

GUIÓN PARA LA REALIZACIÓN DEL PROYECTO FINAL

2º DWES
DAW – AULA
2024 - 2025

Tabla de contenidos

1. Tarea y objetivos
2. Requisitos
3. Diseña y crea la base de datos
4. Maquetando
5. Estructura de directorios para implementar el MVC
6. Conexión a la base de datos
7. Namespaces
8. Rutas amigables
9. Registro de usuarios
 - 9.1. Modificar el registro de usuarios
10. Login de usuarios
11. Cerrar sesión
12. Validando formularios
13. Gestionar categorías
14. Gestionar productos
15. Mostrando productos
16. El carrito de la compra
17. Procesamiento del pedido
18. Envío de correo
19. Mis pedidos
20. Confirmación de Registro
21. Header y Footer
22. Cookies
23. Pagos
24. Despliegue
25. Fecha de entrega
26. Evaluación

1. Tarea y objetivos

Se desea crear una aplicación de pedidos para una tienda que nos permita:

- Cargar dinámicamente categorías y productos disponibles
- Controlar el estado de un pedido
- CRUD de productos y pedidos
- Controlar el acceso con una tabla de usuarios
- Gestión de carrito
- Gestión de pagos
- Enviar correo de confirmación.

OBJETIVOS.

- Crear una aplicación web funcional, que cumpla los requisitos según el guion establecido. Que genere salidas válidas adecuadas a los requisitos.
- Crear un código limpio, organizado y legible siguiendo las buenas prácticas del tema 9.
- Evitar la duplicidad de código. Implementación eficiente con uso de librerías.
- Comentar el código.
- Realizar en el servidor una validación robusta de las entradas de los formularios. Se producirán mensajes de error claros.
- Crear y usar adecuadamente clases y objetos para el desarrollo de la solución. Clases bien definidas y objetos funcionales.
- Usar sesiones y cookies, sin fallos de seguridad.
- Realizar la conexión a la base de datos orientada a objetos PDO. Usar una clase para la implementación
- Usar sentencias preparadas en las consultas a la base de datos. Impedir inyección SQL. Validación
- Crear un CRUD utilizando PDO sobre al menos una de las tablas de la base de datos. Recuerda validar los datos y usar siempre sentencias preparadas.
- Implementar el patrón MVC. Atención a las responsabilidades de las capas. Separación clara de la lógica.
- Usar rutas amigables. (.htaccess)
- Programar adecuadamente el registro e identificación de usuarios en el sistema.
- Utilizar Composer para el autoload.
- Utilizar librerías externas (Composer) para ocultar credenciales, paginar, generar PDFs, etc.
- Utilizar un enrutador (clase Router) que se encargue de redirigir la solicitud del usuario a la acción correspondiente en la aplicación.
- Utilizar la carpeta public (la única que debería ser visible en nuestro servidor web) con el archivo index.php que nos permita pasar todas las peticiones.

2. Requisitos

Se desea crear una aplicación para el departamento de pedidos de una tienda.

La aplicación debe permitir:

1. Consultar categorías
2. Crear categorías para usuarios administradores
3. Consultar productos
4. Crear, editar y borrar productos para usuarios administradores
5. Gestión de pedidos para administradores
6. Añadir una o más unidades de un producto al pedido
7. Consultar el pedido del carrito y eliminar productos de este.
8. Realizar el pedido, introduciéndolo en la base de datos y enviando correo
9. Gestión del pago

Se deben de tener en cuenta las siguientes premisas:

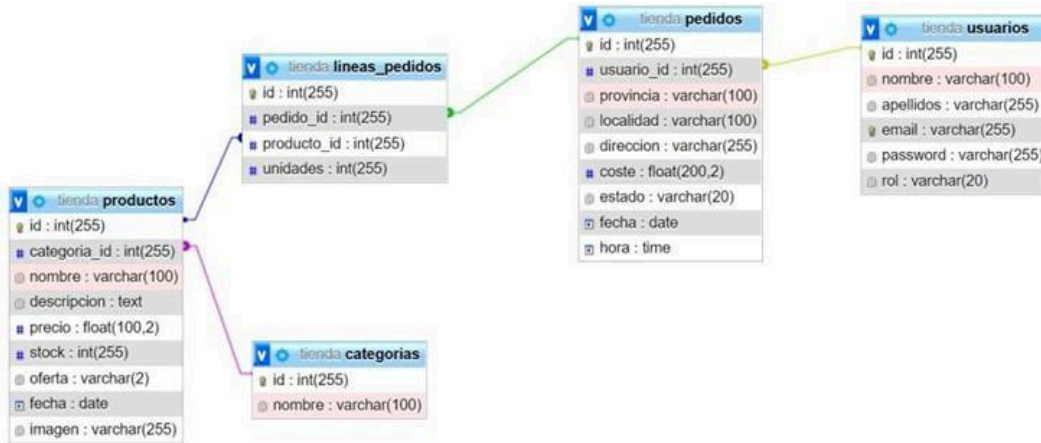
- Para acceder a la aplicación será necesario autenticarse.
- De cada categoría se guarda su código, su nombre y su descripción
- De los productos, su código, nombre, descripción, precio, stock, oferta, fecha, imagen y la categoría a la que pertenecen. Cada producto pertenece a una categoría.
- De los pedidos se conoce su código, la dirección, provincia, localidad, coste, estado, fecha y hora. Un pedido constará de una o varias líneas de pedido. De cada una de ellas se almacena un identificador, las unidades, el código del producto y el código del pedido.
- Los pedidos se introducen en la base de datos como no enviados. Cuando se envíen el departamento de pedidos los marcará como enviados. Nuestra aplicación no se ocupa de esto.
- De los usuarios se guarda: el código, nombre, apellidos, el correo electrónico (se usa como nombre de usuario para identificarse), la clave y el rol. Para simplificar este ejemplo no almacenamos datos como dirección, teléfono, etc. y suponemos que es suficiente con la dirección del pedido. (Se sugiere que en el futuro se tenga en cuenta, así como la posibilidad de guardar direcciones diferentes de facturación, tarjeta usada normalmente, etc.)
- Cualquier otro dato que se suponga de interés podrá modificarse y añadirse adecuadamente en la base de datos.
Es fundamental que antes de empezar a trabajar se realice un **análisis de requisitos** para obtener toda la información necesaria.

3. Diseña y crea la base de datos

Una de las cosas que tendremos claras después del análisis de requisitos será que datos van a ser persistentes, así como el repositorio donde se van a almacenar.

En este primer ejemplo usaremos una base de datos relacional y para su diseño se usará el **esquema Entidad-Relación**.

A continuación, se muestra un diagrama con un ejemplo básico de lo que puede ser la base de datos de esta aplicación.



Si haces cambios deberás aportar el archivo que permita la creación completa de la base de datos usada y un diagrama o esquema Entidad-Relación de la misma.

4. Maquetando

Diseña la estructura básica de tu web.

Intenta realizar el diagrama de flujo de pantallas por las que pasa el usuario al realizar una operación.

El siguiente ejemplo no tiene por qué coincidir con el diseño de tu aplicación:



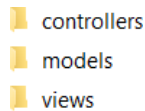
5. Estructura de directorios para implementar el MVC

Recordemos que la idea básica de este patrón es separar nuestros sistemas en tres capas, el Modelo, la Vista y el Controlador.

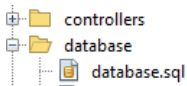
- El **Modelo** se encarga de todo lo que tiene que ver con la persistencia de datos. Guarda y recupera la información del medio persistente que utilicemos, ya sea una base de datos, ficheros de texto, XML, etc.
- La **Vista** presenta la información obtenida con el modelo de manera que el usuario la pueda visualizar.
- El **Controlador**, dependiendo de la acción solicitada por el usuario, es el que pide al modelo la información necesaria e invoca a la plantilla (de la vista) que corresponda para que la información sea presentada.

Por lo tanto, vamos a crear una estructura de directorios que nos permita implementar el Modelo Vista Controlador.

Comenzamos creando una estructura de directorios parecida a la siguiente:



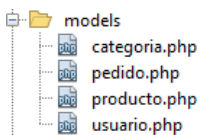
Añade una carpeta para guardar el script con la creación de la base de datos y sus tablas. De esta forma siempre estará disponible para la corrección de este ejemplo.



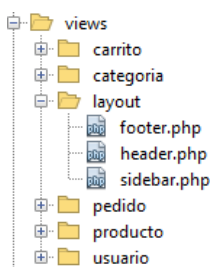
También es aconsejable que crees una carpeta para tener tus hojas de estilo y las imágenes que usarás en el diseño de tu aplicación. En

cada una estas carpetas podemos crear también los archivos que ya sabemos que serán necesarios.

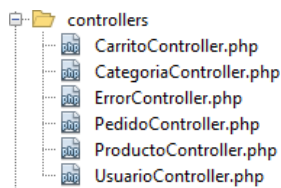
Por ejemplo, en la carpeta models tendremos que tener un archivo para cada una de las clases que nos permitirán recuperar la información de la base de datos. Por tanto, crea los archivos usuario.php, pedido.php, categoria.php y producto.php.



En la carpeta views crearemos una subcarpeta para cada una de las entidades de nuestra base de datos. Además, si hemos decidido dividir el código de nuestras páginas, para reutilizar las diferentes secciones como la cabecera o el pie, también incluiremos estos archivos en las vistas. Crea una carpeta específica para ello.



En la carpeta controllers crea los archivos UsuarioController.php, PedidoController.php, CategoriaController.php y ProductoController.php



Como puedes observar, he añadido dos más. Una para la gestión del carrito y otra para controlar los errores 404.

Cuando un recurso que se ha solicitado al servidor no está lo gestionamos con ErrorController.

Recuerda que esta estructura de carpetas se hace para que sea más didáctico el trabajo. En un tu proyecto real los nombres de las carpetas no coincidirán.

6. Conexión a la base de datos

Aprovecha el archivo `config.php` para guardar como constantes los datos necesarios para conectar con nuestra base de datos.

Crea una nueva carpeta llamada "Lib" y dentro de ella una clase que nos permita hacer la conexión a la base de datos utilizando los valores almacenados en el archivo `.env`.

Aprovecha el archivo `.env` en la raíz de tu proyecto para almacenar de forma segura las variables necesarias para conectar con la base de datos. Estas variables deben incluir los siguientes datos:

`DB_HOST`: El host donde se encuentra la base de datos (por ejemplo, `localhost`).

`DB_NAME`: El nombre de la base de datos.

`DB_USER`: El usuario que tiene permisos para acceder a la base de datos.

`DB_PASSWORD`: La contraseña del usuario.

Para cargar estas variables en tu aplicación, puedes utilizar una biblioteca como `PHP-Dotenv`, que permite leer y utilizar las variables del archivo `.env` de manera sencilla.

En los apuntes de clase del tema 9, en el punto 9, ocultando credenciales, se explica cómo crear el `.env`, `.gitignore` y `.config.php`

7. Namespaces

Desde el principio haz uso de Namespaces.

Recuerda que en cada una de las clases declaramos el namespace. Es decir, la carpeta en la que está esa clase.

8. Rutas amigables (Opcional)

Para que las URLs sean legibles por los humanos deben reflejar el contenido de nuestro sitio web.

Un URL amigable o limpio no muestran los caracteres ? ni =, ni tampoco aparece el nombre del fichero php.

Para conseguirlo haremos uso del enrutamiento de URL con el archivo **.htaccess** (hypertext access) que ubicaremos en la raíz de documentos de nuestro servidor.

Crea este archivo de texto plano (sin olvidar el punto) .

No olvides hacer los cambios necesarios para que el módulo mod_rewrite esté activado.

A continuación, escribe las reglas necesarias en el archivo .htaccess

El resultado podría ser algo parecido a esto:

```
.htaccess
1  # Activar el módulo mod_rewrite
2  RewriteEngine On
3
4  # Redirigir todas las peticiones al archivo index.php
5  RewriteCond %{REQUEST_FILENAME} !-f
6  RewriteCond %{REQUEST_FILENAME} !-d
7  RewriteRule ^ index.php [QSA,L]
8  ~~~
```

9. Registro de usuarios

- Comenzaremos escribiendo el contenido del controlador UsuarioController .
 - Crearemos dentro de este controlador un método que nos permitirá cargar la vista con el formulario de registro(archivo dentro de la carpeta Views/usuario, llamado por ejemplo formregistro.php).
- Desde la vista devolveremos los datos al controlador usando el método POST.
- UsuarioController gestionará cómo almacenar lo que nos ha llegado a través del formulario en la base de datos((no olvides validar todos los datos) .
- Para almacenar los datos se creará una consulta a la base de datos usando la clase PDO.
- Por último modifica la vista, usando sesiones, para indicar que todo ha ido correctamente.
- Recuerda que esos mensajes tendrán que desaparecer posteriormente para poder realizar nuevos registros. Puedes hacerlo eliminando las sesiones que hemos usado en la vista registro.

9.1. Modificar el registro de usuarios

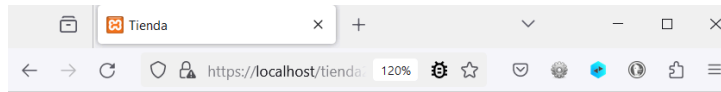
Modifica el código para que los usuarios con el rol de administrador puedan crear usuarios con rol administrador o con rol 'user'.

El resto de usuarios sólo podrán hacer su propio registro.

Recuerda que una vez que se guarda un usuario en la base de datos ya puedes conocer otras propiedades como por ejemplo el id o el rol. Realiza las modificaciones oportunas para que el objeto correspondiente de la clase Usuario disponga de dicho valores.

10. Login de usuarios

- En el controlador UsuarioController crearemos un método que nos permita hacer el login de los usuarios.
- Solicita a través de un formulario el correo y la contraseña para identificar al usuario:



TIENDA

[Identificate](#) [Crear cuenta](#)

Login

Email

Contraseña

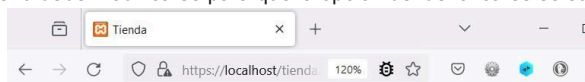
Enviar

Aplica css para que quede bonito.

- En nuestro caso, sabemos que el email no puede estar repetido y que es el campo que se solicita al usuario para la identificación junto con la contraseña.
 - Por lo tanto, es necesario que busquemos en nuestra base de datos si existe algún usuario que tenga el email que nos han proporcionado.
 - Una vez localizado dicho usuario tendremos que comprobar si la contraseña que tenemos guardada en la base de datos coincide con la que nos acaban de dar a través del formulario.
 - Para comprobarlo usa `password_verify()`. Esta función nos devuelve true o false. Ejemplo :

```
$verify = password_verify($password, $usuario->password);
```

- Una vez que hemos comprobado que el correo existe y que la contraseña es la correcta daremos acceso al usuario identificado a nuestra aplicación.
- Se recomienda crear una variable de sesión donde se guarden los datos del usuario identificado.
- Por último, redirigimos al usuario logueado a la página principal. Añadiremos en dicha página el nombre y los apellidos del usuario. (Los datos están en la variable de sesión).
- Recuerda que una vez logueado el menú debe modificarse para que la opción de identificarse se sustituya por cerrar sesión.



TIENDA

Gloria Fuertes

[Cerrar sesión](#) [Mis pedidos](#)

En el próximo apartado implementaremos la opción de cerrar sesión.

11. Cerrar sesión

Tendremos que borrar la sesiones añadiendo al controlador `UsuarioController` el método `logout()`.

Este método borrará la sesión que tiene identificado al usuario.

12. Validando formularios

- Valida todos tus formularios.
- Recuerda que cualquier entrada puede producir errores graves en nuestra aplicación.
- Presta especial interés si los datos van a formar parte de una consulta a la base de datos
- Evita la inyección SQL

13. Gestionar categorías

En este apartado sólo nos centraremos en **crear y listar categorías**. Se propone que, después, vosotros desarrolléis el resto de la funcionalidad necesaria en la aplicación.

- Para la gestión de las categorías necesitamos una nueva clase llamada `CategoriaController`
- También se debe crear el modelo en el archivo `Models/Categoria.php`
- Las vistas las crearemos en la carpeta `Views/categoria`:
 - Para recoger los datos de la categoría que vamos a crear se necesita una vista donde se solicite, al menos, el nombre de la categoría a crear.
 - Para mostrar las categorías que existen en nuestra aplicación se creará una vista con el nombre `index.php`.
- Debemos tener en cuenta que no todos los usuarios están autorizados para crear categorías, sólo pueden hacerlo los que tienen el rol de administrador.
 - Hay que comprobar si el usuario se ha identificado como administrador. Podemos comprobarlo haciendo uso de sesiones.
 - Para comprobar si un usuario es administrador crearemos un método estático llamado `isAdmin()` que comprueba si la sesión es de administrador. Este método se puede crear en una nueva clase llamada `Utils`.
- No olvides validar los datos.
- El controlador será el responsable de solicitar al servicio la información de la base de datos, por ejemplo para poder mostrar las categorías que existen. Igualmente, deberá de existir un método en el controlador para gestionar almacenar las categorías.

14. Gestionar productos

- En nuestra aplicación debe existir el modelo Producto .
- Crea en el controlador PedidosController un método llamado "gestion" y en la carpeta views/productos un archivo llamado gestion.php.
 - El método gestión permitirá obtener todos los productos y mandar a la vista los datos obtenidos.
 - Para comprobar que funciona correctamente puedes escribir directamente algún producto en la tabla (usa phpMyAdmin o la correspondiente sentencia SQL en la consola).

CREAR PRODUCTO

- Necesitaré una vista con un formulario que me solicite los datos del producto.
- En ProductoController pondré un método para crear productos. Este método hará uso de una vista para solicitar los datos del producto.
- Antes de insertar un producto hay que verificar que el usuario tiene el rol de administrador (ya tenemos un método que nos permite comprobarlo).
- Recuerda que para poder poner la categoría a la que pertenece el producto debemos mostrar las categorías en el formulario.
- En ProductoController recibiré todos los datos del formulario y lo insertaremos en la base de datos. Recuerda que hemos acordado que no guardaremos la imagen del fichero en la base de datos, sino el nombre del fichero.
- Habilitaremos una carpeta para guardar todas las imágenes de nuestros productos. Es importante comprobar, antes, si existe la carpeta, y si no es así hay que crearla.
- Si todo va correctamente , en la vista debería de mostrar un mensaje. Podemos controlarlo mediante sesiones y cuando acabemos se borra la sesión.

15. Mostrando productos

- Crea una vista que nos permita mostrar todos los productos de una categoría que haya sido seleccionada.
- Cuando la base de datos tenga un número elevado de productos, nuestra página debe de permitir cargarlos poco a poco. Puedes utilizar [Composer](#) para instalar una librería de paginación y con ella mostrar todos los productos de una categoría.

16. El carrito de la compra

- Cualquier usuario que navegue por nuestro sitio web podrá ver nuestros productos, seleccionar los que desea comprar y por último realizar el pedido. Sólo se solicitará la identificación en el momento que desee confirmar el pedido. En el caso de que aún no sea uno de nuestros clientes se le pedirá que previamente se registre.
- Los usuarios seleccionarán aquellos productos que desean ir añadiendo al carrito.
- En nuestro carrito tendremos la opción de añadir más unidades del mismo producto, eliminar unidades o incluso borrar el carrito por completo. En cualquier caso, el usuario podrá seguir comprando otros productos hasta que decida confirmar el pedido.
- Para almacenar el carrito se utilizará una variable de sesión. El carrito será un array asociativo en el que almacenaremos para cada producto, que el cliente haya seleccionado, tanto el identificador del producto, como el número de unidades pedidas de ese producto.

Resumiendo, la variable de sesión del carrito será un array asociativo en el que las claves de los elementos representan el código o identificador de un producto, y el valor, el número de unidades pedidas de este producto.

El array comenzará estando vacío y cuando se añada un producto al pedido, hay que comprobar si ya hay en el array algún elemento que tenga como clave el código del producto. Si no lo hay, se añade un nuevo elemento.

Por ejemplo, si el carrito está vacío y se añaden 2 unidades del producto con código 4, se creará un elemento del array con clave 4 y valor 2.



Figura 4.5
Carrito de la compra con un elemento.

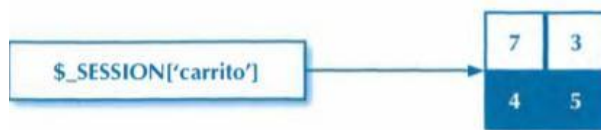
Ejemplo: Si posteriormente se añaden 3 unidades del producto con código 5, el carrito será:



Figura 4.6
Se añade un nuevo producto.

En el caso de que haya un elemento con ese código, se suman las unidades al valor actual del elemento.

Ejemplo: Si se añaden otras cinco unidades del producto con código 4, el resultado será:



- De la misma manera, al eliminar unidades de un producto que tenemos en el carrito habrá que buscar el elemento correspondiente en el array y restarle las unidades.

Cuando se eliminan todas las unidades de un producto, se suprime el elemento correspondiente en el array.

- Por supuesto en cualquier momento el usuario puede decidir vaciar el carrito de la compra, en cuyo caso lo que tenemos que hacer es borrar la variable de sesión.
- Para gestionar el carrito de la compra crea un nuevo controlador llamado CarritoController.php.

En este controlador estarán los métodos que nos permitirán añadir productos al carrito, borrar elementos del carrito, añadir más unidades a un producto que ya está en mi carrito, quitar unidades de un producto que está en el carrito y vaciar el carrito de la compra.

- También es necesario crear una vista que nos permita ver el contenido de nuestro carrito.

Para poder mostrar la imagen, el precio, el nombre del producto, etc. recuerda que en cualquier momento puedes cargar todos los datos de los productos de tu carrito, ya que conoces los identificadores de los productos (están almacenados en la variable de sesión).

Si ya has realizado el apartado dedicado a productos, puedes gestionar fácilmente la obtención de dicha información.

- Es importante que también muestres el número total de productos que hay en el carrito, así como el importe total del pedido que se va a realizar (puedes añadir subtotales por cada producto si lo deseas). Para conseguir este objetivo puedes crear un método en la clase `Utils`.
- No olvides incluir en la vista un botón que permita realizar el pedido de todo lo que tenemos en nuestro carrito.

Posteriormente, veremos cómo procesar el pedido y enviar un correo con la confirmación a nuestro cliente.

17. Procesamiento del pedido

El proceso consiste en :

- insertar el pedido en nuestra base de datos
- Si el pedido se inserta correctamente:
 - decrementar el stock de los productos en la base de datos
 - mandar un correo de confirmación al cliente
 - vaciar el carrito

El método que se encarga de insertar el pedido en la base de datos recibe el carrito de la compra y el identificador del cliente que realiza el pedido.

Con esta información transformamos el carrito de la compra en un pedido, para lo cual:

- creará una nueva fila en la tabla pedidos y
- creará una fila en la tabla lineas_pedido por cada producto diferente que se pida. Ten en cuenta que necesitas usar la clave del nuevo pedido.

Las inserciones deben ocurrir como una transacción. Si alguna falla, hay que deshacer las anteriores (uso de commit).

Es muy importante que se actualice adecuadamente el stock de productos. Si el pedido se realiza tendremos que decrementar el stock de todos los productos que acabamos de vender. En caso contrario, no hay que modificarlo.

Si habéis observado los campos de la tabla "pedidos" habréis podido comprobar que existe el campo " estado". Los estados pueden ser, por ejemplo: confirmado, enviado, finalizado, etc.

Cuando procesemos un pedido debemos de informar al usuario por dos medios: en la propia web y enviando un correo con todos los datos del pedido: número de pedido, productos, cantidad de cada uno de ellos, total del importe, lugar de envío, etc.

18. Envío de correo

Si el pedido se ha realizado correctamente se enviará un correo de confirmación al cliente. El correo incluirá el número del pedido, el nombre del cliente que lo realiza, los productos del pedido y el importe total del pedido.

Con [Composer](#) instalaremos la librería PHPMailer.

Para la gestión del correo crearemos una nueva clase en la carpeta Lib que se usará PHPMailer.

En la clase que gestionar el correo existirá un método que reciba el carrito de la compra, el número de pedido y al menos el correo de nuestro cliente.

Deberemos utilizar los datos de nuestra cuenta de mailtrap. Si quieres usar otro servicio de correo puedes usarlo.

Recuerda que cualquier credencial, contraseña o dato susceptible debe ocultarse y que para ello tenemos el fichero .env

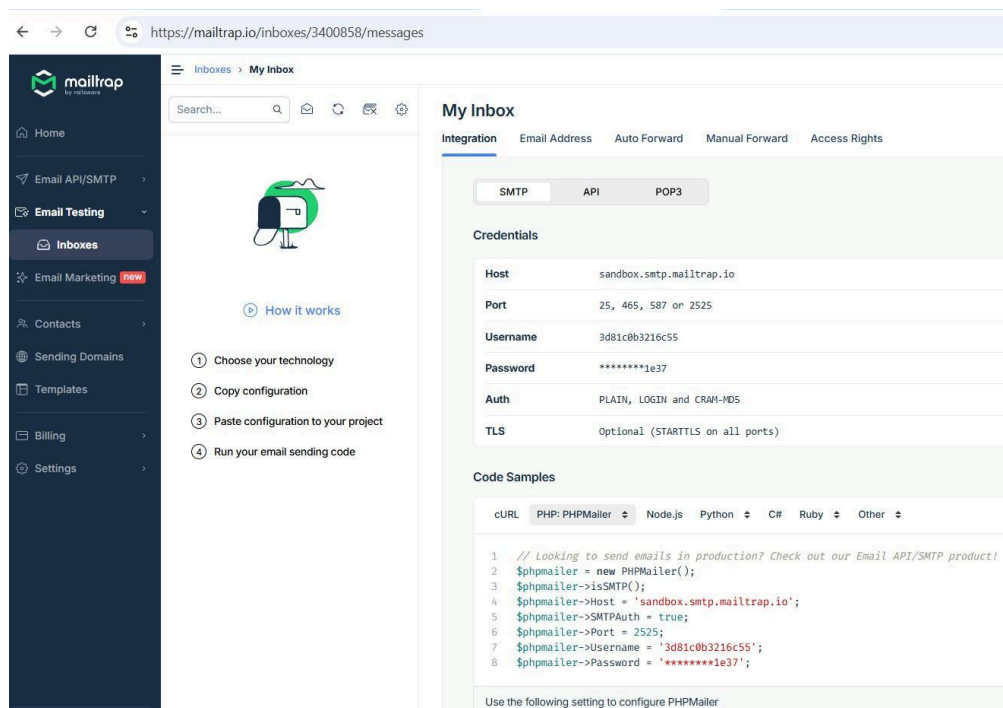
Recomendación:

Mientras estamos en desarrollo es aconsejable poder comprobar que todos los correos se envían y se reciben correctamente.

[Mailtrap](#) proporciona un servidor SMTP falso para probar, ver y compartir correos electrónicos enviados desde los entornos de preproducción, y probar con datos reales sin el riesgo de enviar spam a clientes reales.

Crea una cuenta gratuita en [Mailtrap](#) y usa esas credenciales en la clase de tu aplicación que gestionará el correo.

Busca los datos para realizar la configuración en PHPMailer. En la siguiente imagen se ven mis datos. No los copies , puesto que después de capturar esta imagen haré un reseteo y no te servirán.



The screenshot shows the Mailtrap web interface in a browser. The address bar displays `https://mailtrap.io/inboxes/3400858/messages`. The left sidebar contains navigation links: Home, Email API/SMTP, Email Testing, Inboxes (selected), Email Marketing (new), Contacts, Sending Domains, Templates, Billing, and Settings. The main content area is titled 'My Inbox' and includes tabs for Integration, Email Address, Auto Forward, Manual Forward, and Access Rights. Under the 'Integration' tab, there are three sub-tabs: SMTP (selected), API, and POP3. The 'Credentials' section lists the following details:

Field	Value
Host	sandbox.smtp.mailtrap.io
Port	25, 465, 587 or 2525
Username	3d81c0b3216c55
Password	*****1e37
Auth	PLAIN, LOGIN and CRAM-MD5
TLS	Optional (STARTTLS on all ports)

Below the credentials, there is a 'Code Samples' section with a dropdown menu showing 'cURL', 'PHP: PHPMailer' (selected), 'Node.js', 'Python', 'C#', 'Ruby', and 'Other'. The PHP code sample is as follows:

```
1 // Looking to send emails in production? Check out our Email API/SMTP product!
2 $phpmailer = new PHPMailer();
3 $phpmailer->isSMTP();
4 $phpmailer->Host = 'sandbox.smtp.mailtrap.io';
5 $phpmailer->SMTPAuth = true;
6 $phpmailer->Port = 2525;
7 $phpmailer->Username = '3d81c0b3216c55';
8 $phpmailer->Password = '*****1e37';
```

At the bottom, a note states: 'Use the following setting to configure PHPMailer'.

19. Mis pedidos

Escribe el código necesario para que un cliente pueda consultar sus pedidos y el estado de cada uno de ellos.

20. Confirmación Registro (Opcional)

Modifica el registro de usuarios para que sólo sea posible cuando el usuario haya confirmado su identidad mediante un correo con un token que recibirá. Como usando mailtrap los correos solo llegan al usuario que se ha dado de alta en mailtrap, éste será el encargado de verificar el token de confirmación. Una vez verificado el usuario registrado ya podrá entrar en su cuenta. Para ello deberemos añadir un campo extra en la tabla de usuarios que podría llamarse: "Verificación_Email" o algo por el estilo.

21. Header y Footer

Implementar una vista header y footer para nuestra aplicación web. El header deberá llevar un botón de inicio, cargar las categorías de forma dinámica y un botón de contacto.

Crea el footer como tu desees.

22. Cookies

Cookie de recordar usuario: Deberás implementar una cookie que recuerde al usuario logueado durante 7 días. Para ello en el login deberás ofrecer esta opción.

Cookie del carrito de la compra: Una vez añadida el usuario los productos al carrito de la compra, estos estarán disponibles durante los próximos 3 días hasta que se realice el pedido. Una vez realizado el pedido esta cookie se deberá destruir.

23. Pagos

(Opcional)

Investiga cómo añadir pagos a tu proyecto usando Paypal.

24. Despliegue

(Opcional)

Adicionalmente se podrá alojar la aplicación web en un hosting gratuito o de pago a elección del alumno para mostrar su funcionamiento.

25. Diseño bonito

Utiliza css y bootstrap a tu gusto para hacer que tu sitio web tenga un diseño atractivo.

26. Github

Haz un commit en github por cada fichero que crees, modifiques sustancialmente o por cada nueva funcionalidad que implementes.

27. Fecha de Entrega

La fecha límite de entrega del proyecto serán las 0:00 horas del día 23 de Febrero. Es decir, como muy tarde se podrá entregar a las 23.59 horas del sábado 22 de Febrero.

El proyecto se entregará a través de la plataforma Moodle. El alumno deberá entregar el proyecto junto con la base de datos y una documentación del mismo. Todo esto se deberá subir comprimido a Moodle. En caso de que por algún motivo no se pueda subir me lo podéis enviar al email jalfpin474@g.educaand.es . Pero solo como última opción.

La documentación incluirá:

- Portada
- Índice
- Estructura del proyecto
- Breve explicación de cada uno de los ficheros del proyecto. No hace falta añadir el código de los ficheros. Deberéis ir explicando qué hace cada uno de los ficheros de la estructura.

28. Evaluación

El proyecto se evaluará del siguiente modo:

	Puntuación
- Documentación:	0.3 puntos.
- Clase para la conexión a la base de datos:	0.5 puntos.
- Uso de namespaces:	0.5 puntos.
- Rutas amigables (Opcional):	0.2 puntos.
- Registro de usuarios:	0.5 puntos.
- Modificación de datos de usuario en rol normal:	0.3 puntos.
- Modificación de datos de usuario en rol admin:	0.5 puntos.
- Uso de sesiones:	0.5 puntos.
- Uso de cookies:	0.5 puntos.
- Implementación del logout del usuario:	0.3 puntos.
- Validación de datos de formularios:	0.5 puntos.
- Gestión dinámica de categorías:	0.5 puntos.
- Gestión de productos:	0.5 puntos.
- El carrito de la compra:	0.5 puntos.
- Procesamiento de los pedidos:	0.5 puntos.
- Envío de correo con datos de pedidos	0.5 puntos.
- Envío de correos de confirmación de registro (opcional)	0.5 puntos.
- Implementación de Header:	0.5 puntos.
- Implementación de Footer:	0.2 puntos.
- Cookie del login:	0.5 puntos.
- Cookie del carrito de la compra:	0.5 puntos.
- Pagos (Extra):	0.5 puntos.
- Despliegue (Extra):	0.5 puntos.
- Diseño "bonito" usando css/bootstrap o lo que quieras:	1 punto.
- Desarrollo del proyecto en Github (Obligatorio)	0.4 puntos.
Total:	12 puntos
- Entrega fuera de plazo:	- 3 puntos.
- No seguir el patrón MVC:	- 3 puntos.
- Que no cargue la aplicación:	- 10 puntos.
- Detectar copias entre compañeros:	- 10 puntos.