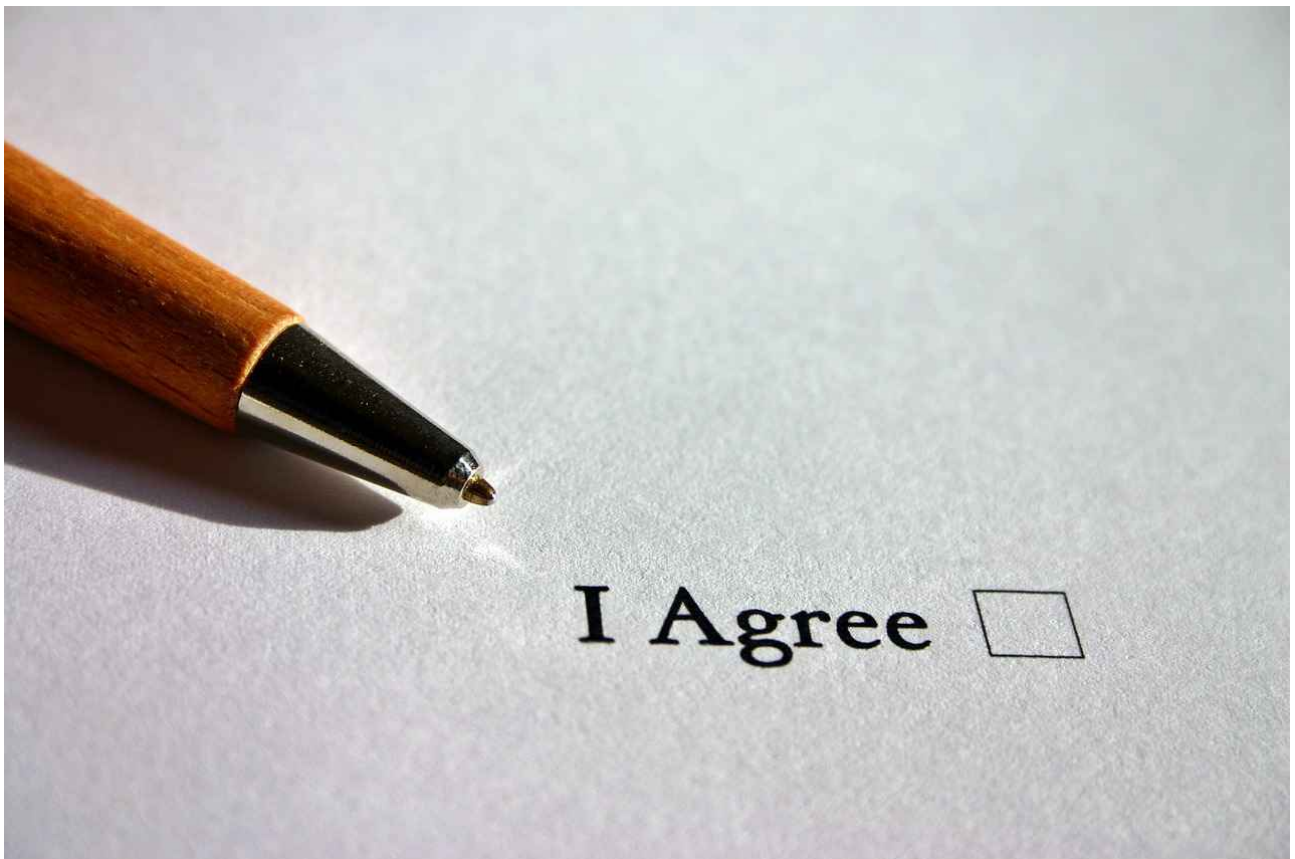


DESARROLLO EN ENTORNO CLIENTE/SERVIDOR

**UD4: PORTFOLIO - ESTRUCTURAS DE DATOS Y
FORMULARIOS**



UD4 - Estructuras de datos y formularios

1. INTRODUCCIÓN.....	3
2. EVALUACIÓN.....	3
3. GESTIÓN DEL ESTADO.....	4
3.1. Cookies persistentes.....	5
3.2. Sesiones del navegador.....	7
3.3. Sesiones de usuario.....	9
3.4. Seguridad.....	12
3.5. Otros usos de las cookies.....	12
4. FORMULARIOS.....	12
4.1. Teoría.....	12
4.2. Proyecto.....	13
4.2.1. Nombre y apellidos.....	16
4.2.2. e-mail.....	21
4.2.3. Teléfono.....	22
4.2.4. Particular o empresa.....	23
4.2.5. Descripción.....	24
4.2.6. Fichero.....	24
4.2.7. Página de confirmación de envío.....	28
4.2.8. Administración de contactos.....	29

1. INTRODUCCIÓN

Sin duda uno de los elementos que posibilitan el dinamismo en las aplicaciones web son los formularios, que permiten al usuario interactuar con la [lógica de negocio](#) implementada en la parte servidor. Se trata de un elemento fundamental para que el usuario participe en el proceso, y es por ello que se debe prestar especial atención a su diseño con el fin de asegurar una correcta **usabilidad y accesibilidad**. Esto está muy ligado al diseño de interfaces, que se abordará en un módulo separado.

En esta unidad veremos, primeramente, cómo llevar a cabo la gestión de estado en aplicaciones web (basadas en MVC), para a continuación abordar la creación de formularios en PHP, los diferentes métodos HTTP que soportan el envío de datos al servidor, controles de validación, seguridad, así como la recuperación de datos.

Una vez visto lo básico sobre formularios en PHP, enriqueceremos nuestra aplicación web del portfolio mediante diferentes funcionalidades basadas en formularios: contacto, login/logout, creación y mantenimiento de proyectos, listados, etc.

2. EVALUACIÓN

El presente documento, junto con sus correspondiente boletín de actividades (publicado adicionalmente), cubren los siguientes criterios de evaluación:

RESULTADOS DE APRENDIZAJE	CRITERIOS DE EVALUACIÓN
RA3. Escribe bloques de sentencias embebidos en lenguajes de marcas, seleccionando y utilizando las estructuras de programación.	e) Se han utilizado formularios web para interactuar con el usuario del navegador web. f) Se han empleado métodos para recuperar la información introducida en el formulario.
RA4. Desarrolla aplicaciones Web embebidas en lenguajes de marcas analizando e incorporando funcionalidades según especificaciones.	a) Se han identificado los mecanismos disponibles para el mantenimiento de la información que concierne a un cliente web concreto y se han señalado sus ventajas. b) Se han utilizado mecanismos para mantener el estado de las aplicaciones web. c) Se han utilizado mecanismos para almacenar información en el cliente web y para recuperar su contenido.

3. GESTIÓN DEL ESTADO

Como se comentó en la UD1, la base de la comunicación en las aplicaciones web es Internet, donde la pila de protocolos utilizada es TCP/IP. En el último nivel, el de aplicación, el protocolo utilizado es HTTP (o HTTPS). Este protocolo fue el elegido por su idoneidad para este tipo de comunicaciones, pero al mismo tiempo también nos limita en una serie de aspectos.

El punto fundamental que nos va a impactar en el desarrollo de aplicaciones web es su **falta de control de estado**. Es decir: el cliente y el servidor conocen de la existencia del otro solo cuando se produce una petición al servidor, por parte del cliente; fuera de esa interacción, no se conserva ningún tipo de información entre una petición y la siguiente. El servidor puede estar recibiendo peticiones desde distintos navegadores, en máquinas diferentes, y no tiene forma de distinguir unas de las otras. Para el servidor son todas iguales.

Esto nos impide particularizar la experiencia del usuario en nuestro sitio web dependiendo de acciones tomadas con anterioridad. Por ejemplo: podríamos dar una serie de mensajes de bienvenida o iniciación en nuestra aplicación web la primera vez que accede el usuario; una vez éste da su conformidad o da por aceptada la información, ya no se volverían a mostrar estos mensajes (la aplicación sería capaz de recordar una acción anterior del usuario). Estos mensajes podrían ser un mini-tutorial, mensajes de aceptación de cookies, etc.

En definitiva, el protocolo HTTP deja al programador/a la tarea de implementar este control de estado entre peticiones.

Existen varias estrategias para poder gestionar el estado entre peticiones HTTP, dependiendo de la arquitectura web que estemos utilizando. A continuación se muestra un cuadro resumen de las alternativas que utilizaremos a lo largo del curso, desde el punto de vista del desarrollo en el lado servidor, dependiendo de la arquitectura web con la que estemos trabajando:

Tipo de proyecto	Gestión del estado
PHP (MVC)	Se realizará mediante dos objetos de PHP: <ul style="list-style-type: none">• Cookies• Sesiones

Django (MVC)	También se utilizará el concepto de cookie y sesión, aunque con las herramientas propias del framework.
Django Rest Framework (API Rest)	Se llevará a cabo en la <u>parte cliente</u> (ya que los servicios REST no conservan estado), dependiendo de si se está utilizando o no un framework. En caso de utilizar framework existen módulos específicos para esta funcionalidad (Redux para React.js, o Vuex para Vue.js, por ejemplo).

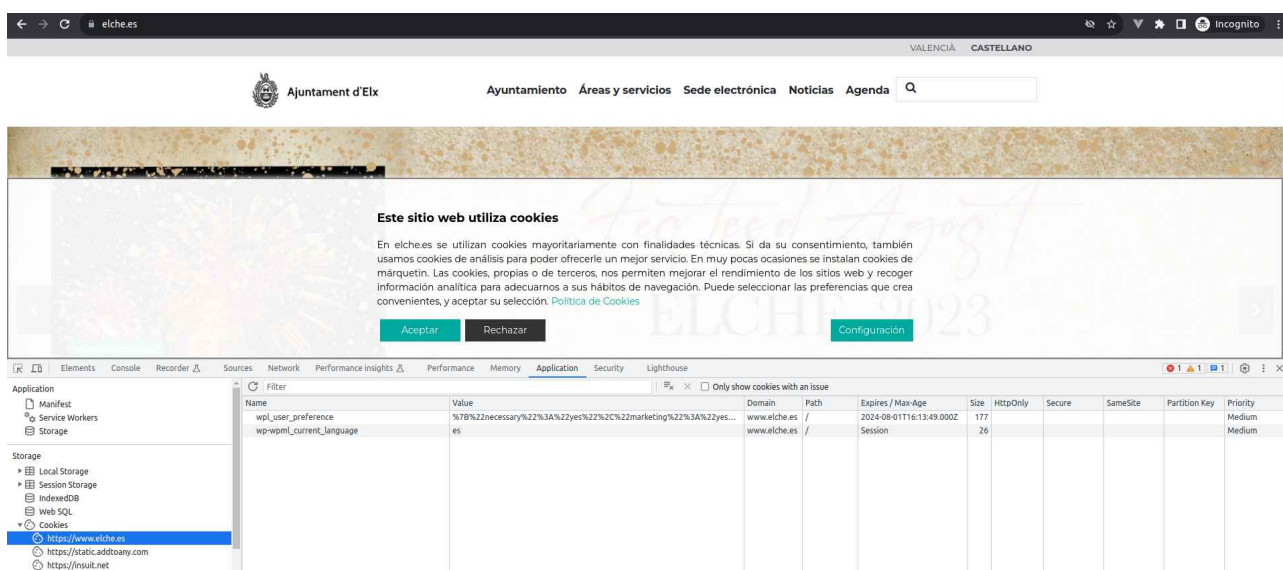
En los siguientes subapartados se detalla el uso de cookies y sesiones en PHP (el primero de los casos). El resto de posibilidades se tratará en las unidades correspondientes (ya pertenezca a la parte cliente, o servidor).

3.1. Cookies persistentes

Las llamadas cookies son en realidad ficheros de pequeño tamaño que el servidor envía al cliente (navegador web) y se almacenan en el dispositivo del cliente, agrupándolas por sitio web.

Si visitamos un sitio web que requiere almacenar cookies en el dispositivo cliente se debe advertir al usuario de ello y del propósito de dichas cookies.

Si consultamos por ejemplo el sitio web del [ayuntamiento](#) de Elche, podremos ver, mediante las herramientas de desarrollador del navegador (depende del navegador que estemos utilizando) las cookies que se han almacenado. Podemos ver en el panel de la izquierda las cookies guardadas, por cada sitio web:



Ajuntament d'Elx | Ayuntamiento | Áreas y servicios | Sede electrónica | Noticias | Agenda

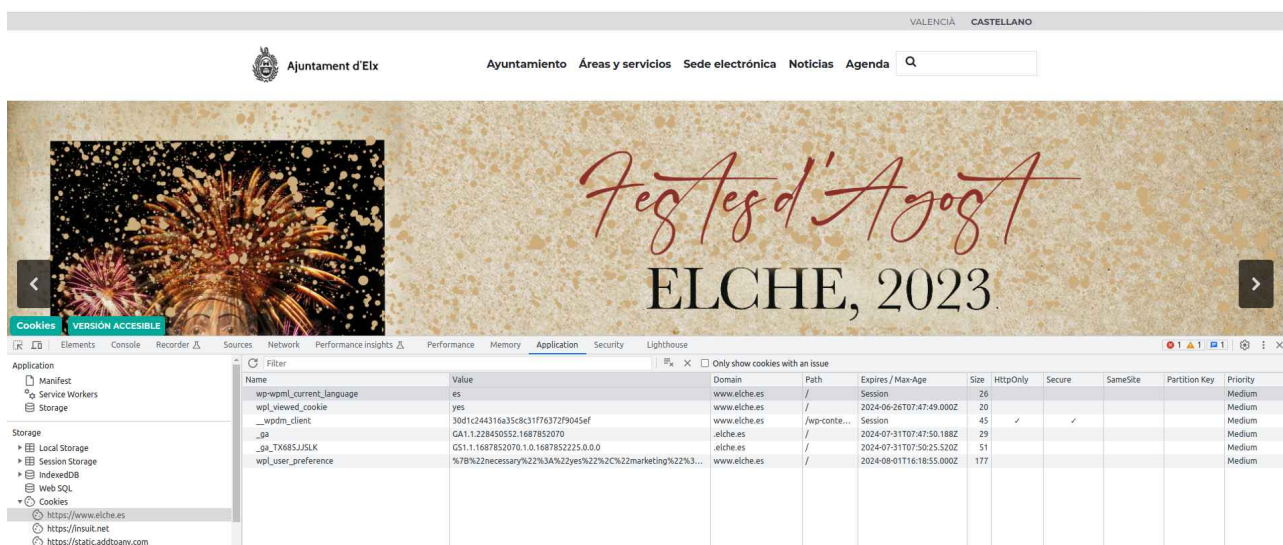
Este sitio web utiliza cookies

En elche.es se utilizan cookies mayoritariamente con finalidades técnicas. Si da su consentimiento, también usamos cookies de análisis para poder ofrecerle un mejor servicio. En muy pocas ocasiones se instalan cookies de marketing. Las cookies, propias o de terceros, nos permiten mejorar el rendimiento de los sitios web y recoger información analítica para adecuarnos a sus hábitos de navegación. Puede seleccionar las preferencias que crea convenientes, y aceptar su selección. [Política de Cookies](#)

Acceptar | Rechazar | Configuración

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Partition Key	Priority
wp_user_preference	%7B%22necessary%22%3A%22yes%22%2C%22marketing%22%3A%22yes%22%7D	www.elche.es	/	2024-08-01T16:13:49.000Z	177					Medium
wp-wpml_current_language	es	www.elche.es	/	Session	26					Medium

Por ser la primera vez que accedemos al sitio web, nos aparece el mensaje de aceptación de cookies. En este caso tenemos solo 2 cookies, pero si aceptamos el aviso, nos aparecerán unas cuantas más:



Ajuntament d'Elx | Ayuntamiento | Áreas y servicios | Sede electrónica | Noticias | Agenda

Festes d'Agost ELCHE, 2023

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Partition Key	Priority
wp-wpml_current_language	es	www.elche.es	/	Session	26					Medium
wpml_client	30d1c244316a35c8c31f76372f9045ef	www.elche.es	/wp-conte...	Session	45	✓		✓		Medium
_ga	GA1.1.22845052.1687852070	elche.es	/	2024-07-31T07:47:50.188Z	29					Medium
_ga_TK685JSLK	GS1.1.1687852070.1.0.1687852225.0.0.0	elche.es	/	2024-07-31T07:50:25.520Z	51					Medium
wp_user_preference	%7B%22necessary%22%3A%22yes%22%2C%22marketing%22%3A%22yes%22%7D	www.elche.es	/	2024-08-01T16:18:55.000Z	177					Medium

En el caso de elche.es, podemos ver que se guardan, entre otras, las cookies "wpl_viewed_cookie" y "wp-wpml_current_language". Según el prefijo de estos nombres, todo apunta a que son cookies creadas por el CMS (Content Management System) llamado Wordpress, en el que se basa este sitio web (lo cual se puede ver claramente al analizar el sitio

web con la herramienta [builtwith](#)).

Si, el lugar de escoger castellano como idioma elegimos valenciano, veremos que el valor de la cookie "wp-wpml_current_language" cambia al valor "va". Por tanto, esta cookie le sirve al servidor para saber en qué idioma prefiere el usuario que se le presente la interfaz web.

Fijémonos en lo siguiente: la columna "Expires / Max Age" contiene valores de fechas con marcas de tiempo específicas, que indican cuándo dejarán de ser válidas las cookies, excepto para dos de ellas, en las que indica "Session". El significado de este valor lo analizaremos en el siguiente punto sobre sesiones del navegador.

Ahora, cada vez que naveguemos por este sitio web, el servidor consultará el valor de las cookies del cliente, y podrá dar una respuesta personalizada en función de sus valores.

Para saber lo básico sobre cookies en PHP, revisamos este [enlace](#) de W3Schools, que utilizaremos posteriormente en las actividades.

Las cookies creadas según el enlace anterior tienen una fecha de caducidad determinada. A estas cookies se les denomina **cookies persistentes**. En el siguiente apartado conoceremos las **cookies de sesión**.

3.2. Sesiones del navegador

El concepto de sesión de navegador identifica una forma de agrupar determinadas variables que están asociadas a un navegador web concreto, y durante el tiempo en que ese navegador está abierto. Si abrimos una pestaña nueva en el mismo navegador, se compartirá la misma sesión, pero si abrimos otro navegador dentro del mismo dispositivo (o el mismo, pero en modo incógnito), se creará una sesión diferente para ese navegador.

Pero, ¿cómo es posible que el servidor pueda saber cuándo iniciar una nueva sesión y cuándo no? La respuesta se resuelve mediante el uso de cookies: al visitar un sitio web mediante un navegador, el servidor recibe la petición HTTP y analiza si existe una cookie de sesión; si no

existe una cookie de sesión genera un ID nuevo de sesión y lo envía al navegador, donde se almacena. La siguiente vez que se navega dentro de ese sitio web (y sin haber cerrado el navegador) el servidor consulta el valor de las cookies, entre las cuales está el ID de sesión. El servidor recibe el ID de sesión, y recupera las variables asociadas. Esto se muestra en el siguiente esquema:



La diferencia respecto a las cookies persistentes es que la información sobre las diferentes sesiones de navegador se guarda en el servidor, no en el lado cliente.

Vamos a averiguar un poco más sobre las cookies de sesión del sitio web que estábamos probando. Según la captura de pantalla del apartado anterior, las dos cookies de sesión almacenadas son:

- `wp-wpml_current_language`: identifica las preferencias del usuario respecto al idioma. Tiene, por defecto, el valor "es".
- `__wpdm_client`: es una cookie de sesión utilizada por el plugin "Wordpress Download Manager", que le permite compartir funcionalidades al navegar entre las páginas web del sitio.

Centrémonos en la primera de ellas. Su comportamiento es:

- Al abrir el navegador y visitar el sitio web, siempre tiene el valor "es".
- Al cambiar a "Valenciano", toma el valor "val".
- Al cerrar el navegador y volverlo a abrir, tiene el valor "es".

Estos pasos implican que, cada vez que el usuario accede al sitio web, después de haberlo cerrado, ha de volver a elegir su preferencia de idioma, si no coincide con el valor por defecto.

Preguntémonos: ¿es una cookie de sesión lo más adecuado en este caso?

Para lograr una mejor experiencia de usuario, necesitaríamos que la preferencia del idioma durase más allá de la sesión del navegador. En este caso, sería mejor implementar la cookie de idioma mediante una cookie persistente (con una fecha de expiración muy alejada en el tiempo), y no una cookie de sesión.

Incluso podríamos llegar a almacenar esta preferencia del usuario en una base de datos, si el usuario hubiese estado identificado mediante un proceso de autenticación (que no es el caso porque el sitio web en cuestión está dedicado a usuarios anónimos). Analizaremos este caso en el siguiente apartado.

En este [enlace](#) se muestra una introducción al comportamiento general de las cookies de sesión.

En clase, revisamos mediante [W3Schools](#) cómo gestionar cookies de sesión con PHP.

3.3. Sesiones de usuario

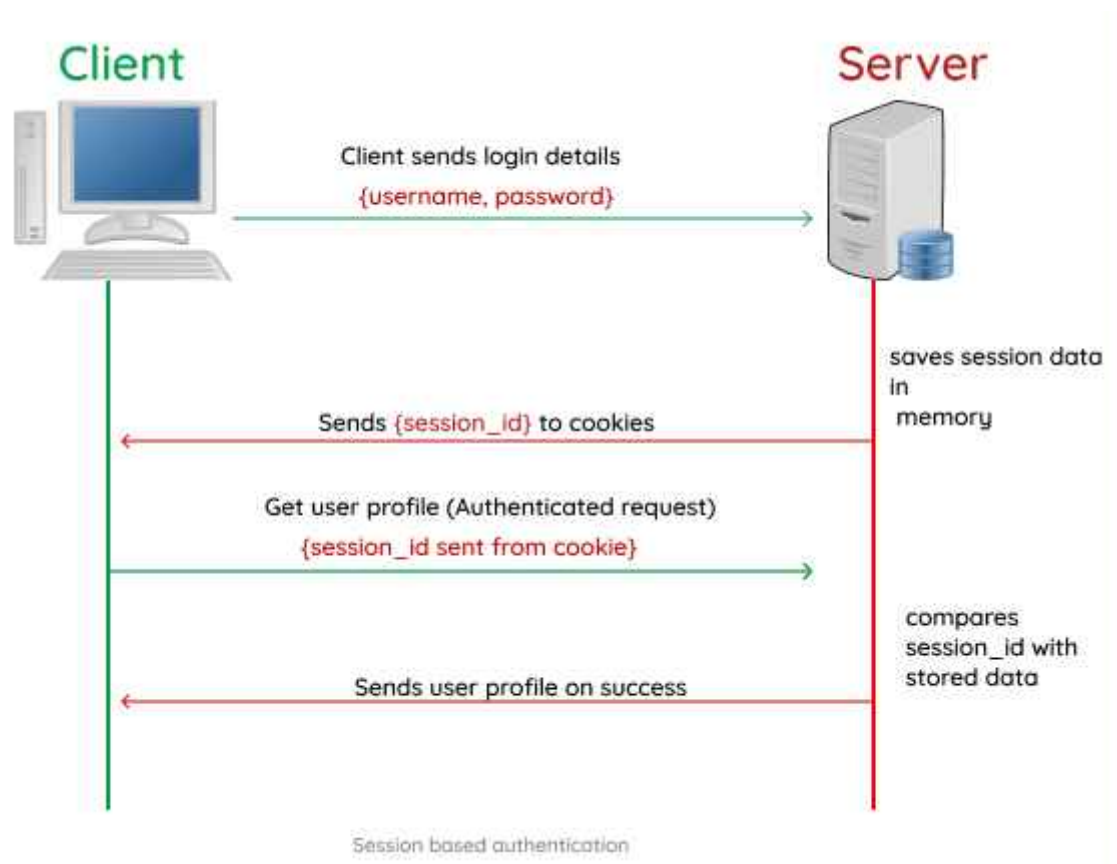
El siguiente paso en el mantenimiento del estado de una aplicación web sería poder identificar de forma concreta qué usuario está accediendo a nuestra aplicación, y particularizar determinados aspectos dependiendo de sus preferencias. Para ello, deberíamos primero poder verificar que el usuario es quien dice ser, para así entonces asociarle determinados parámetros.

Por tanto, una sesión de usuario se establece cuando el usuario completa exitosamente el proceso de autenticación en el sistema. Tras este proceso, el usuario obtiene acceso a una serie

de privilegios a los que no tenía acceso antes de proporcionar sus credenciales, y estos privilegios tendrán efecto durante un tiempo determinado por el servidor. Estos privilegios pueden estar agrupados en conjuntos de privilegios llamados **perfiles de usuario**: en la parte cliente, los perfiles de usuario pueden ayudar a conformar la interfaz de usuario (mostrar u ocultar opciones dependiendo de los privilegios), y en la parte servidor determinan las acciones de la lógica de negocio que el usuario puede llevar a cabo.

En este intervalo de tiempo, el usuario puede ir eligiendo opciones a través de la aplicación para personalizar su experiencia (como por ejemplo, el idioma), y qué mejor lugar para almacenar esas preferencias que una base de datos en el lado servidor, donde la información no está expuesta a posibles vulnerabilidades del lado cliente.

Aún así, ¿necesitamos almacenar algo en el lado cliente? Al menos, deberíamos guardar algo que identifique al usuario, o más bien, algo a través de lo cual el lado servidor pueda recuperar la información del usuario. Esto se consigue a través del **identificador de sesión**, que se genera en el lado servidor y se guarda en el lado cliente tras el proceso de autenticación. Con solo el identificador de sesión, el lado servidor es capaz de recuperar todo lo que necesita sobre el usuario y sus preferencias. El esquema general de estos pasos se ve en la siguiente figura:



NOTA: El esquema anterior aplica a arquitecturas web basadas en el MVC, en el caso de servicios REST el esquema de autenticación es diferente.

La implementación de este proceso se suele llevar a cabo mediante extensiones de un **framework** (como Laravel, Node.js, Django...) y desarrollarlo desde cero puede ser problemático, ya que la solución puede no ser lo suficientemente robusta como para abarcar todos los posibles riesgos de seguridad. Normalmente, las soluciones proporcionadas por los frameworks son lo suficientemente completas y han sido probadas por la comunidad de desarrolladores y usuarios, por lo que suelen ser suficientemente fiables como para optar por implementarlas desde 0.

En las actividades realizaremos un proceso de login rudimentario, a modo de ejemplo, pero en ningún caso será la base para una aplicación web a utilizar en un sistema real.

3.4. Seguridad

Además de los riesgos intrínsecos de exponer un equipo servidor en Internet, cualquier información que viaje por la red y pueda ser almacenada en equipos cliente es susceptible también de ser capturada y utilizada para fines maliciosos.

El uso de las cookies desde el punto de vista de la ciberseguridad es un tema a tratar de forma separada, y fuera de los objetivos de este curso. En este [enlace](#) se presenta una introducción a los riesgos de seguridad que implica el uso de las cookies. En este [enlace](#) puedes profundizar en el tipo de ataques que intentar hacer un uso malicioso de las cookies de sesión de usuario.

3.5. Otros usos de las cookies

Las cookies pueden ser utilizadas para guardar hábitos de navegación del usuario o para mantener identificada las sesiones de los usuarios, como hemos visto hasta ahora.

Pero otra aplicación de las cookies se encuentra en el ámbito de recopilación de datos de navegación de los usuarios, rastreo, análisis estadísticos... En este [enlace](#) encontrarás una visión más amplia del uso de las cookies en las aplicaciones web actuales, que excede a los objetivos del presente curso.

4. FORMULARIOS

4.1. Teoría

En el siguiente [enlace](#) encontrarás lo básico sobre los formularios en PHP.

También revisamos la sección de [PHP Forms](#) de W3CSchools.

Revisamos los dos enlaces en clase.

Tras la lectura de los enlaces y a la vista de los ejemplos podemos extraer el funcionamiento general de una página que contenga un formulario, que va a ser invocada mediante dos métodos:

- **GET**, será el caso en que se llegue a la página mediante una navegación por parte del usuario a través de los enlaces del sitio web. Se pueden dar dos casos:
 - El formulario se encuentra vacío, no es necesario recuperar información para pre-

informar los campos del formulario, y el propósito es **crear** datos nuevos. Por lo general, no sería necesario una lógica que se ejecutase ante una llamada GET al formulario.

- Se llega al formulario pulsando un elemento en una lista anterior, de forma que lo que se pretende es **modificar** los datos que se han consultado. En este caso, se necesita una lógica que recupere información de una fuente de datos (una base de datos es lo más usual). Lo más usual es que se envíe un ID a través de la URL mediante el método GET, y la página del formulario tome ese ID, consulte en la base de datos, y pre-informe los campos del formulario con la información recuperada, para poder ser posteriormente modificada mediante POST.
- **POST**, se trata del caso en que se haya pulsado sobre el botón tipo "submit" del formulario. Normalmente un formulario invoca al propio script PHP en el que se encuentra, porque en dicho script se incluye la lógica a aplicar cuando el método es POST. Por tanto, la página se invoca a sí misma para poder ejecutar otra parte de su lógica (validación y almacenamiento en BBDD, normalmente) perteneciente al método POST.

NOTA: El método GET no es adecuado para enviar datos desde un formulario al servidor, ya que los parámetros se envían por URL y pueden ser capturados, lo que implica un problema de seguridad.

Se podría decir que un script PHP con formulario está preparado para estas situaciones: el acceso en modo consulta al formulario (normalmente por navegación) y la sumisión de su formulario para el procesamiento de la información introducida o modificada.

4.2. Proyecto

En este paso del proyecto vamos a crear un formulario de contacto, para que potenciales empresas o clientes nos puedan contactar para proponernos un proyecto o una oferta de trabajo.

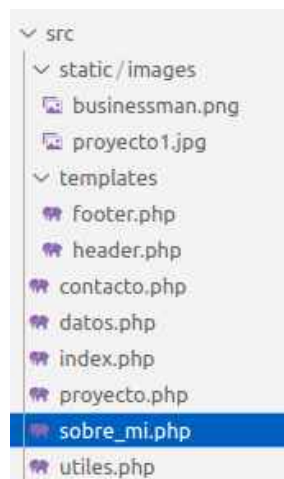
Ahora que ya sabemos crear formularios, vamos a implementar una verdadera página de

contacto. Hasta ahora mostrábamos una ficha de autor en contacto.php, pero sería más conveniente reestructurar la aplicación en este sentido.

Primero vamos a crear una nueva opción de menú que sea "SOBRE MÍ", donde vamos a mover el contenido actual de contacto.php, quedando el menú de la forma:



Creamos también un fichero sobre_mi.php que contenga lo que antes contenía contacto.php:



Y añadimos un enlace (en forma de botón) en la página SOBRE MÍ que nos lleve a CONTACTO, quedando el código del siguiente modo:

```

1  <?php include("templates/header.php"); ?>
2
3  <div class="container">
4      <h2 class="mb-5">SOBRE MÍ</h2>
5      <div class="row">
6          <div class="col-md">
7              
8          </div>
9          <div class="col-md">
10             <h3>Nombre y apellidos</h3>
11             <p>Ciclo Superior DAW.</p>
12             <p>Apasionado del mundo de la programación en general, y de las tecnologías web en particular.</p>
13             <p>Si tienes cualquier tipo de pregunta, contacta conmigo por favor.</p>
14             <p>Teléfono: 87654321</p>
15             <a href="/contacto.php" class="btn btn-primary">CONTACTAR</a>
16         </div>
17     </div>
18 </div>
19
20 <?php include("templates/footer.php"); ?>

```

(Se han realizado cambios en las líneas 4 y 15)

ACTIVIDAD: añade un enlace a contacto.php en el footer de la página, en la sección "Contacte con nosotros".

Es el momento de vaciar contacto.php:

```

1  <?php include("templates/header.php"); ?>
2
3  <div class="container">
4      <h2 class="mb-5">Contacto</h2>
5      <div class="row">
6          <div class="col-md">
7              <div class="form">
8              </div>
9          </div>
10 </div>
11
12 <?php include("templates/footer.php"); ?>

```

Ahora nos planteamos qué campos necesitamos para que un posible usuario quiera contactar con nosotros. Se proponen los siguientes:

Campo	Requisitos
Nombre y apellidos	Obligatorio.
e-mail	Obligatorio, y ha de ser un e-mail correcto.
Número de teléfono	Obligatorio, y ha de ser un número correcto.
Particular/empresa	Obligatorio, campo multiselección.
Descripción	Obligatorio, texto libre.

Fichero	Opcional, pero si se sube uno ha de ser en formato .pdf.
---------	--

Vamos a ir paso a paso, insertando cada uno de los campos del formulario. Para dar estilo a los campos vamos a utilizar los [formularios](#) de Bootstrap.

4.2.1. Nombre y apellidos

Empezamos con el propio formulario y con el campo "Nombre y apellidos". Insertamos el siguiente código en contacto.php:

```
<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>" method="POST">
  <div class="mb-3 col-sm-6 p-0">
    <label for="nombreApellidosID" class="form-label">Nombre y apellidos</label>
    <input type="text" name="nombreApellidos" class="form-control" id="nombreApellidosID"
placeholder="Su nombre y apellidos">
  </div>
  <button type="submit" class="btn btn-success">Enviar</button>
</form>
```

Este campo ha de ser obligatorio. ¿Con este código cumplimos las condiciones? La respuesta es no. ¿Podríamos utilizar el atributo "required" de HTML5 para solucionar esto? En principio este atributo haría este campo del formulario obligatorio, pero tendría sus inconvenientes. Vamos a añadirlo:

```
<input type="text" name="nombreApellidos" class="form-control" id="nombreApellidosID"
placeholder="Su nombre y apellidos" required>
```

Ahora, si pulsamos en el botón Enviar sin introducir ningún texto en el campo obtendremos este mensaje:

Contacto



The screenshot shows a web form titled "Contacto". It has a label "Nombre y apellidos" above a text input field. The input field contains the placeholder text "Su nombre y apellidos". Below the input field is a dark blue button labeled "Enviar". To the right of the button, a yellow tooltip with a red exclamation mark icon displays the message "Please fill out this field."

No parece un mal mensaje de error, en principio, pero no podemos elegir el texto concreto a mostrar al usuario, es un mensaje demasiado genérico, además no sigue los patrones de Bootstrap.

¿Pero cumple su propósito? ¿Realmente obliga al usuario a informar este campo? Un usuario con conocimientos de HTML podría inspeccionar el código y eliminar esta restricción fácilmente con el navegador:



Es decir, si eliminamos este atributo en el código del HTML que reside en nuestro navegador nos podríamos saltar la restricción de obligatoriedad. Por tanto la pregunta es: ¿cómo nos podemos asegurar que realmente se envían los valores correctos desde un formulario independientemente de que la parte cliente se pueda ver manipulada? La respuesta es: en el lado servidor. Es por ello que en ninguno de los ejemplos del apartado de teoría se utiliza este atributo.

Además, podríamos **complementar** todo esto mediante JavaScript y que la interacción con el usuario fuese mucho más adecuada y personalizada.

Al final se produce lo que se llama "doble validación":

- En cliente, como un primer filtro, para controlar mucho mejor la interacción con el usuario.
- En servidor, para asegurarnos que se cumple la lógica de negocio y no se procesan o insertan valores inconsistentes en la base de datos.

La conclusión a extraer de este razonamiento es que es la parte servidor la que ha de asegurar que se implementan las validaciones correspondientes, y la parte cliente se ocupa de la interacción con el usuario. Esto es especialmente crítico en sistemas basados en servicios REST, en los cuales aplicaciones de diferente tipo (aplicaciones web, apps, aplicaciones de escritorio, otros sistemas incluso...) utilizan los mismos servicios web de un mismo servidor. Ha de ser la parte servidor quien asegure que los datos son acorde a las especificaciones, así como sanitizar los datos introducidos por parte del cliente, para evitar posibles situaciones no deseadas.

Esta metodología se sigue independientemente de la tecnología y el modelo de programación que estemos utilizando, y viene provocada por las características de la arquitectura web utilizada (cliente/servidor) y la naturaleza de la comunicación entre ambas partes (protocolo HTTP).

Por todo ello, vamos a introducir la lógica de validación mediante PHP. Primero vamos a definir en `utils.php` la función de limpieza `test_input` que se discute en este [enlace](#):

```
function test_input($data) {  
    $data = trim($data);  
    $data = stripslashes($data);  
    $data = htmlspecialchars($data);  
    return $data;  
}
```

Importamos `utils.php` en `contacto.php`.

Vamos también a definir una variable en `contacto.php` que contenga un mensaje de error definido por nosotros, llamada `nameErr`. El código quedaría del siguiente modo:

```
<?php  
$nameErr = "";  
  
if ($_SERVER["REQUEST_METHOD"] == "POST") {  
    if (empty($_POST["nombreApellidos"])) {  
        $nameErr = "Por favor, introduzca nombre y apellidos";  
    }  
}  
?  
>
```

Solo faltaria introducir un mensaje en HTML que se mostrase solo cuando la variable `nameErr` tuviese un valor. Podemos utilizar un texto en rojo para este mensaje, de la forma:

```
<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>" method="POST">
  <div class="mb-3 col-sm-6 p-0">
    <label for="nombreApellidosID" class="form-label">Nombre y apellidos</label>
    <input type="text" name="nombreApellidos" class="form-control" id="nombreApellidosID"
placeholder="Su nombre y apellidos">
    <span class="text-danger"> <?php echo $nameErr ?> </span>
  </div>
  <button type="submit" class="btn btn-success">Enviar</button>
</form>
```

Vamos a probarlo:

Contacto

Nombre y apellidos

Por favor, introduzca nombre y apellidos

Enviar

¿Estamos ya satisfechos con el resultado? Mucho, pero no del todo. Somos bastante perfeccionistas, y nos preguntamos cómo podríamos deshabilitar el botón Enviar hasta que el campo "Nombre y apellidos" no contenga un valor. Esto sería la situación ideal: no dejar enviar nada al servidor que ya sabemos que no es correcto, reducimos el trabajo que el servidor ha de llevar a cabo. Pero, ¿cómo podríamos deshabilitar el botón Enviar cuando se detectase que alguno de los campos del formulario no tiene un valor correcto, antes de enviarlo al servidor? Para esto se hace necesario añadir más interactividad en la parte cliente mediante JavaScript, ya sea mediante librerías específicas o mediante frameworks reactivos (como Vue.js, React, Angular, Svelte...). Todo esto se complementará con la parte de Desarrollo en Cliente.

¿Nos queda algo por hacer? En principio todo parece funcional, hace lo que debe hacer, pero: ¿es seguro? La respuesta es: se puede mejorar la seguridad. ¿Por qué no es del todo seguro? porque

no estamos controlando el contenido que se está enviando en el campo, se puede enviar cualquier tipo de caracter, código malicioso, etc. Entonces, ¿cómo hacemos para que un nombre y apellidos parezcan realmente un nombre y apellidos y nada más? Podemos aplicar una **expresión regular** al texto que se envíe en el formulario completando las validaciones del lado servidor, de la forma:

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {  
    if (empty($_POST["nombreApellidos"])) {  
        $nameErr = "Por favor, introduzca nombre y apellidos";  
    } else {  
        $name = test_input($_POST["nombreApellidos"]);  
        if (!preg_match("/^[a-zA-Z' ]*$/", $name)) {  
            $nameErr = "Solo se permiten letras y espacios.";  
        }  
    }  
}
```

Si no recuerdas qué son las expresiones regulares, lo puedes revisar en este [enlace](#). Siendo pragmáticos, [aquí](#) tienes una recopilación de algunas expresiones regulares muy utilizadas. Normalmente una correcta búsqueda en Internet te dará la expresión regular que necesitas. Podríamos complicar la expresión regular un poco más para que validase que se envía un nombre y dos apellidos, como se discute en este [enlace](#), pero por el momento lo vamos a dejar como está.

NOTA IMPORTANTE: en HTML, mediante el atributo [pattern](#) también podemos añadir una expresión regular a un elemento input, pero tendríamos el mismo problema que con el atributo required. Por tanto es más conveniente hacer esta validación en el lado servidor.

Ahora, si intentamos introducir cualquier carácter que no sea una letra o un espacio el formulario nos devolverá un error:

Contacto

Nombre y apellidos

Solo se permiten letras y espacios.

Pero estamos notando un comportamiento extraño en nuestro formulario: introducimos un texto inválido y pulsamos en Enviar, obtenemos el mensaje de error correspondiente, ¡pero el texto desaparece! ¿cómo podemos hacer para que se retengan los valores de los campos? Es muy fácil, simplemente hemos de asignar el valor de la variable \$name al atributo value del elemento input:

```
<input type="text" name="nombreApellidos" value="<?php echo $name;?>" class="form-control" id="nombreApellidosID" placeholder="Su nombre y apellidos">
```

De esta forma no desaparecerán los valores del formulario al pulsar sobre Enviar.

4.2.2. e-mail

Primero añadimos el HTML de este campo y vemos cómo queda. Lo situamos debajo del campo

Nombre y apellidos:

```
<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>" method="POST">
  <div class="row">
    <div class="mb-3 col-sm-6 p-0">
      <label for="nombreApellidosID" class="form-label">Nombre y apellidos</label>
      <input type="text" name="nombreApellidos" value="<?php echo $name;?>" class="form-control" id="nombreApellidosID" placeholder="Su nombre y apellidos">
      <span class="text-danger"> <?php echo $nameErr ?> </span>
    </div>
  </div>
  <div class="row">
    <div class="mb-3 col-sm-6 p-0">
      <label for="emailID" class="form-label">e-mail</label>
      <input type="text" name="email" value="<?php echo $email;?>" class="form-control" id="emailID" placeholder="Su e-mail">
      <span class="text-danger"> <?php echo $emailErr ?> </span>
    </div>
  </div>
  <button type="submit" class="btn btn-success">Enviar</button>
</form>
```

Es de hacer notar que hemos definido el elemento input como tipo "text", en lugar de "email". Esto es precisamente para evitar las validaciones del navegador, como la del atributo required o el pattern.

A continuación añadimos el fragmento de código PHP que se va a encargar de validar el e-mail:

```
if (empty($_POST["email"])) {
    $emailErr = "Por favor, introduzca su e-mail.";
} else {
    $email = test_input($_POST["email"]);
    if (!preg_match("/^([<>()\\[\\].,;:\\s@\\"]+\\.([<>()\\[\\].,;:\\s@\\"]+)*)(\\\".+\\\")@((([<>()\\[\\].,;:\\s@\\"]+\\.)+[<>()\\[\\].,;:\\s@\\"]{2,})$)/", $email)) {
        $emailErr = "Introduzca un e-mail válido.";
    }
}
```

Como vemos, la expresión regular ha cambiado. El valor de la expresión regular se ha extraído de este [artículo](#) (en el que se muestran tres posibilidades).

También existe la posibilidad de utilizar la función filter_var de PHP, como se muestra en el [ejemplo](#) de W3CSchools.

Cualquiera de las dos formas es efectiva.

4.2.3. Teléfono

Con el campo teléfono ocurre parecido al de email. Añadiremos una expresión regular basada en este enlace, para validar números de teléfono:

```
/^[9|6]{1}([d]{2}[-]*){3}[d]{2}$/
```

ACTIVIDAD: Creamos el campo e-mail con la expresión regular anterior. Queremos que se sitúe a la derecha del e-mail, para ello utilizamos el siguiente bloque div:

```
<div class="row">
  <div class="mb-3 col-sm-6 p-0">
    <!-- AQUÍ VA EL HTML DEL EMAIL -->
  </div>
  <div class="mb-3 pl-2 col-sm-6 p-0">
    <!-- AQUÍ VA EL HTML DEL TELÉFONO -->
  </div>
</div>
```

Completa el resto del código.

También podemos utilizar como type "tel", como se muestra en este [ejemplo](#).

4.2.4. Particular o empresa

Vamos a implementar esta opción del formulario mediante un campo de selección múltiple. Para ello nos basamos en el correspondiente elemento [Radios](#) de Bootstrap. Por ello, el código HTML queda del siguiente modo:

```
<div class="row mb-4">
  <div class="form-check">
    <input class="form-check-input" type="radio" name="tipo" id="particularID" value="particular" <?
php if (isset($tipo) && $tipo=="particular") echo "checked";?>>
    <label class="form-check-label" for="particularID">
      Particular
    </label>
  </div>
  <div class="form-check">
    <input class="form-check-input" type="radio" name="tipo" id="empresaID" value="empresa" <?php
if (isset($tipo) && $tipo=="empresa") echo "checked";?>>
    <label class="form-check-label" for="empresaID">
      Empresa
    </label>
  </div>
  <span class="text-danger"> <?php echo $tipoErr ?> </span>
</div>
```

Y la validación de este campo será la siguiente (aquí no es necesario aplicar una expresión regular):

```
if (empty($_POST["tipo"])) {
  $tipoErr = "Por favor, introduzca el tipo de consulta.";
} else {
  $tipo = $_POST["tipo"];
}
```

El resultado queda del siguiente modo:

Contacto

Nombre y apellidos

e-mail

Teléfono

☐ Particular

☐ Empresa

Enviar

4.2.5. Descripción

El último de los campos será un área de texto, donde el usuario de nuestra aplicación podrá introducir un mensaje libre. Para ello vamos a utilizar el elemento `textarea`, según el fragmento HTML siguiente:

```
<div class="row mb-4">
  <textarea class="form-control" name="mensaje" id="areaTexto" rows="3" placeholder="Escriba su mensaje..."><?php print $mensaje;?>
</textarea>
  <label for="areaTexto" class="form-label">Mensaje</label>
</div>
```

En este caso no habrá validaciones, pero se guardará el valor enviado:

```
if (!empty($_POST["mensaje"])){
    $mensaje = test_input($_POST["mensaje"]);
}
```

4.2.6. Fichero

Es muy común tener un archivo asociado a un objeto de nuestra aplicación. Normalmente tendremos un registro en una tabla de una base de datos, y una de las columnas del registro almacenará el nombre y la ruta al fichero.

La opción más inmediata será almacenar el archivo en nuestro sistema de archivos local, que en este caso será el servidor. Dicho servidor será normalmente una máquina virtualizada, alquilada a un proveedor de hosting (como Amazon Web Services, Digital Ocean, Linode, etc.).

Almacenar los ficheros en el sistema de archivos local supone tener acceso inmediato a los ficheros, pero también presenta una serie de inconvenientes, entre ellos:

- Es necesario dimensionar la memoria del servidor para que se acomode al volumen creciente de ficheros.
- El mismo servidor ha de gestionar la comunicación con los clientes y transferir ficheros. Esto impacta en la velocidad de transferencia de información y procesamiento del servidor.
- Si, por alguna razón, el servidor se vuelve irrecuperable, se perderán también los archivos.
- Un mismo servidor ha de servir archivos a zonas geográficas dispares, con lo que la respuesta del servidor se verá afectada dependiendo de la localización del cliente.

- En una arquitectura con contenedores y con múltiples servidores trabajando como un clúster aumenta la complejidad para almacenar los archivos en los sistemas de archivos locales.

Por todo ello, la tendencia actual es delegar el almacenamiento y recuperación de archivos a un servicio CDN, ofertado por la mayoría de los proveedores de hosting citados anteriormente. Estos archivos pueden ser, entre otros:

- Archivos que forman parte de nuestra base de datos (normalmente asociados a un registro de una tabla).
- Ficheros estáticos, que son aquellos necesarios para la aplicación (ficheros CSS, JavaScript, multimedia...).
- Copias de la base de datos.
- etc.

Como muestra, el servicio [Spaces](#) de Digital Ocean.

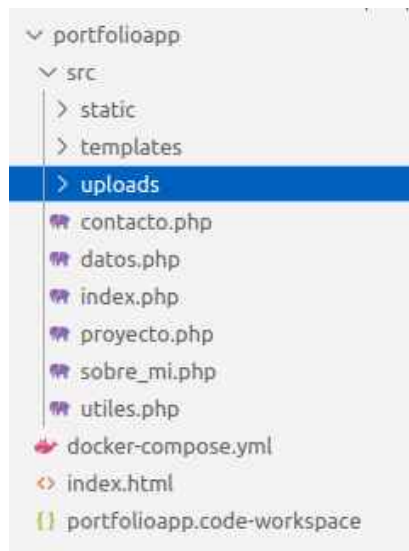
Dicho todo esto, vamos a agregar la funcionalidad para subir ficheros asociados a nuestra página de contacto. Vamos a utilizar el campo de subida de ficheros de Bootstrap. Insertamos el siguiente código HTML bajo el campo de mensaje:

```
<div class="row mb-4">
  <label for="archivoID" class="form-label">Adjuntar archivo</label>
  <input class="form-control" type="file" id="archivoID" name="archivo">
</div>
<span class="text-danger"> <?php echo $archivoErr ?> </span>
<br>
```

NOTA: No vamos a preocuparnos en este momento por los valores por defecto de los textos "Choose file", o "No file chosen". Si quieres probar a cambiarlos, puedes investigar los siguientes enlaces: [Change the "No file chosen"](#), [Change "Choose file"](#).

TEORÍA: Revisamos este [enlace](#), sobre la subida de archivos en PHP.

Vamos a pasar a completar el proyecto. Primero crearemos una carpeta "uploads" en nuestra estructura de directorios, para poder almacenar los ficheros subidos:



Tenemos el directorio en nuestro sistema de archivos, pero tenemos un problema adicional por el hecho de utilizar contenedores Docker si estamos en un sistema Linux: el usuario del servidor web Apache, que está funcionando en el contenedor Docker, no tiene privilegios para escribir ficheros en la ruta `/var/www/html/uploads`, solo para leer. Por eso, hemos de ejecutar el siguiente comando, para que desde el contenedor se puedan escribir archivos en la carpeta "uploads". Desde la terminal, situados **dentro** de la carpeta uploads (CUIDADO):

```
chmod -R 777 ./
```

También lo puedes hacer mediante una ruta absoluta, desde cualquier otro path:

```
chmod -R 777 /<ruta_absoluta>/uploads
```

El siguiente paso es modificar el elemento form, simplemente hemos de añadir el siguiente atributo:

```
<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>" method="POST" enctype="multipart/form-data">
```

Ahora que lo tenemos todo preparado, introducimos el código PHP que procesará el archivo:

```
if (!empty($_FILES['archivo'])) {  
    $nombreArchivo = $_FILES['archivo']['name'];
```

```

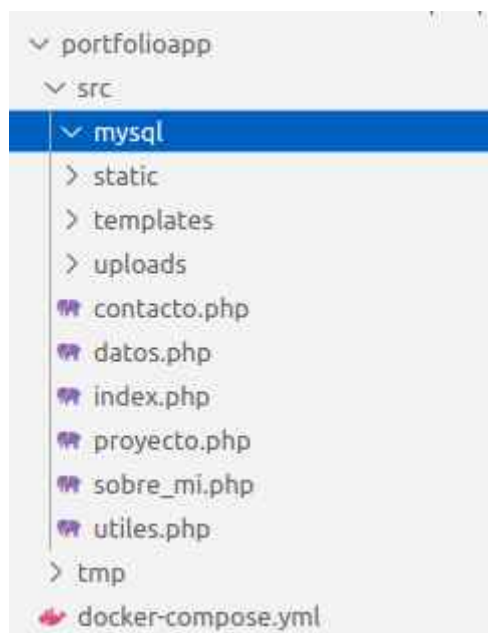
move_uploaded_file($_FILES['archivo']['tmp_name'], "/var/www/html/uploads/{$nombreArchivo}");
if ($nombreArchivo){
    $pathArchivo = "uploads/{$nombreArchivo}";
}
}

```

No podemos mejorar más el comportamiento del botón de la subida de archivos, a no ser que sea mediante CSS y/o JavaScript, según las siguiente [discusión](#).

Pero aún nos queda guardar la información introducida, para poder recuperarla posteriormente. Para ello, guardaremos la información en forma de fichero JSON, para poder recuperarla en otras partes de la aplicación (si fuese necesario), a modo de base de datos.

Por ello, añadimos una nueva carpeta que almacenará ficheros JSON, y la llamaremos mysql:



Al igual que para uploads, necesitaremos permisos de escritura:

```
chmod -R 777 /<ruta_absoluta>/mysql
```

A continuación, al final de la lógica de validación del formulario, insertamos la lógica para insertar el registro en el fichero JSON, si todo ha ido bien:

```

if ($nameErr === "" && $emailErr === "" && $phoneErr === "" && $tipoErr === ""){
    $contacto = [
        "name" => $name,
        "email" => $email,

```

```

    "phone" => $phone,
    "tipo" => $tipo,
    "mensaje" => $mensaje,
    "file" => $pathArchivo,

];

$tempArray = json_decode(file_get_contents('mysql/contactos.json'));
if ($tempArray === NULL){
    $tempArray = [];
}
array_push($tempArray, $contacto);
$contactos_json = json_encode($tempArray);
file_put_contents('mysql/contactos.json', $contactos_json);
}

```

4.2.7. Página de confirmación de envío

Bien, ya tenemos operativo el formulario, pero ahora queremos mostrar un mensaje al usuario que indique que todo ha ido correctamente durante el envío de los datos. Para ello creamos una nueva página llamada `confirma_contacto.php`, con el siguiente contenido:

```

<?php include("templates/header.php"); ?>
<div class="container">
    <div class="alert alert-success mt-5">
        Ha contactado con nosotros satisfactoriamente. En breve nos pondremos en contacto con usted
    </div>
    <div>
        <a class="btn btn-xs btn-info float-right" href="/">
            Volver al inicio
        </a>
    </div>
</div>
<?php include("templates/footer.php"); ?>

```

Pero, ¿cómo hacemos para navegar a esta página desde el código PHP una vez enviamos el formulario satisfactoriamente? En principio existe la directiva [header](#), que se ocupa de esto, pero no siempre funciona correctamente, como se detalla en este [enlace](#).

Es por ello que vamos, de forma excepcional, a utilizar un bloque JavaScript para hacer la redirección:

```

if ($nameErr === "" && $emailErr === "" && $phoneErr === "" && $tipoErr === ""){
    $contacto = [
        "name" => $name,
        "email" => $email,
        "phone" => $phone,

```

```

"tipo" => $tipo,
"mensaje" => $mensaje,
"file" => $pathArchivo,

];
// https://stackoverflow.com/questions/7895335/append-data-to-a-json-file-with-php
$tempArray = json_decode(file_get_contents('mysql/contactos.json'));
if ($tempArray === NULL){
    $tempArray = [];
}
array_push($tempArray, $contacto);
$contactos_json = json_encode($tempArray);
file_put_contents('mysql/contactos.json', $contactos_json);
?>

<script type="text/javascript">
    window.location = "http://localhost/confirma_contacto.php";
</script>

<?php
}

```

Tras enviar un formulario, nos ha de aparecer la página de confirmación:



Si consultamos el fichero contactos.json veremos todos los registros que hemos creado:

```
[{"name": "afa", "email": "n.perez2@edu.gva.es", "phone": "666666666", "tipo": "particular", "mensaje": "aaa", "file": ""}, {"name": "afa", "email": "n.perez2@edu.gva.es", "phone": "666666666", "tipo": "particular", "mensaje": "aaa", "file": ""}, {"name": "afa", "email": "n.perez2@edu.gva.es", "phone": "666666666", "tipo": "particular", "mensaje": "aaa", "file": ""}, {"name": "afa", "email": "n.perez2@edu.gva.es", "phone": "666666666", "tipo": "particular", "mensaje": "aaa", "file": ""}, {"name": "afa", "email": "n.perez2@edu.gva.es", "phone": "666666666", "tipo": "particular", "mensaje": "aaa", "file": ""}, {"name": "asdf", "email": "n.perez2@edu.gva.es", "phone": "666666666", "tipo": "particular", "mensaje": "asdf", "file": ""}, {"name": "asdf", "email": "n.perez2@edu.gva.es", "phone": "666666666", "tipo": "particular", "mensaje": "asdf", "file": ""}, {"name": "asdf", "email": "n.perez2@edu.gva.es", "phone": "666666666", "tipo": "particular", "mensaje": "", "file": ""}, {"name": "asdf", "email": "n.perez2@edu.gva.es", "phone": "666666666", "tipo": "particular", "mensaje": "", "file": ""}, {"name": "asdf", "email": "n.perez2@edu.gva.es", "phone": "666666666", "tipo": "particular", "mensaje": "", "file": ""}, {"name": "asdf", "email": "n.perez2@edu.gva.es", "phone": "666666666", "tipo": "particular", "mensaje": "", "file": ""}, {"name": "asdf", "email": "n.perez2@edu.gva.es", "phone": "666666666", "tipo": "particular", "mensaje": "", "file": ""}, {"name": "asdfsaf", "email": "n.perez2@edu.gva.es", "phone": "666666666", "tipo": "particular", "mensaje": "asdfsdf", "file": ""}, {"name": "asdfsaf", "email": "n.perez2@edu.gva.es", "phone": "666666666", "tipo": "particular", "mensaje": "asdfsdf", "file": ""}]]
```

4.2.8. Administración de contactos

A nuestra aplicación le falta un pequeño detalle: no somos capaces de poder ver los contactos que

nos han realizado. Para ello, vamos a utilizar el menú ADMINISTRACIÓN creado en la unidad anterior. Al pulsar sobre esta opción de menú (recuerda que solo se activa cuando se simula estar autenticado en el sistema), nos llevará a la lista de contactos, que será la página contacto_lista.php, con el siguiente contenido:

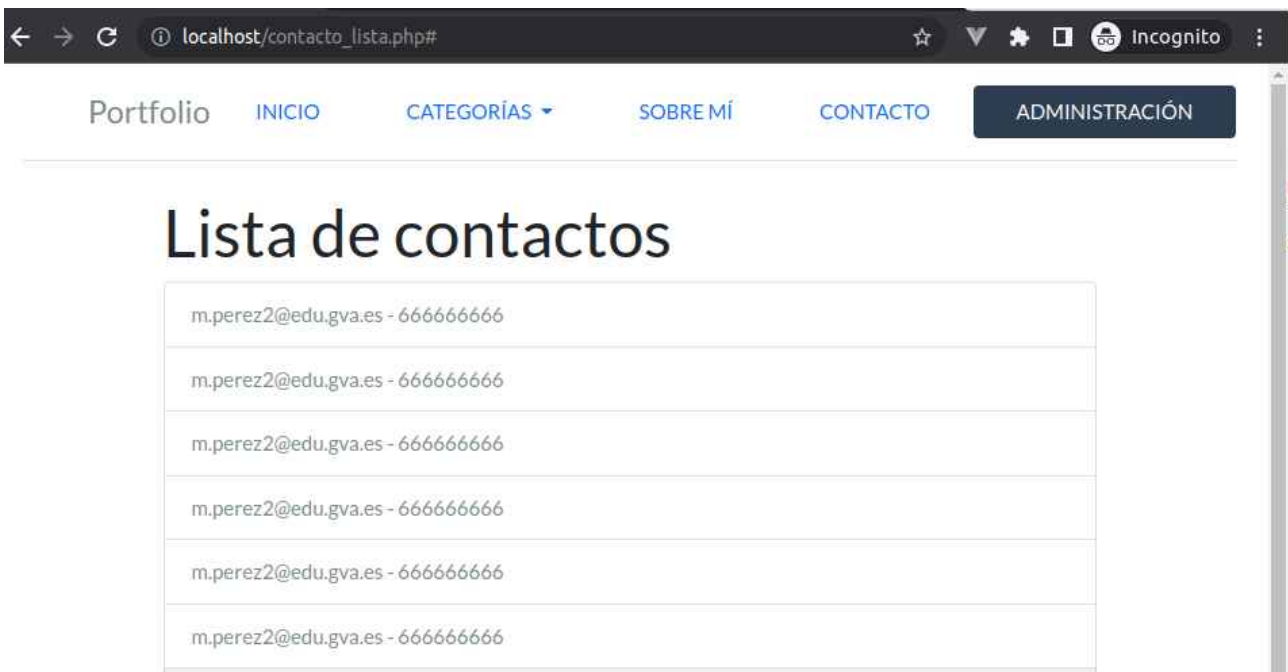
```
<?php include("templates/header.php"); ?>

<?php
    $contactosLista = json_decode(file_get_contents('mysql/contactos.json'), true);
?>

<div class="container mb-5">
    <h1>Lista de contactos</h1>
    <?php if ($contactosLista === NULL) { ?>
        <div class="alert alert-info mt-5">
            Aún no ha sido contactado
        </div>
    <?php } else { ?>
        <div class="list-group">
            <?php foreach ($contactosLista as $contacto): ?>
                <a href="#" class="list-group-item list-group-item-action"><?php echo $contacto['email'] ?> -
<?php echo $contacto['phone'] ?></a>
                <?php endforeach; ?>
            </div>
        <?php } ?>
    </div>

<?php include("templates/footer.php"); ?>
```

Al acceder a ADMINISTRACIÓN, se puede ver la lista de las veces que nos han contactado:



Existen varios aspectos que se podrían mejorar:

1. Además del e-mail y el teléfono, se podría mostrar la fecha en que se ha contactado, y poder ordenar por criterio temporal, o filtrar por e-mail, número de teléfono, etc.
2. Previsualizar parte del mensaje enviado.
3. Al pulsar sobre cada registro, acceder a otra página donde se mostrasen los detalles del contacto, así como el archivo adjunto (si existe). En este caso deberíamos haber introducido un ID en cada contacto, para poder pasarlo por parámetro en la URL a la nueva página, y recuperar el detalle del contacto (como hicimos en el caso de los proyectos).

Vamos a acometer el tercer punto. Para ello necesitamos almacenar un identificador único por cada registro en contactos.json. Lo hacemos con la siguiente línea (en contacto.php):

```
$contacto['id'] = count($tempArray) + 1;
```

```
$tempArray = json_decode(file_get_contents('mysql/contactos.json'));
if ($tempArray === NULL){
    $tempArray = [];
}
```

```
$contacto['id'] = count($tempArray) + 1;
```

```
array_push($tempArray, $contacto);
```

Borramos el fichero contactos.json, si ya contenía valores.

Ya podemos completar el atributo href de contacto_lista.php:

```
<a href="contacto_detalle.php?id=<?php echo $contacto['id'] ?>" class="list-group-item list-group-item-action"><?php echo $contacto['email'] ?> - <?php echo $contacto['phone'] ?></a>
```

Ahora creamos una nueva página llamada contacto_detalle.php, con el siguiente código:

```
<?php include("templates/header.php"); ?>
```

```
<?php
$contacto_id = $_GET['id'];
//El segundo parámetro es para que devuelva un array
$tempArray = json_decode(file_get_contents('mysql/contactos.json'), true);
//Presuponemos que contactos.json no está vacío, pero la URL se puede manipular manualmente
if ($tempArray === NULL){
    $tempArray = [];
} else {
    $contacto_key = array_search($contacto_id, array_column($tempArray, 'id'));
    $contacto = $tempArray[$contacto_key];
}
?>
```

```
<div class="container">
    <h1 class="mb-5">Detalle del contacto</h1>
    <?php if(!empty($contacto)) { ?>
        <p><?php echo $contacto['name'] ?></p>
        <p><?php echo $contacto['phone'] ?></p>
        <p><?php echo $contacto['tipo'] ?></p>
        <p><?php echo $contacto['email'] ?></p>
        <p><?php echo $contacto['mensaje'] ?></p>
        <?php if($contacto['file']) { ?>
            <a href="<?php echo $contacto['file'] ?>" class="btn btn-info mb-4"><i class="fa-solid fa-paperclip"></i> ARCHIVO ADJUNTO</a> <br>
        <?php } ?>
    <?php } else { ?>
        <div class="alert alert-danger mt-5">
            En contacto no existe.
        </div>
    <?php } ?>
```

```
<a href="contacto_lista.php" class="btn btn-secondary"><i class="fa-solid fa-arrow-left mr-2"></i>
Volver</a>
</div>
```

```
<?php include("templates/footer.php"); ?>
```

Probamos a insertar un nuevo contacto con un adjunto, pulsamos sobre él desde la lista de contactos, y accedemos al detalle:

[Portfolio](#)[INICIO](#)[CATEGORÍAS ▾](#)[SOBRE MÍ](#)[CONTACTO](#)[ADMINISTRACIÓN](#)

Detalle del contacto

asdf

666666666

particular

m.perez2@edu.gva.es

asdf

ARCHIVO ADJUNTO

← Volver



Productos y servicios

[Inicio](#)

Contacte con nosotros

87665543

xxx@gmail.com



Portfolio

Los mejores proyectos de Django, WordPress y Drupal