

## Trabajo integrador.

El objetivo del presente trabajo es implementar de manera práctica, todos los puntos teóricos vistos en el curso y agregar algunas variaciones de dichos temas.

Al acabar el trabajo, el alumno habrá desarrollado, una aplicación web que contendrá las principales funcionalidades que todo sistema web debería tener, principalmente acceso a administradores, gestión de accesos y perfiles utilizando sesiones y resguardo de datos con base de datos.

Para una mejor comprensión y para facilitar el trabajo del alumno, dividiremos el desarrollo en actividades.

### *Actividad 1. Armado de la plantilla base.*

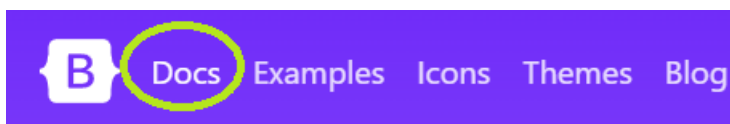
El maquetado del proyecto lo vamos a hacer aplicando los estilos de Bootstrap, por lo que comenzaremos armando nuestra carpeta de trabajo, descargando los archivos de Bootstrap y armando nuestra plantilla base.

Entramos al sitio de Bootstrap y descargamos los archivos de la versión 5. Estos archivos tienen todas las funciones de JavaScript y los archivos de estilo necesarios para implementar este FRAMEWORK por completo en nuestro proyecto.

Si bien solo necesitaremos dos archivos, debido a que son muy livianos y que no nos generan inconvenientes, vamos a armar nuestra carpeta de trabajo base con todo Bootstrap dejando así todo listo para futuros proyectos.

Link: <https://getbootstrap.com/>

Desde la barra de navegación, accedemos a la opción DOCS.



Veremos aparecer un menú lateral de opciones, donde seleccionaremos la opción de descarga:

## Getting started

Introduction

Download

Contents

Luego, estando en la sección de descargas, seleccionaremos la primer opción de descarga que se nos ofrece, y pulsaremos sobre el botón DOWNLOAD.

## Compiled CSS and JS

Download ready-to-use compiled code for **Bootstrap v5.3.3** to easily drop into your project, which includes:

- Compiled and minified CSS bundles (see [CSS files comparison](#))
- Compiled and minified JavaScript plugins (see [JS files comparison](#))

This doesn't include documentation, source files, or any optional JavaScript dependencies like Popper.

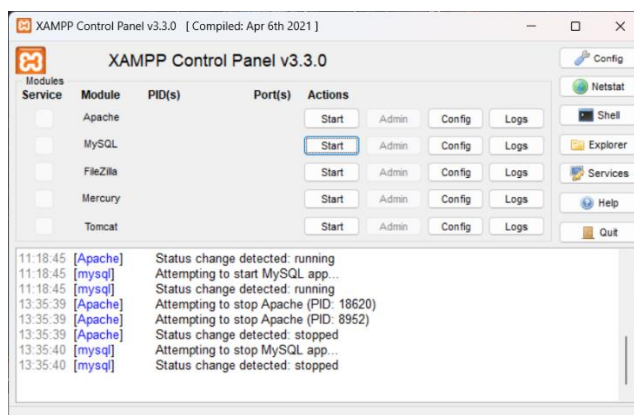
Download

Una vez hecho esto, veremos en la sección de descargas de nuestra PC, un archivo comprimido que es el que tiene todo lo necesario para trabajar con Bootstrap.

Vamos ahora a crear una carpeta de trabajo donde iremos realizando todas las actividades de nuestro proyecto.

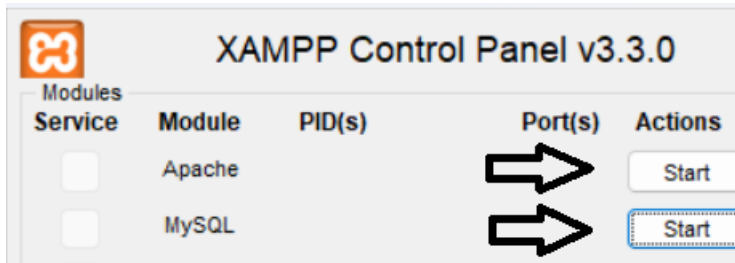
Como todo lo realizaremos con PHP, lo primero es iniciar el servidor de trabajo, para este documento, trabajamos con XAMPP, pero cualquier otra versión comercial de PHP y MySQL sirve para el presente trabajo.

Ejecutamos el panel de control.

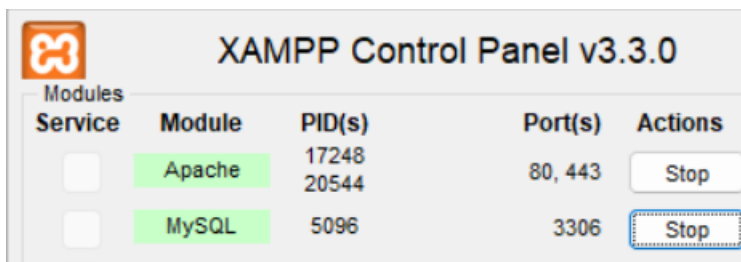


Para esta parte del proyecto, solo vamos a necesitar el servidor Apache, pero no está de mas verificar el funcionamiento del servidor de base de datos, para ir previendo el buen funcionamiento para futuras tareas.

Procedemos a iniciar ambos servidores, pulsando sobre las correspondientes opciones de inicio.



Pulsando en ambos botones START, veremos como el nombre de cada servidor (Apache / MySQL) se remarkan en amarillo y si todo funciona bien, quedan remarcados en color verde.



\*\*\* Error en los puertos.

En algunas ocasiones, nos encontramos con el error de puerto ocupado. Existen muchos tutoriales de como hacer el cambio de puerto, les dejo un par de videos, uno para cada servidor, que pueden ser de utilidad.

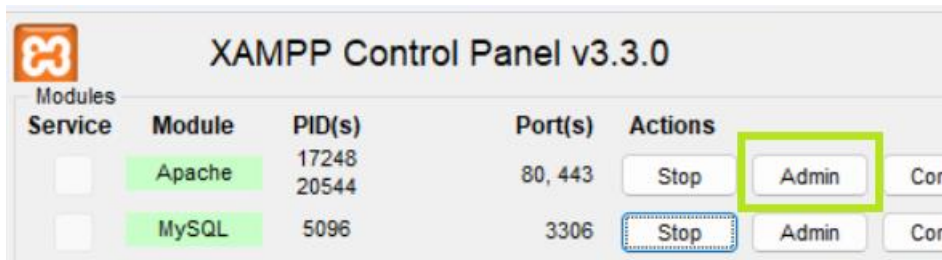
Cambiar puertos Apache.

[https://www.youtube.com/watch?v=MPDchC9qtug&ab\\_channel=JuanMart%C3%ADn](https://www.youtube.com/watch?v=MPDchC9qtug&ab_channel=JuanMart%C3%ADn)

Cambiar puertos MySQL.

[https://www.youtube.com/watch?v=9nl0\\_vEQcGI&ab\\_channel=Byspel-Iv%C3%A1nL](https://www.youtube.com/watch?v=9nl0_vEQcGI&ab_channel=Byspel-Iv%C3%A1nL).

Para verificar si el servidor inició correctamente, además de ver el nombre de este remarcado en color verde, vamos a acceder a la página principal de la aplicación web que viene con la instalación, pulsando sobre el botón ADMIN del servidor Apache.



Si el servidor se inició sin errores, veremos la siguiente pantalla abrirse en nuestro navegador:

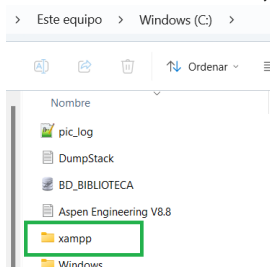


## Welcome to XAMPP for Windows 8.1.10

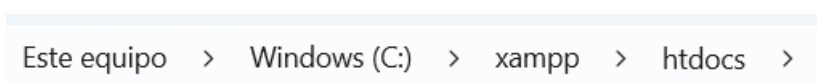
A partir de acá, ya estamos listos para iniciar el armado de nuestra carpeta de trabajo.

Recordemos que siempre que trabajemos con PHP, nuestros proyectos deberán armarse en la carpeta de trabajo del servidor, para el caso de XAMPP, dicha carpeta es HTDOCS.

Si no hemos cambiado la ruta de instalación de XAMPP que viene por defecto en su instalador, lo veremos instalado en la raíz de nuestro disco.



Dentro de la carpeta principal de la instalación, vamos a encontrar nuestra carpeta de trabajo, HTDOCS.



Dentro de esta carpeta vamos a colocar todos nuestros proyectos. Es importante que cada proyecto que vayamos a desarrollar, lo coloquemos dentro de su propia y exclusiva carpeta, para que podamos exportarlo o compartirlo de manera sencilla y sin complicaciones.


En mi caso, voy a crear dentro de HTDOCS, una carpeta llamada PROYECTO donde iré colocando todos los archivos necesarios para el trabajo integrador.


Lo ideal, dentro de las buenas prácticas de programación web, es crear nuestras carpetas y archivos de trabajo con nombres sencillos, en minúscula y sin utilizar caracteres especiales ni espacios en blanco. Consideremos que estos nombres de carpetas y archivos van a terminar formando parte de la URL y debemos mantenerla siempre lo mas simple posible para evitar errores no solo durante el desarrollo del trabajo, sino también para facilitar el uso de nuestro trabajo a los futuros usuarios.

```
Este equipo > Windows (C:) > xampp > htdocs > proyecto
```

Con nuestra carpeta debidamente creada, vamos a comenzar nuestro trabajo en ella, colocando el archivo comprimido que descargamos desde la página de Bootstrap y lo descomprimimos.

Nombre

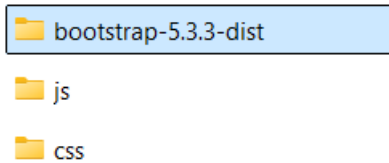
 bootstrap-5.3.3-dist

 bootstrap-5.3.3-dist

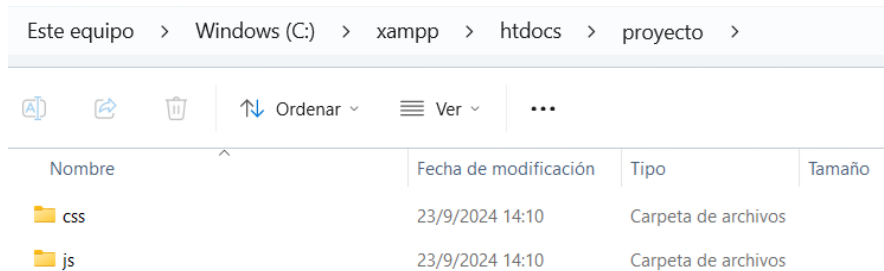
Ya podemos eliminar el archivo comprimido porque no lo vamos a necesitar más.

Vemos que ahora tenemos una carpeta con el mismo nombre del archivo comprimido que descargamos. Dentro de esta carpeta vamos a encontrar otras dos carpetas, CSS y JS que contienen todos los archivos de Bootstrap necesarios para nuestro proyecto.

Vamos a copiar dichas carpetas y las vamos a pegar fuera de la carpeta en la que están, y dentro de nuestra carpeta de trabajo, de esta manera:



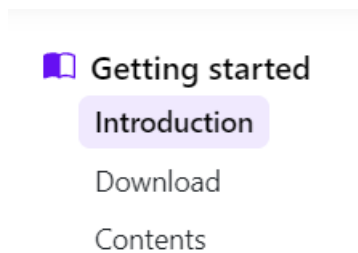
Ahora ya podemos eliminar la carpeta de Bootstrap y quedarnos solo con las carpetas CSS y JS dentro de nuestra carpeta de trabajo.



Ya tenemos nuestros archivos con los estilos y las funciones de JavaScript necesarias para poder implementar Bootstrap.

Comenzaremos con el armado del maquetado base de nuestro proyecto, para lo cual deberemos nuevamente ingresar a la página de Bootstrap.

En el menú de opciones lateral, iremos ahora a la opción INTRODUCTION



Esto nos llevará a la sección de inicio, donde tendremos diferentes opciones para comenzar a trabajar con Bootstrap, dentro de dichas opciones, vamos a utilizar la segunda, que nos ofrece un ejemplo de maquetado web ya armado para el uso de los archivos de nuestro FRAMEWORK.

2. **Include Bootstrap's CSS and JS.** Place the `<link>` tag in the `<head>` for our CSS, and the `<script>` tag for our JavaScript bundle (including Popper for positioning dropdowns, poppers, and tooltips) before the closing `</body>`. Learn more about our [CDN links](#).

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" re
  </head>
  <body>
    <h1>Hello, world!</h1>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.
  </body>
</html>
```

Este maquetado HTML es el que vamos a utilizar para iniciar nuestra plantilla base, para lo cual, crearemos con nuestro editor favorito (en mi caso VS CODE) un archivo llamado PLANTILLA.PHP.

Dentro de este archivo, copiaremos el maquetado sugerido por Bootstrap en la opción 2 (el de la figura anterior) y lo pegaremos en este archivo.

2. **Include Bootstrap's CSS and JS.** Place the `<link>` tag in the `<head>` for our CSS, and the `<script>` tag for our JavaScript bundle (including Popper for positioning dropdowns, poppers, and tooltips) before the closing `</body>`. Learn more about our [CDN links](#).

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width. initial-scale=1">
```

Copy to clipboard

Pegamos el código copiado, en nuestro archivo PLANTILLA.PHP

```
plantilla.php •
C: > xampp > htdocs > proyecto > plantilla.php > ...
1  <!doctype html>
2  <html lang="en">
3    <head>
4      <meta charset="utf-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1">
6      <title>Bootstrap demo</title>
7      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" i
8    </head>
9    <body>
10     <h1>Hello, world!</h1>
11     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="s
12   </body>
13 </html>
14
```

Guardamos los cambios y ya tenemos nuestro maquetado base para empezar a programar.

Vamos a modificar algunas de las líneas de nuestra plantilla base, por ejemplo, el valor del atributo LANG, el TITLE y las rutas de acceso a los archivos CSS y JS ya que vamos a cambiar la búsqueda en internet de dichos archivos para utilizarlos localmente, desde las descargas que hicimos.

Así deberían quedar dichas modificaciones realizadas en las líneas 2,6,7 y 11.

```
plantilla.php X
C: > xampp > htdocs > proyecto > plantilla.php > ...
1  <!doctype html>
2  <html lang="es">
3    <head>
4      <meta charset="utf-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1">
6      <title>Plantilla Base</title>
7      <link href="css/bootstrap.min.css" rel="stylesheet">
8    </head>
9    <body>
10     <h1>Hello, world!</h1>
11     <script src="js/bootstrap.bundle.min.js"></script>
12   </body>
13 </html>
```

Vamos a eliminar la línea 10 ya que agregaremos algún título mejor elaborado desde Bootstrap.



La distribución de los elementos no tiene que ser exactamente como se ve en este instructivo, si bien los elementos a agregar deberían ser todos los mismos, cada alumno los puede organizar como mejor le parezca.

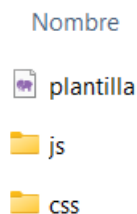
Comencemos con una barra de navegación que contenga en principio las principales actividades que vamos a realizar y con el paso de las clases, iremos agregando las demás.

A modo de sugerencia, vamos a colocar una barra de navegación que se quede fija en la parte superior para facilitar a los usuarios que siempre este disponible cuando necesite usarla.

En cuanto al BODY, voy a optar por un contenedor que aplique un margen importante a izquierda y a derecha de la pantalla ya que no vamos a comenzar agregando gráficos y datos, sino que comenzaremos con las principales funcionalidades.

Es importante pensar nuestro desarrollo como escalable, es decir, a todo lo que vayamos desarrollando, le iremos agregando con el tiempo algunas modificaciones generalmente relacionadas con estilos visuales.

Con estas modificaciones en nuestro maquetado base, nuestra aplicación debería estar así:



Ya tenemos en este punto, armada la estructura base de nuestro proyecto, vamos a concentrarnos ahora en agregar a nuestra plantilla, los primeros elementos que serán los que heredarán los archivos que hagamos a partir de la mencionada plantilla.

## AGREGAMOS UNA BARRA DE NAVEGACIÓN.

Vimos que lo interesante de usar Bootstrap es que podemos tomar de manera muy sencilla cualquier elemento HTML y utilizarlo en nuestro proyecto.

Vamos a dar un pasito mas y vamos a crear cada uno de estos con un paso a paso que nos permita entre otras cosas, ir entendiendo e incorporando el por que de cada clase CSS que utilicemos.

La barra de navegación la vamos a colocar en la parte superior de nuestro maquetado, y vamos a configurarla para que quede fija, algo que los alumnos podrán modificar muy fácilmente si lo que quieren es que la barra no esté fija en una determinada posición.

Como la barra de navegación es algo que los usuarios verán, vamos a colocarla dentro del BODY.

Comenzamos agregando la etiqueta NAV que es la que corresponde usar para este elemento.

Para mayor comprensión del código a utilizar, lo voy a desarrollar en un archivo aparte y luego lo voy a colocar en la plantilla que estamos armando.

A la etiqueta NAV le vamos a colocar las siguientes clases CSS:

```
<nav class="navbar navbar-expand-lg bg-body-tertiary">  
  
</nav>
```

¿Qué hace cada una de estas clases?

.navbar: Esta clase se utiliza para crear una barra de navegación. Es el contenedor principal que define el estilo y el comportamiento de la barra de navegación. Incluye soporte para el branding, la navegación y otros elementos como formularios y botones.

.navbar-expand-lg: Esta clase se usa para hacer que la barra de navegación sea responsive. La parte expand-lg indica que la barra de navegación se expandirá (mostrará todos sus elementos) cuando la pantalla sea de tamaño grande (lg) o mayor. En pantallas más pequeñas, los elementos de la barra de navegación se colapsarán en un menú desplegable.

.bg-body-tertiary: Esta clase aplica un color de fondo específico a la barra de navegación. En Bootstrap 5, las clases de fondo como bg-body-tertiary utilizan variables CSS para definir colores que se integran con el esquema de color general del sitio



El primer DIV que agregamos (líneas 2 y 11, es el que va a guardar todo el contenido de la barra, y le vamos a agregar la clase que hace uso de todo el ancho disponible CONTAINER FLUID).

Recordar: al BODY le aplicamos la clase CONTAINER, lo que aplica margen a izquierda y derecha de nuestro maquetado, y a la barra de navegación le aplicamos la clase CONTAINER-FLUID, para que tome todo el ancho de que dispone el BODY.

Vamos a iniciar agregando de a uno los enlaces de nuestro segundo DIV, el que habíamos colocado inmediatamente debajo del botón.  
Primero le agregamos las clases de estilo que vamos a utilizar:

```
<!-- Segundo DIV -->  
<div class="collapse navbar-collapse" id="navbarSupportedContent">  
  
</div>
```

Todos los enlaces los vamos a armar usando listas, que como ya vimos, son etiquetas LI dentro de etiquetas UL.  
Comencemos con algunos enlaces:

```
<!-- Segundo DIV -->  
<div class="collapse navbar-collapse" id="navbarSupportedContent">  
  <ul class="navbar-nav me-auto mb-2 mb-lg-0">  
    <li class="nav-item">  
      <a class="nav-link active" aria-current="page" href="#">Inicio</a>  
    </li>  
  </ul>  
</div>
```

Vamos a agregar un par de enlaces más, para usar en las primeras prácticas de nuestro proyecto.

```
<!-- Segundo DIV -->
<div class="collapse navbar-collapse" id="navbarSupportedContent">
  <ul class="navbar-nav me-auto mb-2 mb-lg-0">
    <li class="nav-item">
      <a class="nav-link active" aria-current="page" href="#">Inicio</a>
    </li>
    <li class="nav-item">
      <a class="nav-link active" href="#">Formularios</a>
    </li>
    <li class="nav-item">
      <a class="nav-link active" href="#">ABM</a>
    </li>
  </ul>
</div>
```

Vamos a agregar un par de clases para darle estilo DARK a nuestro menú. La etiqueta NAV nos debería quedar así:

```
<nav class="navbar navbar-expand-lg bg-body-tertiary navbar-dark bg-dark sticky-top">
```

Vamos a agregar ahora, a nuestra barra de navegación, un enlace que permita acceder a sub opciones, es decir, una opción que contenga mas de una sub opción.

Esto lo hacemos también con elementos UL/LI y las correspondientes clases de Bootstrap.

Agregamos lo siguiente debajo del último enlace agregado, el de la página ABM.

```
<!-- Opción con Sub opciones -->
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle active" href="#" role="button" data-bs-toggle="dropdown" aria-expanded="false">
    Opciones
  </a>
  <ul class="dropdown-menu">
    <li><a class="dropdown-item" href="#">Sub-opción A</a></li>
    <li><a class="dropdown-item" href="#">Sub-opción B</a></li>
    <li><hr class="dropdown-divider"></li>
    <li><a class="dropdown-item" href="#">Sub-opción C</a></li>
  </ul>
</li>
```

Para finalizar, vamos a maquetar un pequeño formulario dentro de la barra de navegación, a la que mas adelante le programaremos la función de búsqueda de contenido dentro de nuestra página.

Debajo de la última etiqueta de cierre /UL que colocamos recién, vamos a agregar el siguiente formulario:

```
<form class="d-flex" role="search">
  <input class="form-control me-2" type="search" placeholder="Buscar contenido" aria-label="Search">
  <button class="btn btn-outline-success" type="submit">Buscar</button>
</form>
```

COLOCAMOS LA BARRA DE NAVEGACIÓN EN EL BODY.

Apenas comienza nuestro BODY, vamos a colocar el código que acabamos de escribir, en mi caso lo arme en un archivo aparte, pero si lo hicieron directamente dentro de la plantilla, está correcto.

```
<body class="container">
  <!-- Barra de navegación -->
  <nav class="navbar navbar-expand-lg bg-body-tertiary navbar">
    <!-- Primer DIV -->
    <div class="container-fluid">
      <a class="navbar-brand" href="http://www.guille">
      <button class="navbar-toggler" type="button" da
        aria-controls="navbarSupportedContent" aria
        <span class="navbar-toggler-icon"></span>
      </button>
      <!-- Segundo DIV -->
      <div class="collapse navbar-collapse" id="navba
        <ul class="navbar-nav me-auto mb-2 mb-lg-0"
          <li class="nav-item">
            <a class="nav-link active" aria-cur
```

Vamos a guardar todos los cambios y ver nuestro proyecto hasta el momento.

Para esto, desde el navegador de internet, vamos a editar la URL.

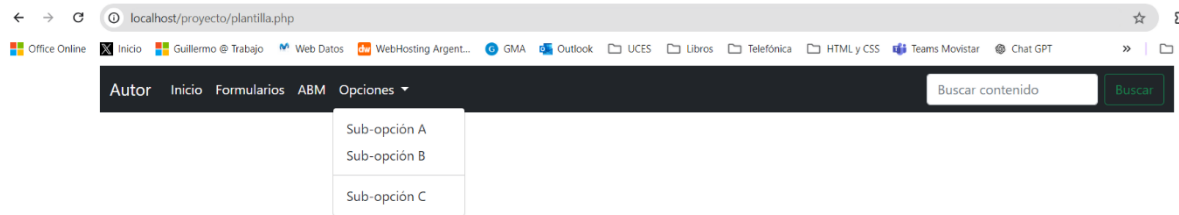
Así se ve ahora:

<http://localhost/dashboard/>

Recordamos que nuestro localhost es la contraparte de nuestra carpeta HTDOCS, por lo que a partir del localhost, vamos a armar la ruta para que el navegador de internet encuentre el archivo que queremos visualizar, para el ejemplo de este apunte, deberíamos escribir lo siguiente:

<http://localhost/proyecto/plantilla.php>

Y esto es lo que deberíamos ver, en mi caso voy a desplegar el menú con sub opciones para verificar que funciona:




Si llegaron hasta acá con el mismo resultado, podemos dar por terminada la actividad número uno.

## *Actividad 2. Barra de navegación en una función PHP.*

Nuestra plantilla base va a servir de punto de partida para la creación de muchos de los archivos que vamos a agregar a nuestro proyecto, por lo que nos sería de utilidad que contenga la mayor cantidad de elementos comunes de nuestro desarrollo, para evitarnos la complicación de tener código reutilizable en más de un archivo.

Habiendo visto como incluir archivos y como declarar funciones, veremos como aprovechar este conocimiento, para crear un archivo externo que contenga a nuestra barra de navegación.

Como primer paso, vamos a crear en nuestro desarrollo, una carpeta llamada INC donde iremos guardando los archivos a incluir, que sean de extensión PHP.

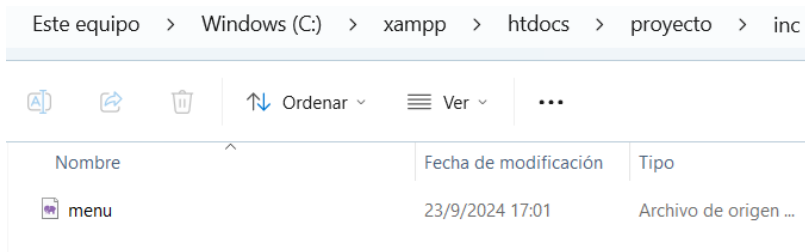
 plantilla

 js

 inc

 css

Dentro de la carpeta INC, vamos a crear un archivo, con extensión PHP, en el que vamos a crear una función llamada MENU.



Dentro del archivo, que va a contener solo código PHP, comenzaremos colocando las etiquetas obligatorias para dicho lenguaje y vamos a declarar la función.

```
<?php
// Declaramos la función
function menu(){

}
?>
```

Dentro de esta función PHP, vamos a colocar el maquetado de la barra de navegación que hicimos en la actividad uno, pero para que el interprete de PHP no se confunda, vamos a partir en dos partes nuestro código, y entre estas dos partes, pegaremos el maquetado, así, de esta manera, cuando el interprete de PHP lea nuestro código, no va a encontrar ningún error de sintaxis.

Vamos con la primer parte, partimos la función en dos partes.

```
<?php
// Declaramos la función
function menu(){
?>

<!--
Este espacio, al no estar entre etiquetas
PHP sin cerrar, no se considera de dicho
lenguaje, lo que nos permite colocar HTML
-->

<?php
}
?>
```



Ahora vamos a ir a nuestra página llamada plantilla y vamos a cortar el maquetado de la barra de navegación, y lo vamos a pegar dentro del archivo que acabamos de crear, el que contiene la función menú, y lo vamos a pegar justo entre las dos partes en que dividimos nuestra función.

Si actualizamos nuestro navegador, veremos que la barra de navegación desapareció.

Vamos a agregar la línea de código que incluye nuestro archivo externo y luego la que invoca la función, para que tengamos nuestra plantilla igual que antes, pero mucho mas prolija y reutilizando el código del menú.

Así debería quedar nuestro código del archivo plantilla.

```
<!doctype html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Plantilla Base</title>
    <link href="css/bootstrap.min.css" rel="stylesheet">
    <?php
      |   require("inc/menu.php")
    ?>
  </head>
  <body class="container">
    <!-- Barra de navegación -->
    <?php menu(); ?>
    <script src="js/bootstrap.bundle.min.js"></script>
  </body>
</html>
```

Agregué el siguiente código:

```
<?php
|   require("inc/menu.php")
?>
```

Y este:

```
<!-- Barra de navegación -->
<?php menu(); ?>
```

Nuevamente, deberíamos poder ver desde nuestro navegador de internet, la siguiente pantalla:

## Actividad 3: Agregamos un título permanente.

Para que el usuario pueda identificar en que sector de nuestro sistema web se encuentra, además de la etiqueta TITLE que agregaba una breve descripción a la pestaña del navegador, vamos a colocar un pequeño contenedor al que le agregaremos un título genérico y lo modificaremos en cada archivo que vayamos a crear a partir de nuestra plantilla.

El elemento que vamos a agregar es un ALERT de Bootstrap, y lo haremos inmediatamente después del menú de navegación.

Así debería quedar nuestro BODY:

```
<body class="container">
  <!-- Barra de navegación -->
  <?php menu(); ?>
  <!-- Título de la página -->
  <div class="alert alert-primary text-center fst-italic" role="alert">
    <h4>Título de la página.</h4>
  </div>
  <script src="js/bootstrap.bundle.min.js"></script>
</body>
```

Y esto deberíamos ver por pantalla:

*Título de la página.*

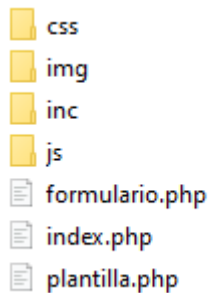
Si llegamos a este resultado, damos por finalizada la actividad tres.

## Actividad 4: Página de formulario.

En esta actividad vamos a aprender a enviar datos desde la casa de nuestros usuarios hasta el servidor donde está alojada nuestra página web. Comenzaremos creando el archivo de la práctica a partir de nuestra plantilla.

Vamos a llamarlo “Formulario” y lo guardamos junto a las páginas de nuestra web.

Le modificamos la etiqueta TITLE y el título de la página, podemos colocarle el texto: “Envíos de datos al servidor.”



A continuación, vamos a editar el archivo que contiene el menú de navegación y a agregar el enlace para este archivo que acabamos de crear.

```
<li class="nav-item">
  <a class="nav-link active" aria-current="page" href="index.php">Inicio</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="formulario.php">Formulario</a>
</li>
```

Guardamos todos los cambios y actualizamos para ver si ya tenemos visible nuestra nueva página.

Procedemos luego a maquetar una grilla a partir de una fila que dividiremos en tres secciones.

```
<!-- Fila 1 -->
<div class="row">
  <div class="col-3"></div>
  <div class="col-6"></div>
  <div class="col-3"></div>
</div>
```

Y en la división del centro, colocamos un pequeño formulario en principio para enviar dos datos hasta el servidor, un nombre y un apellido. Colocamos también un botón para el envío del formulario.

```
<form>
  <div class="mb-3">
    <label for="nombre" class="form-label">Nombre</label>
    <input type="text" class="form-control" id="nombre" name="nombre">
  </div>
  <div class="mb-3">
    <label for="apellido" class="form-label">Apellido</label>
    <input type="text" class="form-control" id="apellido" name="apellido">
  </div>
  <button type="submit" class="btn btn-primary">Enviar</button>
</form>
```

Agregamos la página de destino y el método de envío.

Luego creamos dicha página, que podría llamarse “formulario\_destino”

Los valores del atributo NAME son muy importantes porque es el índice con el que el servidor va a almacenar los datos que enviamos.

A continuación, abrimos esta nueva página de destino donde comenzaremos a escribir el código que nos permitirá ver el envío de datos desde el cliente, y para hacer las pruebas con diferentes elementos, veremos como se reciben desde el servidor, las asignaremos a nuevas variables y las mostraremos por pantalla.

Colocamos el siguiente código en la nueva página:

```
<?php
// Recibo los datos.
$p_nombre = $_POST['nombre'];
$p_apellido = $_POST['apellido'];

echo 'Nombre recibido: '.$p_nombre.'<br>';
echo 'Apellido recibido: '.$p_apellido;

?>
```

Lo que hicimos fue crear un par de variables y asignarles el contenido del arreglo POST, que es el arreglo donde se guardan los datos que enviamos por dicho método, y recordamos que se guardan en dicho arreglo usando como índice el valor que colocamos dentro del formulario usando el atributo NAME en cada etiqueta.

Luego, los mostramos por pantalla para verificar que los datos fueron enviados.

El nombre ingresado es: Guillermo

El apellido ingresado es: Alfaro

### Campo PASSWORD.

El campo PASSWORD es un campo de texto con la particularidad que no se muestran los caracteres ingresados.  
Agreguemos uno a nuestro formulario.

```
<div class="form-group">  
  <label for="clave">Clave del usuario.</label>  
  <input type="password" id="clave" name="clave" placeholder="Ingresa su c  
lave" class="form-control">  
</div>
```

Así quedaría nuestro formulario:

Nombre.

Apellido.

Password.

Así como agregamos un elemento en nuestro formulario, tenemos que agregarlo en la página destino si queremos utilizarlo.

```
// Recibo los datos.  
$p_nombre = $_POST['nombre'];  
$p_apellido = $_POST['apellido'];  
$p_clave = $_POST['clave'];  
  
echo 'Nombre recibido: '.$p_nombre.'<br>';  
echo 'Apellido recibido: '.$p_apellido.'<br>';  
echo 'Clave ingresada: '.$p_clave;
```

Nombre recibido: Guillermo  
Apellido recibido: Alfaro  
Clave ingresada: aplicacionesweb

### CHECKBOX.

Este elemento permite al usuario seleccionar varias opciones a la vez.  
La siguiente es la sintaxis básica:

```
<hr size="2px" color="black" />  
  
<h6 class="font-italic">Seleccione sus materias favoritas.</h6>  
<div class="form-group">  
  <input type="checkbox" id="materia1" name="materia1" value="php">  
  <label for="materia1">Aplicaciones Web</label>  
</div>  
<div class="form-group">  
  <input type="checkbox" id="materia2" name="materia2" value="java">  
  <label for="materia2">JAVA</label>  
</div>  
<div class="form-group">  
  <input type="checkbox" id="materia3" name="materia2" value="asp_net">  
  <label for="materia3">ASP Net</label>  
</div>
```

De la misma forma que hicimos con los cuadros de texto, para este caso también deberemos recibirlos en la página destino mediante POST, si miramos el código en detalle, veremos que cada una de las opciones del CHECKBOX tiene un valor distinto en el atributo NAME, por lo que

deberemos tratarla a cada una de dichas opciones, como si fuese un elemento independiente de los demás.

```
if(isset($_POST['materia1'])){  
    $p_materia1 = $_POST['materia1'];  
}else $p_materia1 = '';  
  
if(isset($_POST['materia2'])){  
    $p_materia2 = $_POST['materia2'];  
}else $p_materia2 = '';  
  
if(isset($_POST['materia3'])){  
    $p_materia3 = $_POST['materia3'];  
}else $p_materia3 = '';
```

Es importante verificar que el POST exista, caso contrario, si el usuario no selecciona alguna de las opciones, el intérprete PHP nos dará error.

Lo usual es verificar la existencia mediante la función ISSET y en caso de no existir, asignarle un valor por defecto, en nuestro caso, una cadena vacía.

**Nombre recibido: Guillermo**

**Apellido recibido: Alfaro**

**Clave ingresada:**

**Tus materias favoritas son: java asp\_net**

## RADIO BUTTON.

Este elemento permite al usuario elegir una, y solo una, opción de varias. Veamos el código básico para agregar dicho elemento a nuestro formulario: Si vemos el siguiente código, notaremos que todos los elementos tienen el mismo valor en el atributo NAME, es por ese motivo que, al seleccionar, solo podremos elegir una sola de todas las opciones.

```
<fieldset>
<legend>Seleccione su nivel de inglés</legend>
  <div class="form-group">
    <label>
      <input type="radio" name="nivel" value="alta">Alto
    </label>
  </div>
  <div class="form-group">
    <label>
      <input type="radio" name="nivel" value="medio">Medio
    </label>
  </div>
  <div class="form-group">
    <label>
      <input type="radio" name="nivel" value="bajo">Bajo
    </label>
  </div>
</fieldset>
```

Veamos como recibir los valores del elemento RADIO en el archivo de destino.

De la misma manera que hicimos con el CHECKBOX, acá también debemos verificar que el dato realmente fue enviado para evitar que se produzca un error.

```
if(isset($_POST['nivel'])){
    $p_nivel = $_POST['nivel'];
}else $p_nivel = 'El nivel de ingles no fue enviado';
```

## Desplegable (SELECT).

El elemento SELECT posibilita al usuario elegir una opción de una lista desplegable. Veamos el siguiente ejemplo básico:

```
<div class="form-group">
  <label for="selector1">Seleccione el motivo de su contacto.</label>
  <select name="selector1" id="selector1">
    <option value="consulta">Consulta</option>
    <option value="sugerencia">Sugerencia</option>
    <option value="queja">Queja</option>
  </select>
```

Y de esta forma recibimos el valor seleccionado en la página de destino:



```
$selector = $_POST['selector1'];
```

### *Actividad 5. ABM básico, parte 1.*

Ya vimos el funcionamiento de los formularios con PHP, hicimos el envío y recepción de información entre diferentes páginas y verificamos la forma de recibir la información del lado del servidor y almacenarla en variables locales para disponer de ellas como sea necesario.

Vamos ahora a aprovechar el uso de formularios y tablas para trabajar con datos de una base de datos.

#### **CREAMOS UNA BASE DE DATOS DE PRUEBA.**

El primer paso para trabajar con datos es crear la base que vamos a usar. Podemos hacerlo desde código como hicimos en la clase de MySQL o utilizando el PHPMyAdmin.

**PASO 1)** Ingresamos al gestor de bases de datos y creamos la base TP1, y para el set de caracteres usaremos utf8\_general\_ci.



**PASO 2)** Creamos la tabla usuario:

```
create table usuario (  
    usuario varchar(255) primary key,  
    clave varchar(255) not null,  
    rol varchar (255) not null  
)
```

**PASO 3)** Cargamos algunos datos de prueba.  
¿Qué pasaría si intentáramos cargar lo siguiente?

```
insert into usuario values ('admin','1234','administrador');  
insert into usuario values ('admin','1234','administrador');
```

\*\*\* El campo usuario está definido como clave primaria, por lo que la base de datos no admitiría valores duplicados para dicho campo.

Insertamos algunos datos de prueba.

```
insert into usuario values ('admin','1234','administrador');  
insert into usuario values ('gma','vamoschaca','administrador');  
insert into usuario values ('perezj','apweb','analista');
```

**PASO 4)** Por último, cargamos un set de datos que tenga alguno de sus valores con un acento para verificar si escogimos bien el juego de caracteres de la base.

```
insert into usuario values ('rodríguez','cañuelas','analista');
```

### **CONEXIÓN A LA BASE DE DATOS.**

Vamos a utilizar nuestra plantilla de práctica para ir guardando los ejemplos prácticos que hagamos en clase y los ejercicios propuestos.

Abrimos el archivo plantilla y lo guardamos con el nombre abm, cambiándole título, subtítulo y agregando el nombre en el menú de navegación.

Este nuevo archivo, como ya sabemos, lo creamos a partir de la plantilla web que diseñamos para toda nuestra aplicación.

Esta práctica la vamos a hacer de forma tradicional ya que aún no hemos visto programación orientada a objetos.

Para conectarnos a la base de datos seguiremos los siguientes pasos:

- 1- Definir los datos de usuario, clave, servidor y base de datos.

```
// PASO 1) Datos de la conexión
$usuario = 'root';
$clave = '';
$servidor = 'localhost';
$basededatos = 'tp1';
```

- 2- Crear la conexión utilizando los datos definidos en el punto 1.

```
// PASO 2) Creamos la conexión
$conexion = mysqli_connect($servidor,$usuario,$clave);
```

- 3- Conectar a la base de datos.

```
// PASO 3) Me conecto a la base de datos
$db = mysqli_select_db($conexion,$basededatos);
```

- 4- Definir la consulta o consultas SQL y guardarla en una variable.

```
// PASO 4) Consultas a SQL de un valor.
$consulta1 = "select count(distinct usuario) as usuarios from usuario";
```

- 5- Ejecutar la consulta o consultas.

```
// PASO 5) Obtenemos y guardamos el resultado
$resultado1 = mysqli_query($conexion,$consulta1);

// Obtenemos el valor
while($fila = mysqli_fetch_assoc($resultado1)){
    $cantidad_usuarios = $fila['usuarios'];
}
```

Si guardamos el archivo y actualizamos la página, veremos que no aparece ningún mensaje porque hasta acá todo está correcto.

\*\*\* Probemos de incluir algún error para ver mejor el funcionamiento, por ejemplo, modifiquemos el valor de la variable \$CLAVE.

```
// Datos
$usuario = 'root';
$clave = '1234';
$servidor = "localhost";
$basededatos = "tp1";
```

Si guardamos y actualizamos, veremos un mensaje de error del sistema mas el nuestro, No se ha podido conectar al servidor.

Lo mismo sucedería para el caso de tener algún dato erróneo para usuario, servidor o base de datos (probar).

Incluso hemos agregado un mensaje de error en caso de haber algún inconveniente con la consulta, por ejemplo, si quisiéramos consultar una tabla inexistente

```
// Creamos la consulta SQL.
$consulta = 'select * from usuariosssss';
```

O si la consulta tuviera algún error de sintaxis:

```
// Creamos la consulta SQL.  
$consulta = 'select * from usuario';
```

Como podemos ver, en el caso de los errores de SQL, no tenemos demasiada información con los mensajes de error ya que en general solo veremos el mensaje por defecto que pongamos.

Es por esto que se aconseja probar todas las consultas SQL en el gestor de bases de datos (PHPMyAdmin) antes de ponerlas en nuestro código PHP.

### **CONSULTAR UN VALOR.**

Veamos el ejemplo más simple, queremos hacer una consulta que nos devuelva un solo valor, por ejemplo, la cantidad de usuarios cargados en nuestra tabla.

```
select count(distinct usuario) as usuarios  
from usuario
```

Para esto hacemos uso de MYSQLI\_FETCH\_ASSOC para recuperar una sola fila, en nuestro caso, es una fila con solo un campo, y lo guardamos en una variable.

```
// Obtenemos y guardamos el resultado.  
while ($fila = mysqli_fetch_assoc($resultado)) {  
    $cantidad_usuarios = $fila['usuarios'];  
}
```

Y de esta forma lo mostramos en nuestra página, una vez guardado el valor en una variable, lo tendremos disponible en cualquier parte de nuestro código.

```
<!-- Primer fila -->
<div class='container'>
  <div class='row'>
    <button type="button" class="btn btn-primary">
      Cantidad de usuarios <span class="badge badge-light">
        <?php echo $cantidad_de_usuarios; ?>
      </span>
    </button>
  </div>
</div>
```

Cantidad de usuarios **4**

### **CONSULTAR UN VALOR USANDO UNA VARIABLE.**

La forma más sencilla de trabajar con consultas SQL y variables PHP, es encerrando la consulta entre comillas dobles y dentro de la misma, insertar la variable encerrada en comillas simples como en el ejemplo que sigue:

```
// Creamos la consulta SQL usando una variable.
$rol = 'administrador';
$consulta = "select count(distinct usuario) as usuarios
from usuario where rol = '$rol' ";
```

Lo que hicimos fue crear una variable llamada \$rol a la que le cargamos el valor administrador y luego la incluimos dentro de la consulta SQL para que nos cuente la cantidad de usuarios cargados, pero solo aquellos cuyo campo rol tenga el valor administrador.

### **EJECUTAR MAS DE UNA CONSULTA.**

Podemos definir y ejecutar tantas consultas como sea necesario, veamos, por ejemplo, consultar tres valores, la cantidad total de usuarios, la cantidad de administradores y la cantidad de usuarios con una clave de 5 dígitos que además sean analistas.

El primer paso es definir las variables que usaremos para consultar un tipo específico de rol:

```
$rol1 = 'administrador';  
$rol2 = 'analista';
```

Segundo paso, escribimos las tres consultas que queremos ejecutar:

```
$consulta1 = "select count(distinct usuario) as usuarios  
from usuario";  
  
$consulta2 = "select count(distinct usuario) as usuarios  
from usuario where rol = '$rol1' ";  
  
$consulta3 = "select count(distinct usuario) as usuarios  
from usuario where rol = '$rol2' and length(clave)=5 ";
```

Como tercer paso, ejecutamos las consultas:

```
// Ejecutamos las consultas SQL.  
$resultado1 = mysqli_query( $conexion, $consulta1 )  
or die ('No se ha podido ejecutar la consulta 1.');
```

```
$resultado2 = mysqli_query( $conexion, $consulta2)  
or die ('No se ha podido ejecutar la consulta 2.');
```

```
$resultado3 = mysqli_query( $conexion, $consulta3 )  
or die ('No se ha podido ejecutar la consulta 3.');
```

Por último, recorremos el registro obtenido buscando el campo que necesitamos y lo asignamos a una variable para luego poder utilizarla en nuestra aplicación.

```
// Obtenemos y guardamos los resultados.
while ($fila = mysqli_fetch_assoc($resultado1)) {
    $usuarios_total = $fila['usuarios'];

    while ($fila = mysqli_fetch_assoc($resultado2)) {
        $usuarios_administrador = $fila['usuarios'];

        while ($fila = mysqli_fetch_assoc($resultado3)) {
            $usuarios_analista = $fila['usuarios']; }
    }
```

Mostramos por pantalla.

```
<div class="row">
    <?php echo 'Usuarios cargados: '.$usuarios_total; ?><br>
    <?php echo 'Usuarios administrador: '.$usuarios_administrador; ?><br>
    <?php echo 'Usuarios analista: '.$usuarios_analista; ?><br>
</div>
```

Esto

deberíamos ver por pantalla:

```
Usuarios cargados en la base: 4
Usuarios con rol de administrador: 2
Usuarios cargados en la base: 1
```

### *Actividad 6. ABM básico, parte 2.*

Vamos a ver ahora como consultar todo el contenido de una tabla y mostrarlo por pantalla.

Agregamos otra división para separar el código nuevo del anterior.

```
<div class="row">

</div>
```

El procedimiento para hacer las consultas es similar, comenzamos por escribir la instrucción SQL que queremos ejecutar.



```
$consulta4 = "select distinct * from usuario";
```

Luego ejecutamos la consulta:

```
$resultado4 = mysqli_query( $conexion, $consulta4 )  
or die ('No se ha podido ejecutar la consulta 4.');
```

Y a partir de acá lo nuevo, ahora no vamos a recorrer la fila y obtener un campo, lo que vamos a hacer es recorrer todo el arreglo de datos que obtuvimos y mostramos por pantalla cada fila con un WHILE hasta que no haya más datos.

Básicamente vamos a armar una tabla con HTML y dentro de la misma, mostraremos el resultado de la consulta.

```
<div class="table-responsive">  
  <table>  
    <thead>  
      <tr>  
        <td>Usuario</td><td>Clave</td><td>Rol</td>  
      </tr>  
    </thead>  
    <tbody>  
    </tbody>  
  </table>  
</div>
```

Una vez maquetada la tabla, procedemos a incrustar instrucciones en PHP dentro del cuerpo de esta donde recorreremos el arreglo de datos que obtuvimos con la consulta e iremos mostrando todas las filas mientras contengan datos.

```
while ($columna = mysqli_fetch_array( $resultado4))
{
    echo "<tr>";
    echo "<td>" . $columna['usuario'] . "</td>"
    echo "<td>" . $columna['clave'] . "</td>"
    echo "<td>" . $columna['rol'] . "</td> </tr>";
}
```

Para el caso de necesitar todo un arreglo en lugar de solo un registro, utilizamos `mysqli_fetch_array`

Si guardamos los cambios y refrescamos el navegador, veremos la tabla con sus datos, por el momento sin ningún tipo de formato.

Aprovechamos que estamos usando Bootstrap y le damos algo de estilo de forma rápida:

```
<table class="table table-bordered table-sm table-hover table-dark">
```

Así debería verse nuestra tabla ahora:

Usuario	Clave	Rol
admin	1234	administrador
gma	vamoschaca	administrador
perezj	apweb	analista
rodríguez	cañuelas	analista

## Columna de acciones sobre la base de datos.

Vamos a agregar una columna a nuestra tabla, donde podremos optar por diferentes acciones sobre los datos cargados.

Lo primero es agregar al encabezado, el nombre de nuestra nueva columna.

```
<td>Usuario</td><td>Clave</td><td>Rol</td><td>Acciones</td>
```

A continuación, iremos agregando una etiqueta `<a>` por cada acción que vayamos a desarrollar. La idea es que LINK agregado nos lleve a una página que ejecute algún tipo de acción.

Por ejemplo, agregamos un ícono para editar los datos de la tabla, para ello debemos agregar otra celda más a cada fila y en dicha celda iremos agregando las acciones:

```
while($columna = mysqli_fetch_array($resultado1)){
    echo '<tr>';
    echo '<td>'.$columna['usuario'].'</td>';
    echo '<td>'.$columna['clave'].'</td>';
    echo '<td>'.$columna['rol'].'</td>';
    echo '<td>
        <a href="#" target="_blank">Editar</a>
        <a href="#" target="_blank">Eliminar</a>
        <a href="#" target="_blank">Detalles</a>
    </td>';
    echo '</tr>';
}
```

En la etiqueta <a> hacemos referencia al archivo que se va a encargar de eliminar el registro de la base de datos.

## *Actividad 7. ABM básico, parte 3. Retoques.*

Continuamos con el desarrollo que comenzamos, para agregar a nuestra aplicación, un ABM básico, desde la web, sobre nuestra base de datos.

Vimos como conectarnos a la base de datos, como hacer consultas y recuperar los datos para ser utilizados por la aplicación, incluso llegamos a modificar las consultas en SQL para poner criterios de selección en base a variables declaradas en PHP.

Habíamos dejado maquetadas las opciones para empezar a construir la parte lógica del ABM con PHP y teníamos pendiente la separación del código de conexión de nuestro archivo, para poder reutilizarlo.

### **ARCHIVO DE CONEXIÓN.**

De todos los puntos en que separamos los pasos de conexión a la BDD, solo los tres primeros son comunes a todas las páginas que requieran conectarse a la base, por lo que procedemos a extraerlo de nuestro archivo BDD.PHP y lo colocamos en un nuevo archivo al que llamaremos CONEXION.PHP y lo dejaremos en la carpeta INCLUDES.

```
// PASO 1) Datos de conexion
$usuario = 'root';
$clave = '';
$servidor = 'localhost';
$basededatos = 'tp1';

// PASO 2) Creo la variable de conexion
$conexion = mysqli_connect($servidor,$usuario,$clave)
or die('No se pudo conectar con el servidor');

// PASO 3) conectamos con la base de datos.
$db = mysqli_select_db($conexion,$basededatos)
or die('No se conecto a la BDD');
mysqli_set_charset($conexion,'utf-8');
```

Una vez que esta parte del código está separada y lista para ser reutilizada, deberemos incluirla en nuestro archivo, para que podamos seguir trabajando con la base de datos.

```
include('includes/conexion.php');
```

## ALGUNOS RETOQUES.

Antes de comenzar a programar, vamos a terminar de definir un par de puntos de estilo que nos quedaron pendientes, por ejemplo, el mostrar todos los cuadros de datos con el mismo tamaño y color.

Podemos usar cualquier elemento de Bootstrap, en este caso a modo de ejemplo, voy a usar botones adaptados para notificaciones.

```
<div class="col-3">
  <button type="button" class="btn btn-primary container-fluid text-start">
    Usuarios: <span class="badge text-bg-primary"><?php echo $cantidad_usuarios; ?></span>
  </button>
</div>
```

Usuarios: 5

Administradores: 2

Analistas: 3

Agregamos también un poco de estilo al botón de nuevo usuario:

```
<div class="col-3">
  <button type="button" class="btn btn-danger container-fluid">
    <a href="#" target="_blank" style="color:white;text-decoration:none">Nuevo usuario</a>
  </button>
</div>
```

Práctica de ABM.

Usuarios: 5

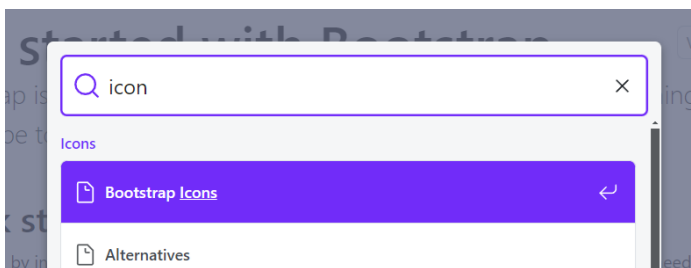
Administradores: 2

Analistas: 3

Nuevo usuario

Por último, en lo que a estilo se refiere, vamos a cambiar el texto de editar y eliminar por íconos, que pueden descargarse de las aplicaciones sugeridas en la documentación de Bootstrap o directamente utilizar alguno de los provistos.

Buscamos íconos en el buscador de Bootstrap:



Seleccionamos la opción de íconos de Bootstrap:

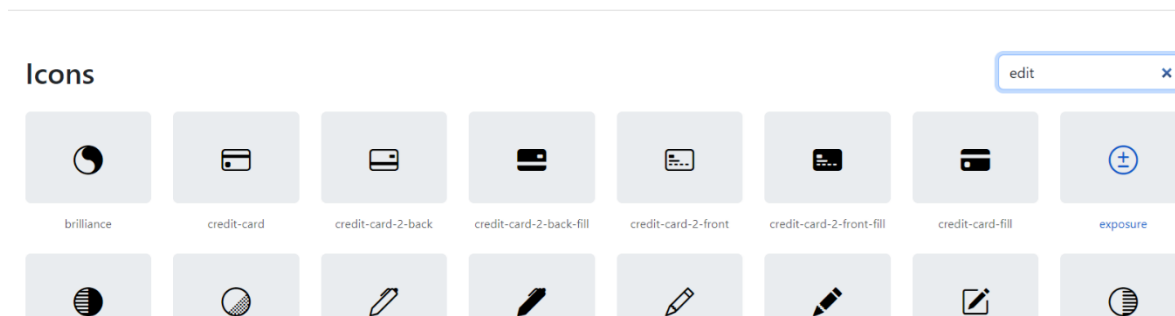
## Bootstrap Icons #

Bootstrap Icons is a growing library of SVG icons that are de  
[Team](#). The beginnings of this icon set come from Bootstrap'  
more. Bootstrap has very few icon needs out of the box, so  
we couldn't stop making more.

Oh, and did we mention they're completely open source? Li  
available to everyone.

[Learn more about Bootstrap Icons, including how to install t](#)

Y accedemos a la página donde podemos buscar íconos para utilizar solo copiando y pegando el código que encontramos, por ejemplo, para el ícono EDIT:



Seleccionamos el que queremos usar y veremos el código a copiar:




Este es el código que copiamos y pegamos para reemplazar el texto EDITAR que dejamos como opción en la tabla cuando la maquetamos.

```
echo "<td>
  <a href='#' style='text-decoration:none'>
    <svg xmlns='http://www.w3.org/2000/svg' width='16' height='16'>
      <path d='M12.854 1.46a 5.5 0 0 0 -.707 0L10.5 1.46'>
    </path>
    </svg>
  </a>
  <a href='#' style='text-decoration:none'>
    <svg xmlns='http://www.w3.org/2000/svg' width='16' height='16'>
      <path d='M11 1.5v1h3.5a 5.5 0 0 1 0 1h-.538l-.538 1.46'>
    </path>
    </svg>
  </a>
</td>";
```

Algunas consideraciones: debemos reemplazar las comillas dobles del código que copiamos por comillas simples y podemos agregar el estilo de “texto sin decoración” para que elimine el subrayado que agrega la etiqueta <a>

Así nos debería quedar la columna de acciones:











Usuario	Clave	Perfil	Acciones
admin	1234	administrador	 
gma	1234	administrador	 
master	12345	analista	 
perezj	tpfundacion	analista	 
rodríguez	cañuelas	analista	 

Antes de pasar a las nuevas funcionalidades, podemos aplicarle algo de color a la tabla.

Simplemente aplicamos alguno de los estilos de color de forma separada al THEAD y al TBODY.

```
<div class="table-responsive">
  <table class="table table-bordered table-sm table-hover">
    <thead class="table-dark text-center">
      <tr>
        <td>Usuario</td><td>Clave</td><td>Perfil</td><td>Acciones</td>
      </tr>
    </thead>
    <tbody class="table-success">
      <?php
```

Y nos queda algo un poco mas agradable:

Usuario	Clave	Perfil	Acciones
admin	1234	administrador	 
gma	1234	administrador	 
master	12345	analista	 
perezj	tpfundacion	analista	 
rodríguez	cañuelas	analista	 

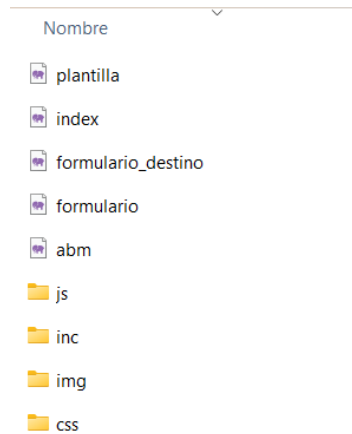
## **ICONO DE PÁGINA.**

Podemos utilizar cualquier imagen que nos resulte adecuada para agregar a todas las pestañas.

Para muestra, me voy a descargar el ícono de HTML.

<https://cdn-icons-png.flaticon.com/512/174/174854.png>

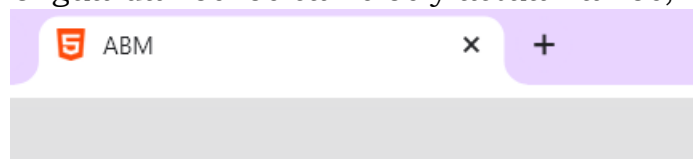
Lo vamos a guardar en una nueva carpeta llamada IMG que la destinaremos a imágenes.



Luego, agregamos la siguiente línea de código en el encabezado de todas nuestras páginas, en especial en la de plantilla, así ya nos queda para los futuros archivos de nuestra aplicación.

```
<?php
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Formulario</title>
  <link href="css/bootstrap.min.css" rel="stylesheet">
  <link rel="icon" href="img/icono_prueba1.png">
</head>
<?php
  require("inc/menu.php");
  require("inc/conexion.php");
?>
```

Si guardamos los cambios y actualizamos, veremos lo siguiente:





## **FONDO DE PANTALLA.**

De manera optativa podemos agregar algún tipo de fondo que ayude a resaltar el contenido de cada pantalla, por ejemplo, de la siguiente manera:

[https://www.google.com/url?sa=i&url=https%3A%2F%2Fdepositphotos.com%2Fes%2Fphoto%2Fwhite-wall-texture-background-for-for-backdrop-composition-for-website-magazine-or-graphic-design-242452432.html&psig=AOvVaw2cW\\_0XHMCnwywuul3ZyMqM&ust=1729018874670000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCKCU4K\\_HjokDFQAAAAAdAAAAABAE](https://www.google.com/url?sa=i&url=https%3A%2F%2Fdepositphotos.com%2Fes%2Fphoto%2Fwhite-wall-texture-background-for-for-backdrop-composition-for-website-magazine-or-graphic-design-242452432.html&psig=AOvVaw2cW_0XHMCnwywuul3ZyMqM&ust=1729018874670000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCKCU4K_HjokDFQAAAAAdAAAAABAE)

El enlace anterior es de un ejemplo de textura que lo podemos aplicar al BODY de la siguiente manera:

```
?>
</head>
<body class="container" background="img/textura1.jpg">
<!-- Barra de navegación -->
<?php menu(); ?>
```

## *Actividad 8. ABM básico, parte 4. Alta.*

Para permitirle al usuario dar de alta un nuevo registro, vamos a crear un nuevo archivo al que llamaremos ALTA\_USUARIO.PHP

Dicho archivo lo vamos a crear a partir de la plantilla de nuestra aplicación, porque necesitamos interactuar con quien va a dar de alta el registro y queremos que la página tenga el estilo que venimos definiendo en nuestra aplicación.

**PASO 1)** Creamos el archivo de altas, modificamos la etiqueta TITLE, el título de la página y le quitamos el menú de navegación.  
Lo llamaremos ALTA\_USUARIO.PHP

**PASO 2)** Modificamos los datos del enlace en la página principal.

**PASO 3)** En este punto, tenemos dos opciones, puede ser que el usuario intente cargar un registro que ya estaba cargado, en cuyo caso deberíamos avisarle y si no estaba cargado, le permitiremos que lo haga.

Vamos a generar un formulario simple para que el usuario pueda cargar un nuevo registro, pero vamos a revisar que dicho registro no este previamente

cargado, teniendo en cuenta que el campo USUARIO es la clave primaria de nuestra tabla.

**PASO 4)** Declaramos una variable para mostrar mensajes al usuario, dicha variable se podrá modificar desde otras páginas mediante GET, por lo que deberemos verificar si el GET existe, y de ser así, pisamos el valor previamente cargado en la variable.

```
$mensaje = 'Ingrese los nuevos datos';
if(isset($_GET['mensaje'])){
    if($_GET['mensaje']=='uno'){ $mensaje = 'El usuario ya existe en la base';}
}
```

**PASO 5)** Maquetamos el formulario y le agregamos la variable que creamos para los mensajes.

```
<!-- FORMULARIO -->
<div class="container">
    <div class="row">
        <div class="col-3"></div>
        <div class="col-6">
            <form action="alta_usuario_sql.php" method="post">
                <div class="form-group">
                    <label for="usuario" style="color:green" class="font-weight-bold">Ingrese el usuario</label>
                    <input type="text" id="usuario" name="usuario" placeholder="Ingrese su usuario" class="form-control">
                </div>
                <div class="form-group">
                    <label for="clave" style="color:green" class="font-weight-bold">Ingrese la clave</label>
                    <input type="password" id="clave" name="clave" placeholder="Ingrese su clave" class="form-control">
                </div>
                <div class="form-group">
                    <label for="rol" style="color:green" class="font-weight-bold">Ingrese el rol</label>
                    <input type="text" id="rol" name="rol" placeholder="Ingrese su rol" class="form-control">
                </div>
                <button type="submit" class="btn btn-primary btn-block">Cargar registro</button>
                <?php echo $mensaje; ?>
            </form>
        </div>
    </div>
</div>
```

**PASO 6)** Creamos el archivo que recibe los datos del formulario. Es una página encargada de procesar los datos, por lo que solo necesitaremos usar PHP y SQL.

Lo llamaremos ALTA\_USUARIO\_SQL.PHP y deberá estar conectado a la BDD.

**PASO 7)** En el archivo que recibe los datos, guardamos dichos datos en variable para ser utilizados.

```
// Recibo los valores por POST.  
$p_usuario = $_POST['usuario'];  
$p_clave = $_POST['clave'];  
$p_rol = $_POST['rol'];
```

**PASO 8)** Verificamos si el usuario a cargar, ya existe en la BDD.

Para esto, lo que vamos a hacer es tomar el nuevo usuario que se intenta cargar, y contamos cuantas veces aparece en la base de datos. Si aparece una vez, lo que vamos a hacer es modificar nuestra variable de mensajes y devolver al usuario nuevamente al formulario.

```
// Verificamos si el nuevo usuario ya existe en la BDD  
$consulta1 = "select count(distinct usuario) as nuevo from usuario where usuario = '$p_usuario' ";  
  
$resultado1 = mysqli_query( $conexion, $consulta1);
```

```
while($a = mysqli_fetch_assoc($resultado1)){  
    $existe = $a['nuevo'];  
}
```

```
// Estructura de decision.  
if($existe==1){  
    // Modificamos Mensaje y volvemos al formulario.  
    header("Location: alta_usuario.php?mensaje=uno");  
}
```

Si probamos de cargar un usuario que ya existe en la base, la aplicación nos devolverá automáticamente al formulario y veremos el mensaje modificado.

## Ingrese el usuario

## Ingrese la clave

## Ingrese el rol

El usuario ya existe en la base

**PASO 9)** Si comprobamos que el usuario no existe en la base, entonces le permitimos darlo de alta, mediante una consulta de SQL utilizando la sentencia INSERT.

Luego de ejecutar la inserción, cerramos la página y actualizamos la anterior mediante JS.

Opción 1:

```
// Estructura de decision.
if($existe==1){
    // Modificamos Mensaje y volvemos al formulario.
    header("Location: alta_usuario.php?mensaje=uno");
} else{
    // El usuario NO existe, permitimos darlo de alta.
    $alta = "insert into usuario values('$p_usuario','$p_clave','$p_rol')";
    $resultado_alta = mysqli_query($conexion,$alta);

    // Cierro el formulario y recargo la pagina anterior.
    echo "<script lenguaje='javascript' type='text/javascript'>
        | window.opener.document.location.reload();self.close()
        | </script>";

    echo "<script lenguaje='javascript' type='text/javascript'>
        | window.close();
        | </script>";
}
```

Opción 2:

\*\*\* NO USAR EL TARGET \_BLANCK Y UTILIZAR ESTE CÓDIGO:

```
// Estructura de decision
if($existe==1){
    header("Location: alta_usuario.php?mensaje=uno");
}else{
    $alta = "insert into usuario values ('$usuario','$clave','$rol')";
    $resultado_alta = mysqli_query($conexion,$alta);

    header("Location: bdd.php");
}
```

### *Baja física de un registro.*

Vamos a desarrollar ahora la parte de nuestro ABM que se encargará de eliminar definitivamente de la base de datos, los registros que ya no necesitamos.

Vamos a proceder de manera similar a como lo hicimos para dar de alta un nuevo registro, dirigiremos al usuario a una nueva pestaña del navegador, donde le mostraremos los datos a eliminar y le pediremos que confirme o cancele.

**PASO 1)** Creamos el archivo de eliminación, modificamos la etiqueta TITLE, el título de la página y le quitamos el menú de navegación.

Lo llamaremos BAJA\_USUARIO.PHP

**PASO 2)** Modificamos los datos del enlace en la página principal.

**PASO 3)** Desde el archivo principal, BDD.PHP, le enviamos al archivo de baja, los datos del registro que se quiere eliminar para mostrarselos al cliente y que confirme que desea eliminarlos.

```
<a href='baja_usuario.php?usuario=".$d['usuario']."
&clave=".$d['clave']."'
&rol=".$d['rol']."'
target='_blank'>Eliminar</a>
```

**PASO 4)** En el archivo de bajas, recibimos los datos enviados desde la tabla.

```
<!-- Recibo los valores por GET -->
<?php
$usuario = $_GET['usuario'];
$clave = $_GET['clave'];
$rol = $_GET['rol'];
```

**PASO 5)** También en el archivo de bajas, maquetamos un formulario para mostrar los datos a eliminar.

Podemos reutilizar el formulario de alta y editarlo, cambiando las etiquetas LABEL, eliminando el PLACEHOLDER y agregando VALUE y READONLY.

**PASO 6)** Agregamos un botón para cancelar la operación.

A cada botón, le agregamos un atributo VALUE diferente para poder identificar cuál de los dos fue presionado y el atributo NAME.

```
<button type="submit" class="btn btn-primary btn-block" name="boton" value=1>
<button type="submit" class="btn btn-danger btn-block" name="boton" value=0>
```

**PASO 7)** Creamos la página BAJA\_USUARIO\_SQL donde se ejecutará la consulta que eliminará al registro seleccionado o se cancelará la operación según decida el cliente.

Lo primero que hacemos es incluir el archivo de conexión y recibir los valores del formulario.

```
// Conexion al servidor
include('includes/conexion.php');

// Recibo los valores por POST.
$p_usuario = $_POST['usuario'];
$p_clave = $_POST['clave'];
$p_rol = $_POST['rol'];
$p_boton = $_POST['boton'];
```

Si el valor recibido en el elemento BOTON es cero, entonces el cliente eligió cancelar la operación.

En cambio, si dicho valor es uno, entonces procedemos con la eliminación del registro.

**PASO 8)** Verificamos si el valor del botón es cero, en cuyo caso, redirigimos al usuario nuevamente a la página principal, la que contiene la tabla.

```
// Verifico si boton = cero
if($p_boton==0){
    // Cierro el formulario y recargo la pagina anterior.
    echo "<script lenguaje='javascript' type='text/javascript'>
        window.close();
    </script>";
}
```

**PASO 9)** Verificamos si el valor del botón es uno, y de ser así, procedemos a eliminar el registro.

```
$baja = "delete from usuario where usuario = '$p_usuario' ";
$resultado_baja = mysqli_query($conexion,$baja);
```

Luego de ejecutar la consulta SQL, cerramos la página y recargamos la página anterior para que se actualicen los resultados que se ven en la tabla. Para esto utilizamos la opción 2 que vimos cuando hicimos el alta de nuevo usuario.