
4	0.911044	192.168.0.15	185.176.43.76	TCP	66	51319 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
5	0.911417	192.168.0.15	185.176.43.76	TCP	66	51320 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
6	0.983257	185.176.43.76	192.168.0.15	TCP	62	80 → 51319 [SYN, ACK] Seq=0 Ack=1 Win=42340 Len=0 MSS=1452 WS=256
7	0.983306	192.168.0.15	185.176.43.76	TCP	54	51319 → 80 [ACK] Seq=1 Ack=1 Win=66560 Len=0
8	0.983556	192.168.0.15	185.176.43.76	HTTP	503	GET /wireshark0.html HTTP/1.1
9	0.984683	185.176.43.76	192.168.0.15	TCP	62	80 → 51320 [SYN, ACK] Seq=0 Ack=1 Win=42340 Len=0 MSS=1452 WS=256
10	0.984762	192.168.0.15	185.176.43.76	TCP	54	51320 → 80 [ACK] Seq=1 Ack=1 Win=66560 Len=0
11	1.057904	185.176.43.76	192.168.0.15	TCP	54	80 → 51319 [ACK] Seq=1 Ack=450 Win=42240 Len=0
12	1.060023	185.176.43.76	192.168.0.15	HTTP	870	HTTP/1.1 200 OK (text/html)
13	1.104450	192.168.0.15	185.176.43.76	TCP	54	51319 → 80 [ACK] Seq=450 Ack=817 Win=65792 Len=0
14	1.356338	192.168.0.15	192.168.0.1	DNS	104	Standard query 0xf13e A anti-mining-protection-toolbar.urban-vpn.com
15	1.377902	192.168.0.15	192.168.0.1	DNS	104	Standard query 0xf13e A anti-mining-protection-toolbar.urban-vpn.com
16	1.385593	192.168.0.1	192.168.0.15	DNS	177	Standard query response 0xf13e A anti-mining-protection-toolbar.urban-vpn.com CNAME anti-mining-p...
17	1.386042	192.168.0.15	207.182.151.242	TCP	66	51321 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
18	1.421064	fe80::7a81:2ff:fe0e::fe0e	fe80::d8da:c1aa:35a::	ICMPv6	86	Neighbor Solicitation for fe80::d8da:c1aa:35a::282b from 78:81:02:0e:e4:f8
19	1.421099	fe80::d8da:c1aa:35a::	fe80::7a81:2ff:fe0e::fe0e	ICMPv6	86	Neighbor Advertisement fe80::d8da:c1aa:35a::282b (sol, ovr) is at laica:7a:80:3f:94
20	1.426111	192.168.0.1	192.168.0.15	DNS	177	Standard query response 0xf13e A anti-mining-protection-toolbar.urban-vpn.com CNAME anti-mining-p...
21	1.560094	207.182.151.242	192.168.0.15	TCP	66	443 → 51321 [SYN, ACK] Seq=0 Ack=1 Win=29208 Len=0 MSS=1452 SACK_PERM WS=128
22	1.560377	192.168.0.15	207.182.151.242	TCP	54	51321 → 443 [ACK] Seq=1 Ack=1 Win=66560 Len=0
23	1.537132	192.168.0.15	207.182.151.242	TLSv1.2	636	Client Hello
24	1.668106	207.182.151.242	192.168.0.15	TCP	54	443 → 51321 [ACK] Seq=1 Ack=583 Win=30464 Len=0
25	1.670382	207.182.151.242	192.168.0.15	TLSv1.2	1506	Server Hello
26	1.671578	207.182.151.242	192.168.0.15	TCP	1506	443 → 51321 [ACK] Seq=1453 Ack=583 Win=30464 Len=1452 [TCP segment of a reassembled PDU]
27	1.671578	207.182.151.242	192.168.0.15	TLSv1.2	1246	Certificate
28	1.671578	207.182.151.242	192.168.0.15	TLSv1.2	286	Server Key Exchange, Server Hello Done

En esta imagen podemos ver el tráfico de nuestro navegador al solicitar la página que tenemos como ejemplo, y se puede observar distintos paquetes y protocolos.

No.	Time	Source	Destination	Protocol	Length	Info
8	0.983556	192.168.0.15	185.176.43.76	HTTP	503	GET /wireshark0.html HTTP/1.1
12	1.060023	185.176.43.76	192.168.0.15	HTTP	870	HTTP/1.1 200 OK (text/html)

Aquí podemos ver como al aplicar el filtro sobre la página solicitada nos muestra únicamente los protocolos *http* que ha habido en el tráfico de nuestra red.

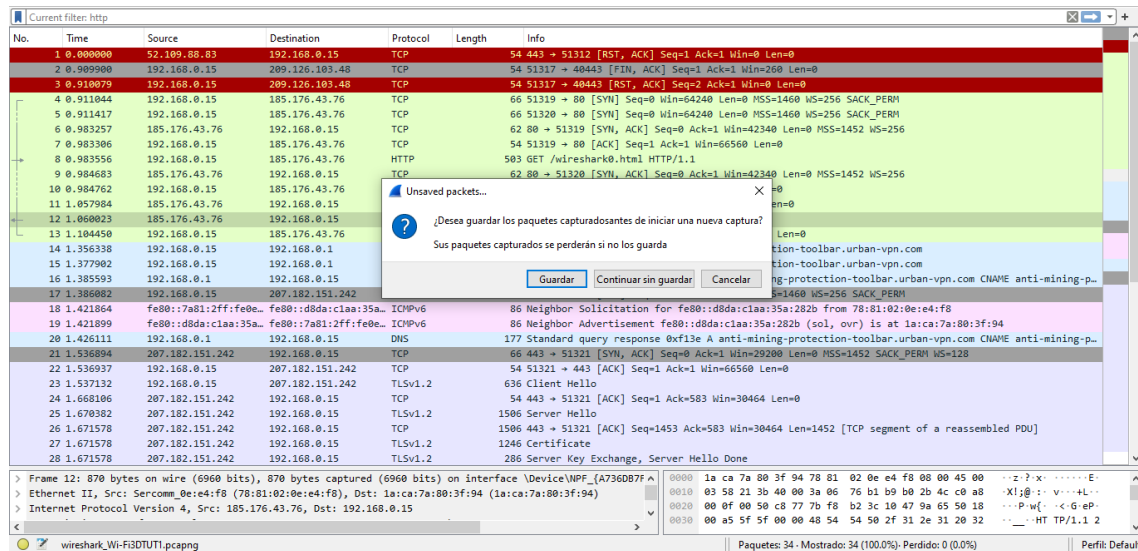
El paquete 8 contiene la petición de nuestro navegador

(GET /wireshark0.html HTTP/1.1).

Y en el paquete 12 podemos ver la respuesta por el servidor (HTTP/1.1 200 OK text/html). Wireshark nos permite ver el contenido de la cabecera de este paquete en el apartador de *Hypertext Transfer Protocol*, y más abajo podemos ver el contenido del *html* en *Line-based text data: text/html*

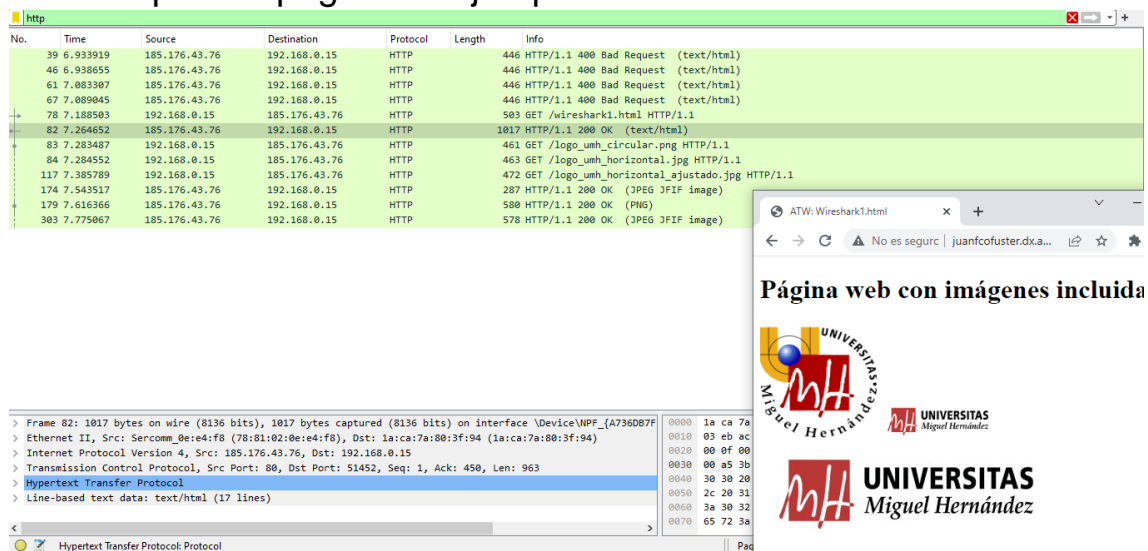
1.1.2 Ejemplo: petición y respuesta de una página con elementos multimedia.

A continuación, haremos un ejemplo de una petición a una página con elementos multimedia(imágenes) y veremos como exportar y guardar dichas imágenes.



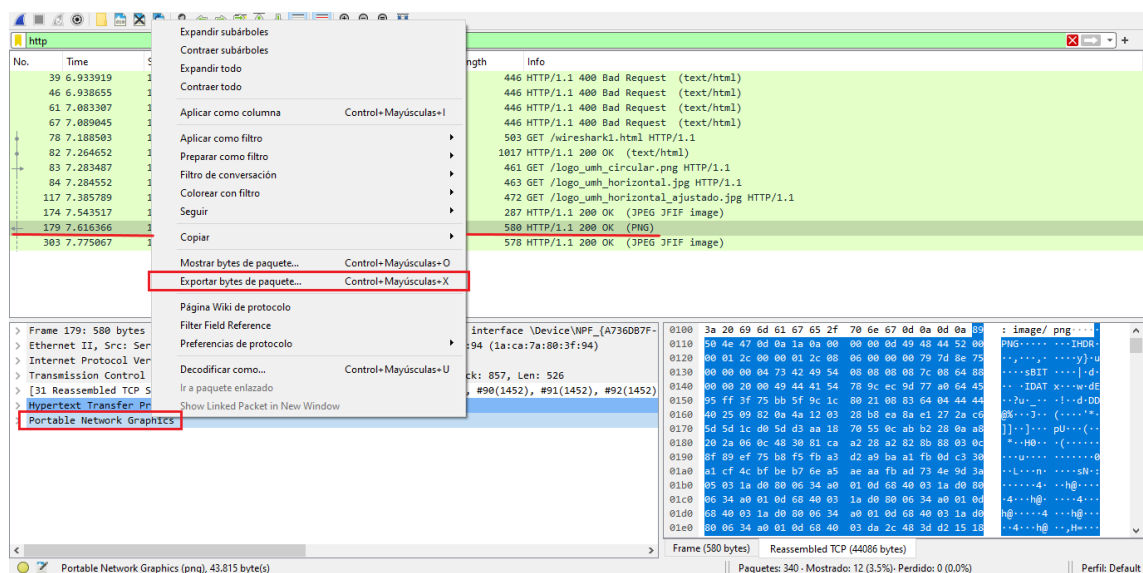
Para ello empezaremos otra captura con wireshark, que nos pedirá si queremos guardar los paquetes capturados anteriormente, lo cual no es necesario y podemos darle a **continuar sin guardar**.

En la imagen siguiente podremos ver las peticiones y respuestas del servidor para la página de ejemplo.



El paquete 78 contiene la petición a la página, y el 82 la respuesta del servidor. Después se puede ver las peticiones a las imágenes de la página, paquetes 83,84 y 117, y las respuestas a dichas peticiones del servidor, paquetes 174,179 y 303.

Para exportar y guardar las imágenes de los paquetes capturados, seleccionaremos el paquete respuesta del servidor que contiene la imagen, y en la ventana de protocolos daremos clic derecho en el apartado de *Portable Network Graphics* u en otro caso *JPEG File Interchange Format*, y le daremos a **Exportar bytes de paquetes**.



1.1.3 Ejemplo: petición y respuesta de una página con formulario.

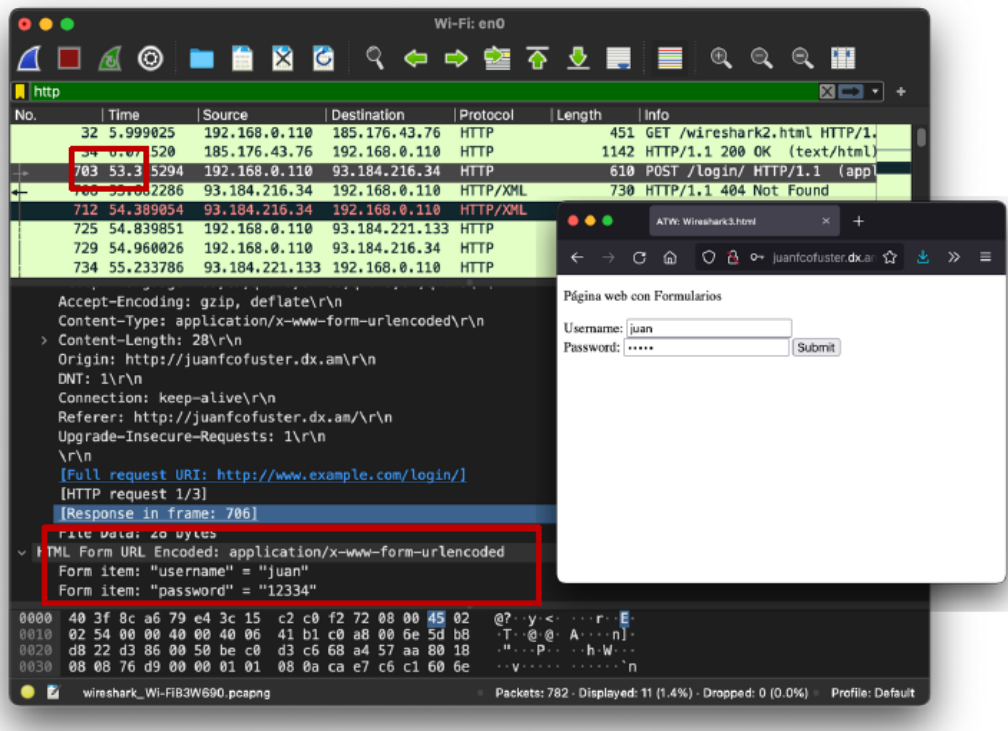
A continuación, veremos que sucede en nuestra red al realizar un formulario de una página. Para ello realizaremos otra captura con otra página de ejemplo.

The image shows a Wireshark network capture of HTTP traffic. The packet list on the left shows several 'Bad Request' (400) responses and one successful 'GET' (200 OK) response for 'wireshark2.html'. The packet details pane on the right shows the structure of the selected packet, including Ethernet II, Internet Protocol Version 4, Transmission Control Protocol, and Hypertext Transfer Protocol. Overlaid on the Wireshark window is a browser window titled 'ATW: Wireshark2.html' showing a login form with fields for 'Username:' and 'Password:', and a 'Submit' button. The page content is 'Página web con Formularios'.

Al enviarlo los datos de *username* y *password*, y darle al botón *Submit*, nuestro navegador no encuentra el sitio web indicado por lo que Wireshark no encuentra ningún protocolo http en esta petición.

The image shows a Wireshark network capture of traffic to 'www.example.com'. The packet list shows DNS queries and responses, followed by TCP connections. The packet details pane shows the structure of the selected packet, including Ethernet II, Internet Protocol Version 4, User Datagram Protocol, and Domain Name System. Overlaid on the Wireshark window is a browser window titled 'www.example.com' showing an error message: 'No se puede acceder a este sitio web'. The message text says: 'Comprueba si hay un error de escritura en www.example.com. Si está escrito correctamente, prueba a ejecutar el diagnóstico de red de Windows.' The browser's address bar shows 'example.com/login/'.

En otro caso de encontrar la página solicitada al enviar los datos del formulario debería pasar lo de la imagen a continuación.



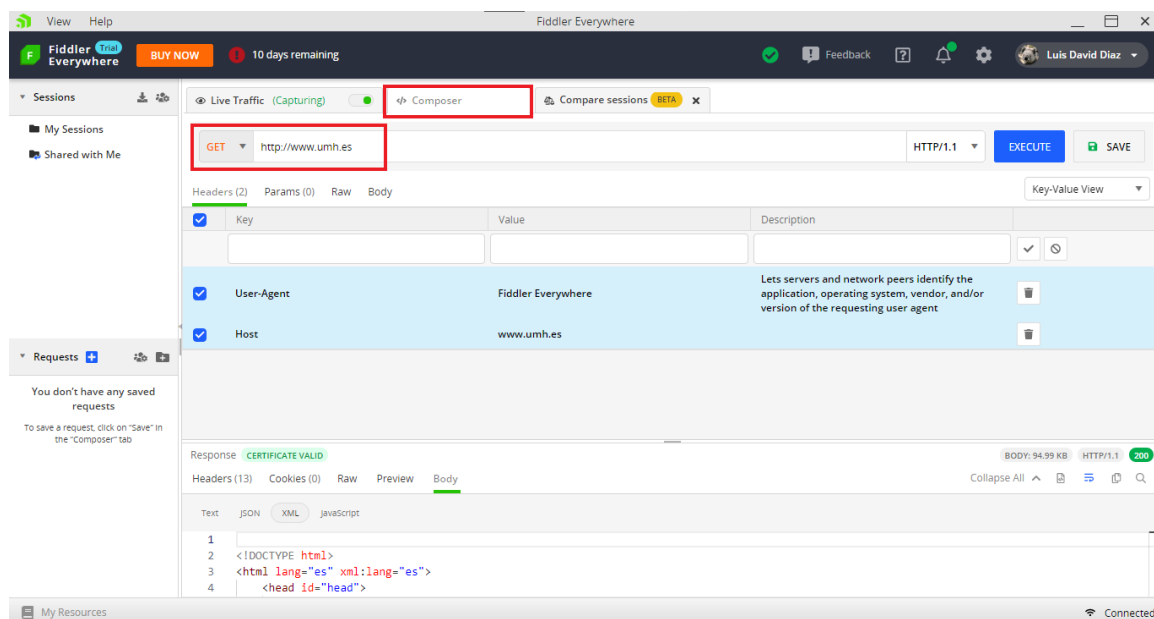
En este caso podemos ver que el servidor devuelve un POST en el paquete 703, donde estarán los datos que se han introducido en el formulario como texto plano, ya que no se está utilizando con conexión cifrada entre cliente y servidor.

2. El protocolo HTTP. (Fiddler y ServidorHTTP)

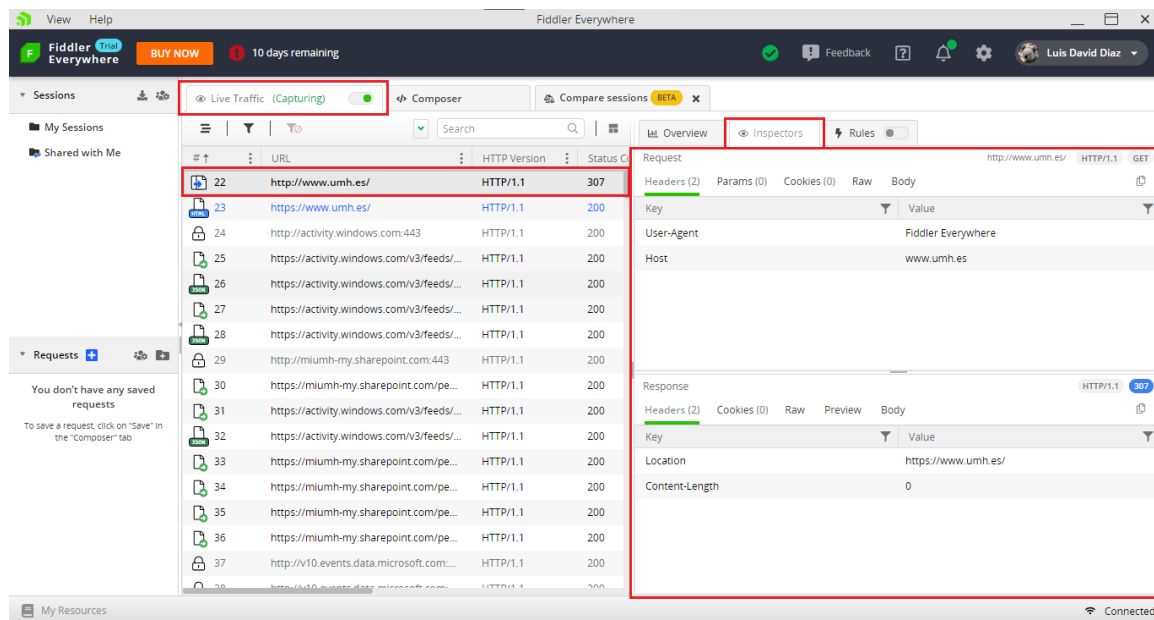
2.1 Ciclo Petición / Respuesta con Fiddler

En este apartado veremos como es empleado el protocolo HTTP entre el cliente y el servidor en la web. En esta parte usaremos el programa Fiddler y otro pequeño programa llamado ServidorHTTP, los cuales permiten “espiar” la información que envía la otra parte.

Ceñido al guion de la práctica primero trabajaré con Fiddler. En el paso 1 se pide realizar una petición GET sobre la web www.umh.es, en la pestaña **composer**.

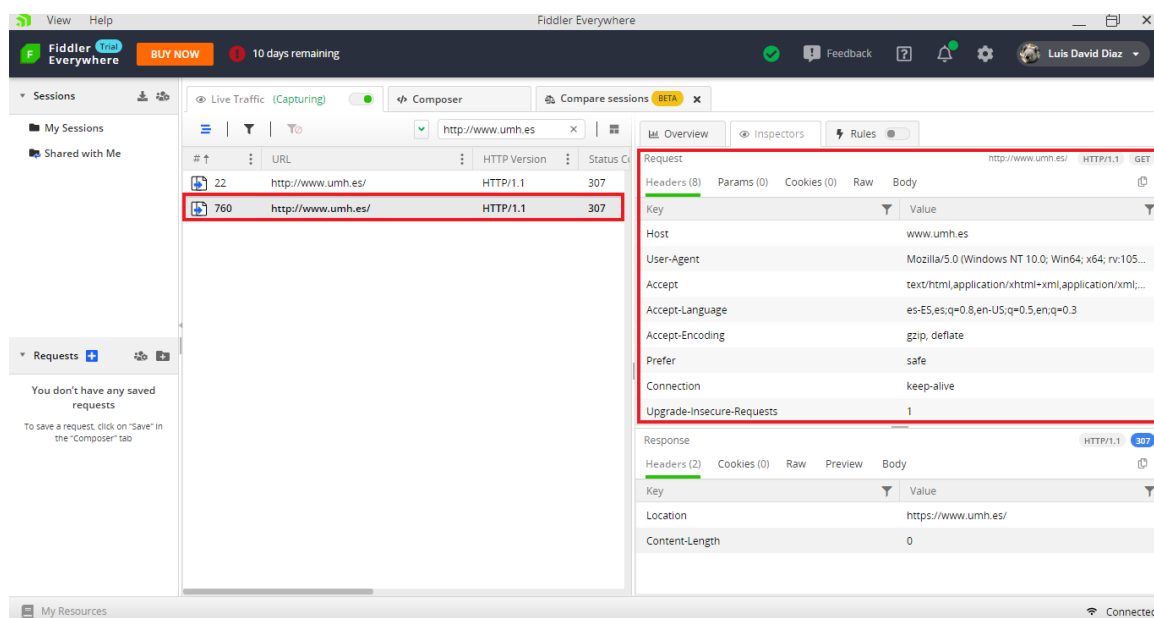


El siguiente paso (2) nos pide acceder a la pestaña **Inspectors** y analizar la petición enviada por el cliente y describir los resultados.



En esta petición como dice el guion de la práctica, se puede observar que la petición GET sobre www.umh.es tiene “/” al final, esto es equivalente a solicitar la página principal del sitio web.

Para el paso 3 se nos pide hacer una solicitud de la misma página, pero por el navegador y apreciemos las diferencias entre solicitudes.



En este caso se puede observar que al realizar la petición por el navegador se nos da mucha más información en la cabecera del cliente.

Para el paso 4 se nos pide rellenar la siguiente tabla para ver las diferencias entre peticiones, de Fiddler y el navegador, dónde se puede observar que la petición por el navegador ofrece más información, y lo único que tienen en común es el host del servidor (www.umh.es)

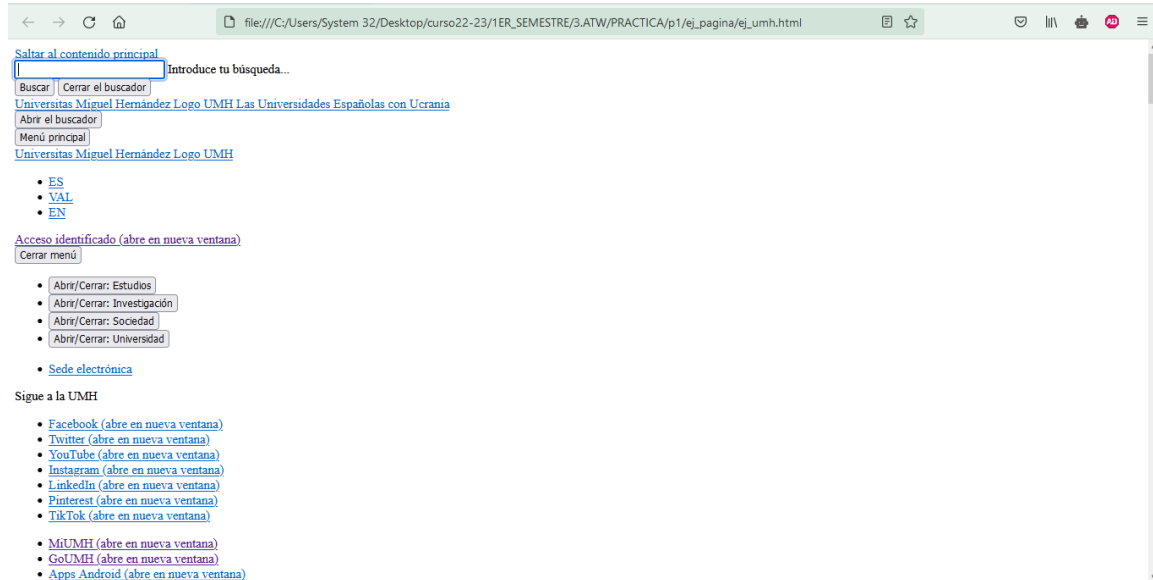
PETICIÓN FIDDLER		PETICIÓN NAVEGADOR	
LINEA DE PETICIÓN:			
www.umh.es		www.umh.es	
CABECERAS DE CLIENTE:			
CABECERA	VALOR / UTILIDAD	CABECERA	VALOR / UTILIDAD
Accept	text/html,application/xhtml+xml,application/xml;q=0.9 ,image/avif,image/webp,*/*;q=0.8	Accept	NULL
Accept-Language	es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3	Accept-Language	NULL
Accept-Encoding	gzip, deflate	Accept-Encoding	NULL
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0	User-Agent	Fiddler Everywhere
Host	www.umh.es	Host	www.umh.es
CABECERAS GENÉRICAS:			
CABECERA	VALOR / UTILIDAD	CABECERA	VALOR / UTILIDAD
Connection	keep-alive	Connection	NULL

A continuación, rellenaremos otra tabla para analizar esta vez la respuesta del servidor en la pestaña **Inspectors**, usando las pestañas de **Headers** y **Body** de la pestaña inferior.

RESPUESTA DEL SERVIDOR	
CÓDIGO DE ESTADO (VALOR / UTILIDAD):	
200	
CABECERA GENÉRICAS:	
CABECERA	VALOR / UTILIDAD
Date	Sat, 15 Oct 2022 17:55:40 GMT
Connection	keep-alive
CABECERAS DE SERVIDOR	
CABECERA	VALOR / UTILIDAD
Accept-Ranges	bytes
Server	
CABECERAS DE ENTIDAD	
CABECERA	VALOR / UTILIDAD
Content-Lenght	23.73 KB
Content-Type	text/html; charset=UTF-8
Content-Location	
Content-Lenguaje	es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3

En esta tabla podemos observar que nos aparece otros datos distintos a la tabla anterior como el código de estado, Date, Accept-Ranges, Content-Lenght y Content-Type

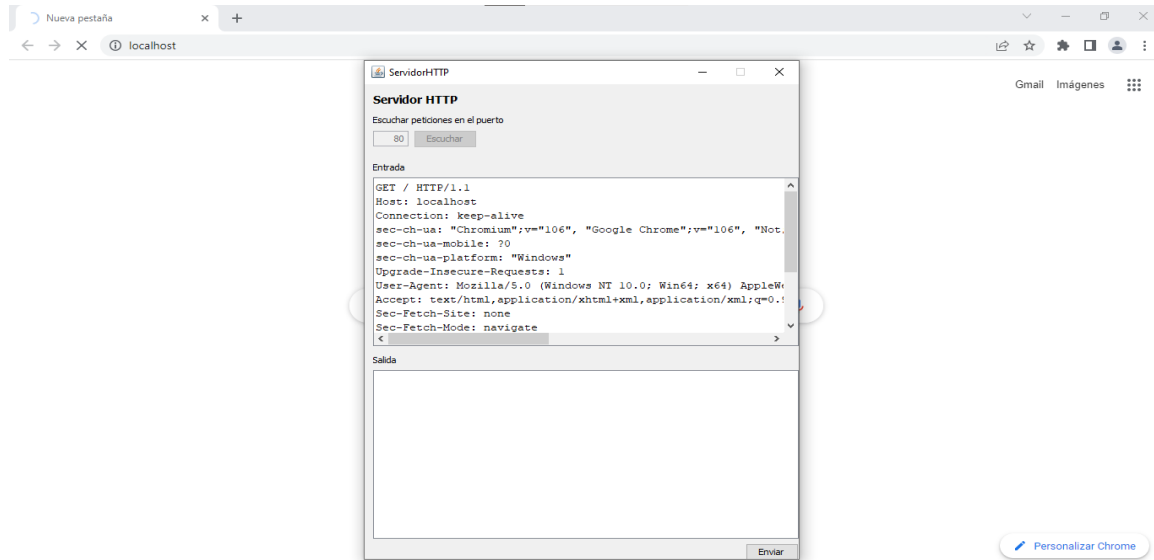
Luego de rellenar la tabla se pide que copiemos el cuerpo de la entidad HTML en un documento de texto plano y abrirlo en un navegador para ver que se trata de una página web, aunque esta no contendrá imágenes ni formatos de estilos CSS.



Se puede observar que el único contenido que aparece es texto, botones y un recuadro de búsqueda, ya que no disponemos de las imágenes y hojas de estilo CSS.

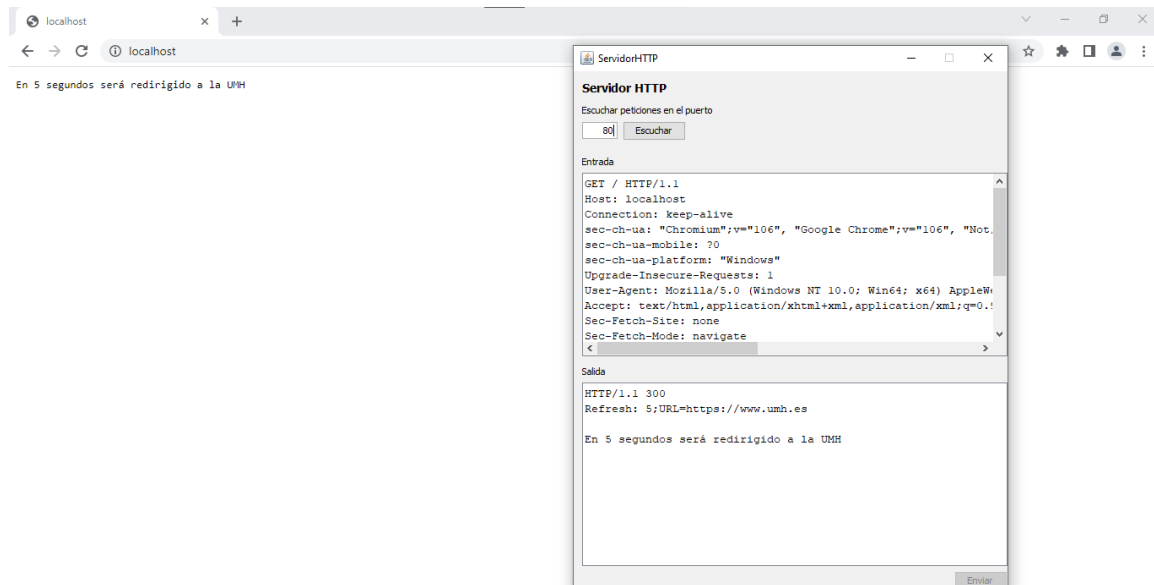
2.2 Observar la petición del cliente con ServidorHTTP

En este apartado observaremos como funciona una petición a un navegador real. Para ello haremos uso de un servidor ficticio que está situado en nuestra máquina local, y localhost es el nombre que se le da a esta en internet.



En la aplicación ServidorHTTP, tiene dos apartados, uno de entrada que es para la petición del cliente, aparece la información enviada por el navegador. Y el otro apartado, es para la respuesta del servidor, aquí podemos escribir información que queremos devolver al navegador

Siguiendo el guion de prácticas, se enviará una cabecera al cliente para redireccionarlo a otro sitio.



Para que esto suceda debemos haber escrito en el apartado de **Salida** las siguientes líneas:

HTTP/1.1 300

Refresh: 5;URL=https://www.umh.es

En 5 segundos será redirigido a la UMH.

En la primera línea ponemos el protocolo y el código de estado, el código que hemos usado sirve para las redirecciones de elección múltiple, y de esto se encarga el navegador. La siguiente línea hemos usado la cabecera **Refresh** para redirigir al navegador automáticamente, pasado 5 segundos, a la URL indicada. La última línea después de un salto, lo usamos para enviar un mensaje por el navegador.

Una vez le demos al botón de **Enviar** veremos como se actualiza la ventana del navegador con ese mensaje, y pasado 5 segundos nos mandará a la página principal de la UMH.

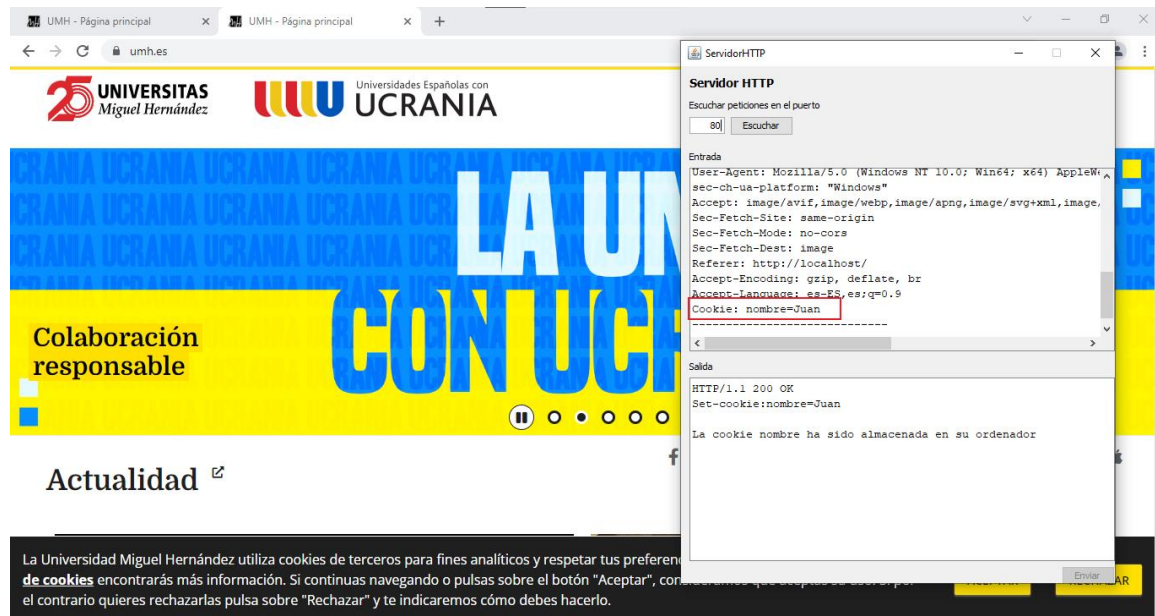
2.3 Cookies con ServidorHTTP

Las cookies son datos que el servidor puede almacenar en el cliente(navegador). En este ejemplo guardaremos un dato directamente desde el servidor con la siguiente respuesta a una petición HTTP:

HTTP/1.1 200 OK

Set-cookie:nombre=Juan

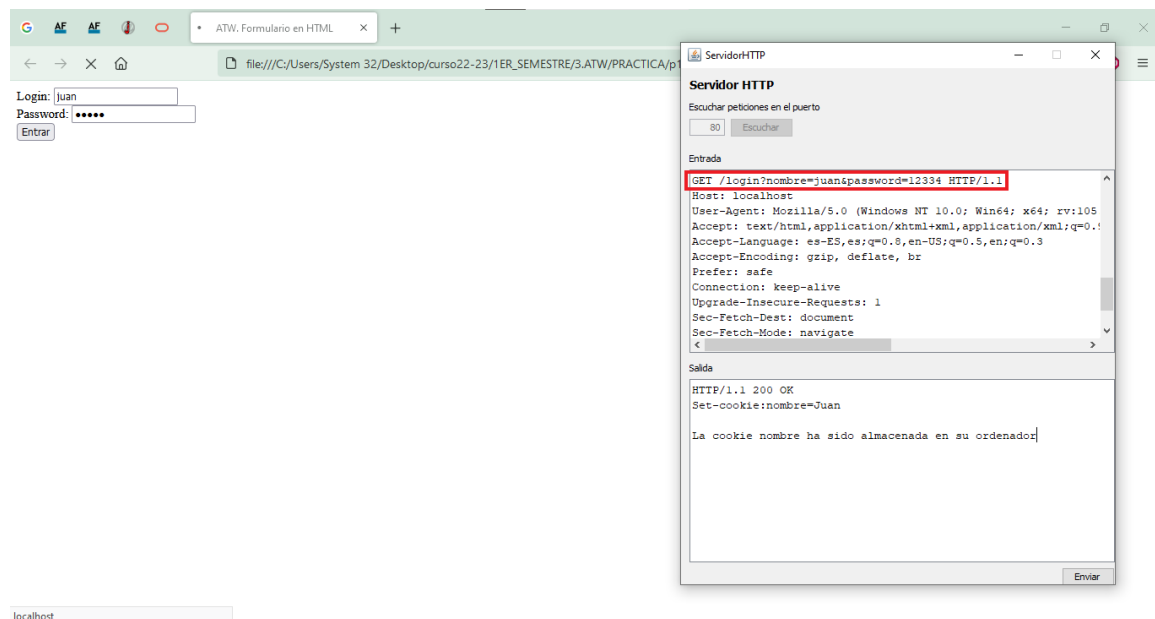
La cookie nombre ha sido almacenada en su ordenador.



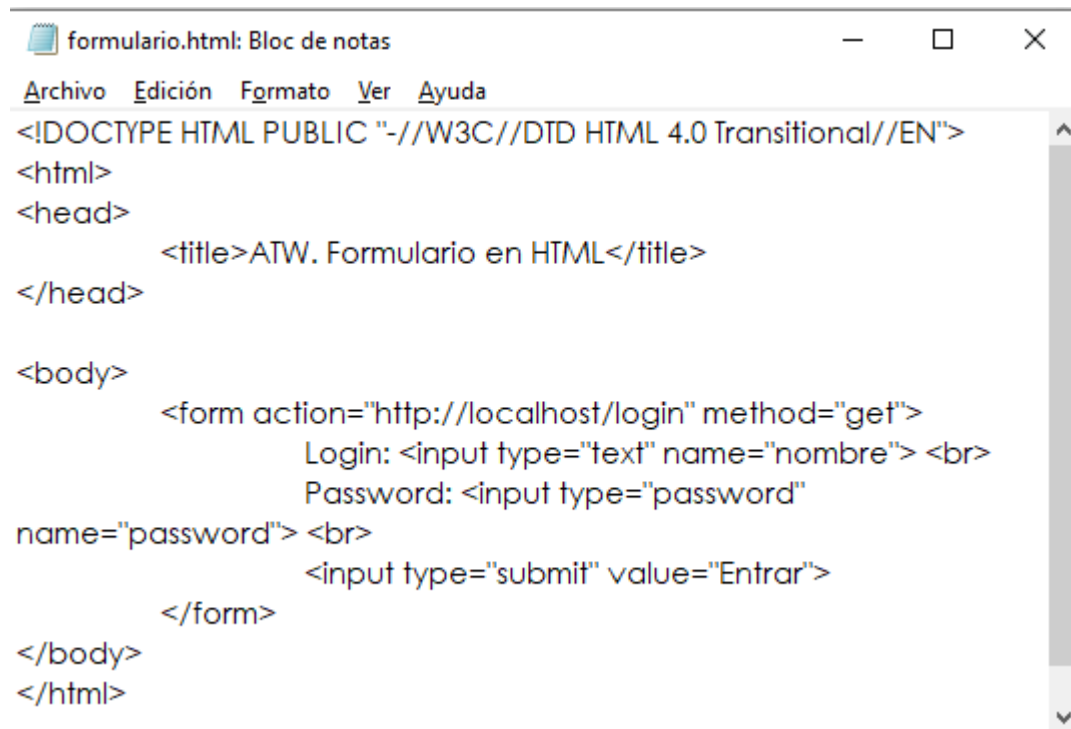
Se puede observar que al realizar una nueva petición sin cerrar la pestaña dónde hemos enviado la cookie, en esta nueva el cliente nos envía la información con esta cookie ya guardada.

2.4 Paso de parámetros con ServidorHTTP

En este ejemplo vemos como se pasan parámetros en una petición HTTP mediante el método GET



Para revisar en que parte del fichero se envían las variables del método GET se ha de revisar el código de este. Y se puede ver fácilmente que se envía dentro de la etiqueta *body* y *form*.



The screenshot shows a Notepad window titled 'formulario.html: Bloc de notas'. The menu bar includes 'Archivo', 'Edición', 'Formato', 'Ver', and 'Ayuda'. The text area contains the following HTML code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
  <title>ATW. Formulario en HTML</title>
</head>

<body>
  <form action="http://localhost/login" method="get">
    Login: <input type="text" name="nombre"> <br>
    Password: <input type="password"
name="password"> <br>
    <input type="submit" value="Entrar">
  </form>
</body>
</html>
```

3. Servidores y aplicaciones web. (Tomcat)

3.1 Programas en JSP

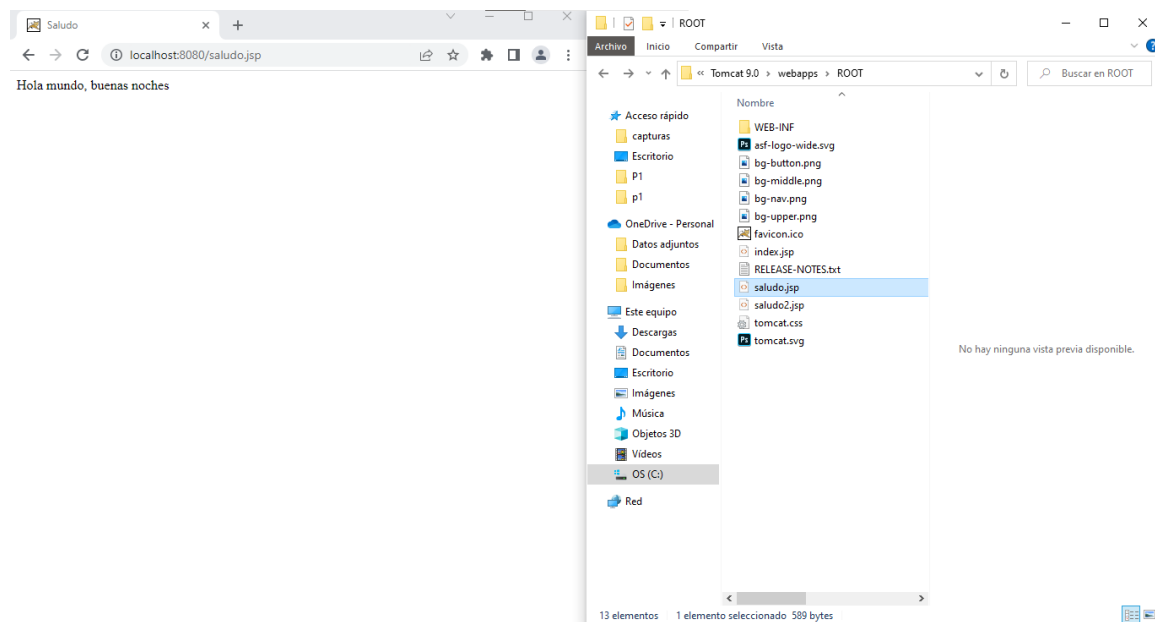
JavaServer Pages (JSP) es una tecnología que permite crear páginas web dinámicas mediante la inclusión de código Java en páginas HTML. El código Java se ejecuta en el servidor al solicitar el cliente las páginas JSP, y genera en ese momento el HTML. El cliente recibe únicamente el HTML generado, con lo que el proceso del servidor es transparente para él.

Para entenderlo mejor veremos un fichero .jsp como ejemplo

Ejemplo de página JSP. (Fichero saludo.jsp)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ page import = "java.util.Calendar" %>
<html>
  <head>
    <title>Saludo</title>
  </head>
  <body>
    <%
      Calendar ahora = Calendar.getInstance();
      int hora = ahora.get(Calendar.HOUR_OF_DAY);
    %>
    Hola mundo,
    <% if ((hora>20)|| (hora<6)) { %>
      buenas noches
    %>
    else if ((hora>=6)&&(hora<=12)) { %>
      buenos días
    %>
    else { %>
      buenas tardes
    %> } %>
  </body>
</html>
```

La salida de este fichero al navegador sería esta:



Y su estructura HTML no sería la misma que el fichero .jsp si no que sería esta:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

```
<html>
```

```
<head>
```

```
<title>Saludo</title>
```

```
</head>
```

```
<body>
```

```
Hola mundo,
```

```
buenas noches
```

```
</body>
```

```
</html>
```

3.2 Aplicaciones Web en el cliente

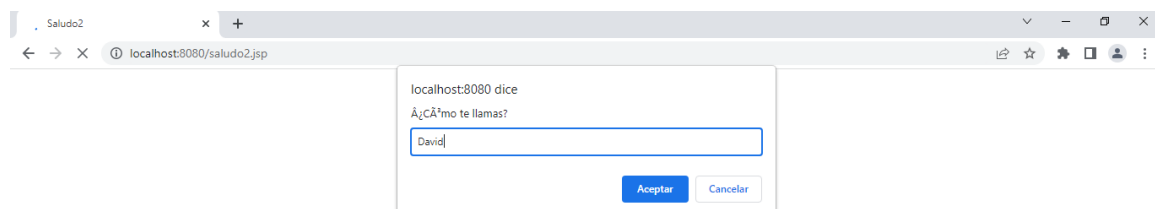
También se puede ejecutar código en el navegador web. En este caso, el navegador recibe el código del programa junto con el código HTML.

Como ejemplo usaremos el siguiente código:

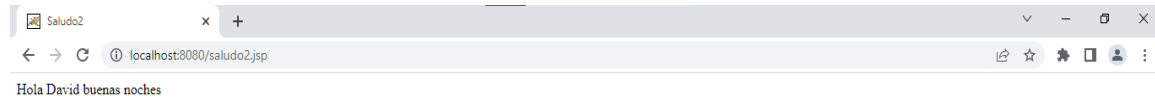
Página JSP con código JavaScript (Fichero saludo2.jsp)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ page import = "java.util.Calendar" %>
<html>
  <head>
    <title>Saludo</title>
  </head>
  <body>
    <%
      Calendar ahora = Calendar.getInstance();
      int hora = ahora.get(Calendar.HOUR_OF_DAY);
    %>
    Hola
    <script language="JavaScript">
      nombre = prompt("¿Cómo te llamas?");
      document.write(nombre); </script>
    <% if ((hora>20) || (hora<6)) { %>
      buenas noches
    <% }
    else if ((hora>=6) && (hora<=12)) { %>
      buenos días
    <% }
    else { %>
      buenas tardes
    <% } %>
  </body>
</html>
```

La parte sombreada del código hará que nos aparezca una ventana emergente dónde se nos pedirá nuestro nombre



Al darle a aceptar, desaparecerá y se nos mostrará la página con el datos que hemos enviado



3.3 Páginas estáticas

Para comprobar que aunque Tomcat en un servidor para Aplicaciones, también puede emplearse para servir páginas estáticas, usaremos un fichero *html* como ejemplo

