

# Documentación

<b>Estructuración del programa:</b>	<b>2</b>
<b>Funciones de la página por carpeta:</b>	<b>4</b>
config :	4
controllers :	5
carritoController :	5
categoriaController :	6
pedidoController :	7
productoController :	8
usuarioController :	8
Core	10
config	10
dataBase	10
dataBase	10
helpers	10
email Helpers	10
utils	11
lib	12
Database	12
models	13
categorias	13
pedido	14
producto	16
usuario	18
public	20
uploads	20
vendor	20
Views	21
Carrito:	21
categorías:	21
layouts :	21
pedido :	21
productos :	22
usuario :	22
.env	22
.gitignore	22
.htaccess	23
autoload.php	23
index.php	23

## Estructuración del programa:

config/

├── parameters.php

controllers/

├── carritoController.php

├── categoriaController.php

└── errorController.php

- pedidoController.php
- productoController.php
- usuarioController.php

Core/

- config.php

dataBase/

- dataBase.sql

helpers/

- emailHelpers.php
- utils.php

lib/

- Database.php

Libraries/

- Mailer.php
- Pagination.php
- PDFGenerator.php

models/

- categorias.php
- pedido.php
- producto.php
- usuario.php

public/

- css/
  - styles.css
- img/
  - logotipo.avif
- uploads/
  - images/

vendor/

views/

- carrito/
  - index.php
- categorias/
  - crear.php
  - editar.php
  - ver.php
- errors/
  - 404.php
  - error.php

```
├── layouts/
│   ├── footer.php
│   ├── header.php
│   └── sidebar.php
├── orders/
│   ├── create.php
│   ├── details.php
│   └── index.php
├── pedido/
│   ├── confirmado.php
│   ├── detalle.php
│   ├── hacer.php
│   └── misPedidos.php
├── productos/
│   ├── crear.php
│   ├── destacados.php
│   ├── edit.php
│   ├── gestion.php
│   ├── index.php
│   └── ver.php
├── usuario/
│   ├── editar.php
│   ├── gestion.php
│   ├── modificar.php
│   ├── profile.php
│   └── registro.php
├── .gitignore
├── .htaccess
├── autoload.php
├── composer.json
├── composer.lock
└── index.php
```

## Funciones de la página por carpeta:

config :







Tengo variables q voy a utilizar en todo mi programa

```
<?php

define(constant_name: "base_url",value: "http://localhost/proyectos/php/proyectoFinalPHP/");
define(constant_name: 'controller_default', value: 'productoController');
//Definición de método por defecto
define(constant_name: 'action_default', value: 'index');
```

## controllers :

Tengo todos los controladores usados en mi página

-  carritoController.php
-  categoriaController.php
-  errorController.php
-  pedidoController.php
-  productoController.php
-  usuarioController.php

## carritoController :

### 1. index

Verifica si existe el carrito en la sesión o en la cookie. Si no existe en la sesión pero sí en la cookie, carga el carrito desde la cookie. Asegura que la sesión 'carrito' está inicializada y luego carga la vista index.php.

### 2. add

Agrega un producto al carrito. Verifica si se proporciona un ID de producto válido, verifica si ya existe el carrito en la sesión, y luego añade o incrementa la cantidad del producto en el carrito. Guarda el carrito actualizado en la cookie y redirige a la página del carrito.

### 3. delete

Elimina completamente el carrito de la sesión y de la cookie. Luego redirige a la página del carrito.

### 4. deleteEspecifico

Elimina un producto específico del carrito basado en el índice proporcionado. Actualiza la sesión y la cookie con los cambios y redirige a la página del carrito.

### 5. up

Incrementa la cantidad de un producto específico en el carrito basado en el índice proporcionado. Actualiza la sesión y la cookie con los cambios y redirige a la página del carrito.

#### 6. down

Decrementa la cantidad de un producto específico en el carrito basado en el índice proporcionado. Si la cantidad llega a cero, elimina el producto del carrito. Actualiza la sesión y la cookie con los cambios y redirige a la página del carrito.

### categoriaController :

#### 1. index

Muestra todas las categorías disponibles. Crea una instancia de la clase categoria, obtiene todas las categorías usando el método getAll() y luego carga la vista cat.php para mostrarlas.

#### 2. crear

Permite a los administradores crear nuevas categorías. Verifica si el usuario es un administrador y luego carga la vista crear.php para permitir la creación de una nueva categoría.

#### 3. save

Guarda una nueva categoría en la base de datos. Verifica si se ha enviado un formulario con un nombre de categoría, crea una instancia de la clase categoria, establece el nombre y guarda la categoría en la base de datos. Luego redirige al usuario a la página de índice de categorías.

#### 4. ver

Muestra los detalles de una categoría específica y sus productos asociados. Verifica si se ha proporcionado un ID válido, obtiene los detalles de la categoría y sus productos asociados, y luego carga la vista ver.php para mostrar esta información.

#### 5. editar

Permite a los administradores editar una categoría existente. Verifica si el usuario es un administrador y si se ha proporcionado un ID válido, obtiene los detalles de la categoría a editar y carga la vista editar.php para permitir la edición.

#### 6. update

Actualiza una categoría existente en la base de datos. Verifica si el usuario es un administrador y si se han enviado los datos del formulario correctamente, actualiza la

categoría en la base de datos y establece una sesión con un mensaje de éxito o error según corresponda. Luego redirige al usuario a la página de índice de categorías.

#### 7. eliminar

Elimina una categoría de la base de datos. Verifica si el usuario es un administrador y si se ha proporcionado un ID válido, elimina la categoría de la base de datos y establece una sesión con un mensaje de éxito o error según corresponda. Luego redirige al usuario a la página de índice de categorías.

### pedidoController :

#### 1. hacer

Carga la vista hacer.php para permitir al usuario hacer un nuevo pedido.

#### 2. add

Procesa el formulario de creación de un nuevo pedido. Verifica si el usuario está autenticado, obtiene los datos del formulario y del carrito, crea un objeto Pedido, guarda el pedido y sus líneas, y luego redirige a la página de confirmación.

#### 3. confirmado

Muestra la vista de confirmación del pedido. Si el usuario está autenticado, obtiene la información del último pedido realizado por el usuario, construye un array con los detalles del pedido, envía un correo electrónico de confirmación y carga la vista confirmado.php.

#### 4. misPedidos

Muestra la lista de pedidos del usuario actual. Verifica si el usuario está autenticado, obtiene todos los pedidos del usuario y carga la vista misPedidos.php.

#### 5. detalle

Muestra los detalles de un pedido específico. Verifica si el usuario está autenticado y si se proporciona un ID válido, obtiene los detalles del pedido y sus productos asociados, y carga la vista detalle.php. Si no se proporciona un ID válido, redirige a la lista de pedidos.

#### 6. gestion

Muestra la lista de todos los pedidos para administración. Verifica si el usuario tiene permisos de administrador, obtiene todos los pedidos y carga la vista misPedidos.php con una variable que indica que es la vista de gestión.

#### 7. estado

Actualiza el estado de un pedido. Verifica si el usuario tiene permisos de administrador y si se proporcionan los datos del formulario correctamente, actualiza el estado del pedido y redirige a la vista de detalle del pedido. Si no se cumplen las condiciones, redirige a la página principal.



## productoController :

### 1. index

Muestra los productos destacados. Carga la vista destacados.php.

### 2. gestion

Muestra la lista de todos los productos para su gestión. Crea una instancia de la clase Producto, obtiene todos los productos usando el método getAll() y carga la vista gestion.php.

### 3. crear

Permite a los administradores crear nuevos productos. Verifica si el usuario es un administrador y luego carga la vista crear.php para permitir la creación de un nuevo producto.

### 4. save

Guarda o actualiza un producto en la base de datos. Verifica si el usuario es un administrador, obtiene los datos del formulario, crea una instancia de la clase Producto, establece los valores y guarda o actualiza el producto en la base de datos. Maneja también la subida de imágenes y redirige a la página de gestión de productos.

### 5. eliminar

Elimina un producto de la base de datos. Verifica si el usuario es un administrador y si se ha proporcionado un ID válido, elimina el producto de la base de datos y establece una sesión con un mensaje de éxito o error según corresponda. Luego redirige a la página de gestión de productos.

### 6. editar

Permite a los administradores editar un producto existente. Verifica si el usuario es un administrador y si se ha proporcionado un ID válido, obtiene los detalles del producto a editar y carga la vista crear.php (usada tanto para crear como para editar) con los datos del producto.

### 7. ver

Muestra los detalles de un producto específico. Verifica si se ha proporcionado un ID válido, obtiene los detalles del producto y carga la vista ver.php para mostrar esta información.

## usuarioController :

### 1. index

Muestra un mensaje indicando que se está en el controlador Usuarios y la acción index.

### 2. registro

Carga la vista registro.php para permitir a los usuarios registrarse.

### 3. save

Procesa el formulario de registro de un nuevo usuario. Verifica si se han recibido datos por POST, realiza validaciones de los campos del formulario, crea una instancia de la clase Usuario, guarda el usuario en la base de datos y redirige al formulario de registro con mensajes de éxito o error según corresponda.

### 4. login

Procesa el formulario de inicio de sesión. Verifica si se han recibido datos por POST, intenta autenticar al usuario, establece la sesión y una cookie si la autenticación es exitosa, y redirige a la página principal o muestra un mensaje de error si falla.

### 5. logout

Cierra la sesión del usuario actual. Destruye las variables de sesión relacionadas con el usuario y redirige a la página principal.

### 6. modificar

Permite a los usuarios autenticados modificar sus datos personales. Verifica si el usuario está autenticado, procesa el formulario de modificación, realiza validaciones, actualiza los datos del usuario en la base de datos y redirige al perfil del usuario con mensajes de éxito o error según corresponda.

### 7. gestionarUsuarios

Muestra la lista de todos los usuarios para su gestión. Verifica si el usuario es administrador, obtiene todos los usuarios usando el método getAll() y carga la vista gestion.php para listar los usuarios.

### 8. editarUsuario

Permite a los administradores editar los datos de un usuario específico. Verifica si el usuario es administrador y si se ha proporcionado un ID válido, obtiene los datos del usuario a editar y carga la vista editar.php para permitir la edición.

### 9. actualizarUsuario

Actualiza los datos de un usuario en la base de datos. Verifica si el usuario es administrador, procesa el formulario de actualización, verifica si los datos son válidos, actualiza los datos del usuario en la base de datos y redirige a la gestión de usuarios con mensajes de éxito o error según corresponda.

### 10. eliminarUsuario

Elimina un usuario de la base de datos. Verifica si el usuario es administrador y si se ha proporcionado un ID válido, elimina el usuario de la base de datos y redirige a la gestión de usuarios con mensajes de éxito o error según corresponda.

## Core

### config

Cargo las variables de entorno desde un archivo .env y utilizo esas variables para definir constantes que serán utilizadas para configurar la conexión a una base de datos

```
<?php

use Dotenv\Dotenv;

// Cargar variables del archivo .env
$dotenv = Dotenv::createImmutable(paths: __DIR__ . '/../..');
$dotenv->load();

// Definir constantes para la conexión a la base de datos
define(constant_name: 'DB_HOST', value: $_ENV['DB_HOST']);
define(constant_name: 'DB_NAME', value: $_ENV['DB_NAME']);
define(constant_name: 'DB_USER', value: $_ENV['DB_USER']);
define(constant_name: 'DB_PASSWORD', value: $_ENV['DB_PASSWORD']);

?>
```

## dataBase

### dataBase

Está toda la base de datos junto con valores insertados

## helpers

### email Helpers

Aqui tengo la función para permitir enviar un mensaje de texto al hacer un pedido utilizando `PHPMailer`

```

1 reference
function enviarCorreoPedido($emailDestino, $nombreCliente, $pedido): bool|string
{
    $mail = new PHPMailer(true);
    try {
        // Configurar SMTP
        $mail->isSMTP();
        $mail->Host = 'smtp.gmail.com'; // Servidor SMTP (Gmail, Outlook, etc.)
        $mail->SMTPAuth = true;
        $mail->Username = 'enviarcorreo71@gmail.com'; // Tu correo
        $mail->Password = 'tgvy djpc ayvq gvbc'; // Contraseña o App Password si usas Gmail
        $mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;
        $mail->Port = 587; // Puerto de salida

        // Configuración del correo
        $mail->setFrom('enviarcorreo71@gmail.com', 'Tienda Online');
        $mail->addAddress($emailDestino, $nombreCliente); // Destinatario
        $mail->Subject = 'Confirmación de Pedido - Tienda Online';

        // Construir el cuerpo del email
        $mensaje = "<h2>Hola $nombreCliente,</h2>";
        $mensaje .= "<p>Gracias por tu compra. Aquí están los detalles de tu pedido:</p>";
        $mensaje .= "<ul>";
        foreach ($pedido['productos'] as $producto) {
            $mensaje .= "<li>{$producto['nombre']} - Cantidad: {$producto['cantidad']} - Precio: {$producto['precio']}€</li>";
        }
        $mensaje .= "</ul>";
        $mensaje .= "<p>Total: {$pedido['total']}€</p>";
        $mensaje .= "<p>Esperamos verte pronto.</p>";

        // Enviar email en formato HTML
        $mail->isHTML(true);
        $mail->Body = $mensaje;

        // Enviar el correo
        $mail->send();
        return true;
    } catch (Exception $e) {
        return "Error al enviar el correo: {$mail->ErrorInfo}";
    }
}

```

## utils

### 1. deleteSession(\$name)

Propósito: Elimina una variable de sesión específica.

Funcionamiento: Verifica si existe la variable de sesión con el nombre proporcionado, la establece como null y luego la elimina usando unset(). Finalmente, devuelve el nombre de la variable.

### 2. isAdmin()

Propósito: Verifica si el usuario actual es un administrador.

Funcionamiento: Comprueba si existe la variable de sesión admin. Si no existe, redirige al usuario a la página principal (base\_url). Si existe, devuelve true.

### 3. showCategorias()

Propósito: Muestra todas las categorías disponibles.

Funcionamiento: Requiere el archivo `categorias.php`, crea una instancia de la clase `Categoria`, obtiene todas las categorías usando el método `getAll()` y devuelve la lista de categorías.

#### 4. `statsCarrito()`

Propósito: Calcula estadísticas básicas del carrito de compras (número de productos y total).

Funcionamiento: Inicializa un array `$stats` con dos claves: `count` (para contar los productos) y `total` (para calcular el costo total). Luego verifica si existe el carrito en la sesión, cuenta los productos y calcula el total multiplicando el precio de cada producto por su cantidad.

Finalmente, devuelve el array `$stats`.

#### 5. `isIdentity()`

Propósito: Verifica si el usuario actual está autenticado.

Funcionamiento: Comprueba si existe la variable de sesión `identity`. Si no existe, redirige al usuario a la página principal (`base_url`). Si existe, devuelve `true`.

#### 6. `showStatus($status)`

Propósito: Devuelve una representación legible del estado de un pedido.

Funcionamiento: Recibe un parámetro `$status` que representa el estado del pedido. Según el valor de `$status`, asigna una cadena correspondiente (por ejemplo, 'Pendiente', 'En preparación', etc.) y la devuelve.

## lib

## Database

#### 1. `__construct`

Carga las variables de entorno desde el archivo `.env`, intenta crear una instancia de PDO usando estas variables para conectarse a la base de datos y maneja cualquier error de conexión.

```

require_once __DIR__ . '/../vendor/autoload.php'; // Ajusta la ruta si es necesario

use Dotenv\Dotenv;

9 references | 0 implementations
class Database
{
    3 references
    private static $instance = null;
    2 references
    private $pdo;

    // Constructor privado para evitar instanciación directa
    2 references | 0 overrides
    function __construct()
    {
        // Cargar el archivo .env
        $dotenv = Dotenv::createImmutable(paths: __DIR__ . '/../'); // Ruta a la raíz del proyecto
        $dotenv->load(); // Carga las variables de entorno

        try {
            // Usar las variables de entorno cargadas para la conexión
            $dsn = 'mysql:host=' . $_ENV['DB_HOST'] . ';dbname=' . $_ENV['DB_NAME'] . ';charset=utf8';
            $options = [
                PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
                PDO::ATTR_EMULATE_PREPARES => false,
            ];

            // Crear la instancia de PDO con los datos de .env
            $this->pdo = new PDO($dsn, $_ENV['DB_USER'], $_ENV['DB_PASSWORD'], $options);
        } catch (PDOException $e) {
            // Manejar el error si no se puede conectar a la base de datos
            die("Error al conectar a la base de datos: " . $e->getMessage());
        }
    }
}

```

## 2. getInstance

Implementa el patrón Singleton, asegurando que solo exista una instancia de la clase Database. Si la instancia no existe, la crea y la devuelve; de lo contrario, devuelve la instancia existente.

## 3. getConnection

Proporciona acceso a la conexión PDO creada en el constructor.

## 4. \_\_clone

Evita que la instancia de la clase Database sea clonada, manteniendo así la unicidad de la instancia según el patrón Singleton.

# models

## categorias

### \_\_construct

Inicializa la conexión a la base de datos utilizando el método getInstance() de la clase Database.

## 2. getId

Devuelve el valor del ID de la categoría.

### 3. getNombre

Devuelve el nombre de la categoría.

### 4. setId

Establece el valor del ID de la categoría.

### 5. setNombre

Establece el nombre de la categoría.

### 6. getAll

Obtiene todas las categorías de la base de datos y las devuelve como un array asociativo. Realiza una consulta SQL para seleccionar todas las categorías ordenadas por ID en orden descendente.

### 7. save

Guarda una nueva categoría en la base de datos. Prepara una consulta SQL para insertar una nueva categoría con el nombre proporcionado y ejecuta la consulta.

### 8. getOne

Obtiene una categoría específica de la base de datos basada en su ID y la devuelve como un objeto. Realiza una consulta SQL para seleccionar la categoría con el ID especificado.

### 9. update

Actualiza el nombre de una categoría existente en la base de datos. Prepara una consulta SQL para actualizar el nombre de la categoría con el ID especificado y ejecuta la consulta.

### 10. delete

Elimina una categoría y todos los productos asociados a ella de la base de datos. Primero elimina los productos que pertenecen a la categoría y luego elimina la categoría misma. Realiza dos consultas SQL: una para eliminar los productos y otra para eliminar la categoría.

## pedido

### 1. \_\_construct

Inicializa la conexión a la base de datos utilizando el método getInstance() de la clase Database.

### 2. getId

Devuelve el valor del ID del pedido.

3. getUsuario\_id

Devuelve el ID del usuario asociado al pedido.

4. getProvincia

Devuelve la provincia asociada al pedido.

5. getLocalidad

Devuelve la localidad asociada al pedido.

6. getDireccion

Devuelve la dirección asociada al pedido.

7. getCoste

Devuelve el coste total del pedido.

8. getEstado

Devuelve el estado actual del pedido.

9. getFecha

Devuelve la fecha en que se realizó el pedido.

10. getHora

Devuelve la hora en que se realizó el pedido.

11. setId

Establece el valor del ID del pedido.

12. setUsuario\_id

Establece el ID del usuario asociado al pedido.

13. setProvincia

Establece la provincia asociada al pedido.

14. setLocalidad

Establece la localidad asociada al pedido.

15. setDireccion

Establece la dirección asociada al pedido.

16. setCoste

Establece el coste total del pedido.

17. setEstado

Establece el estado actual del pedido.

18. setFecha

Establece la fecha en que se realizó el pedido.



#### 19. setHora

Establece la hora en que se realizó el pedido.

#### 20. getAll

Obtiene todos los pedidos de la base de datos y los devuelve como un array de objetos. Realiza una consulta SQL para seleccionar todos los pedidos ordenados por ID en orden descendente.

#### 21. getOne

Obtiene un pedido específico de la base de datos basado en su ID y lo devuelve como un objeto. Realiza una consulta SQL para seleccionar el pedido con el ID especificado.

#### 22. getOneByUser

Obtiene el último pedido realizado por un usuario específico y lo devuelve como un objeto. Verifica que el ID del usuario no sea nulo o inválido antes de realizar la consulta SQL.

#### 23. getAllByUser

Obtiene todos los pedidos realizados por un usuario específico y los devuelve como un array de objetos. Realiza una consulta SQL para seleccionar todos los pedidos del usuario ordenados por ID en orden descendente.

#### 24. getProductosByPedido

Obtiene los productos asociados a un pedido específico y los devuelve como un array de objetos. Realiza una consulta SQL que une las tablas productos y lineas\_pedidos para obtener los detalles de los productos del pedido especificado.

#### 25. save

Guarda un nuevo pedido en la base de datos. Prepara una consulta SQL para insertar un nuevo pedido con los datos proporcionados y ejecuta la consulta.

#### 26. save\_linea

Guarda las líneas del pedido (productos) en la base de datos. Obtiene el ID del último pedido insertado y luego inserta cada producto en la tabla lineas\_pedidos con sus respectivas unidades.

#### 27. edit

Edita el estado de un pedido existente en la base de datos. Prepara una consulta SQL para actualizar el estado del pedido con el ID especificado y ejecuta la consulta.

### producto

#### 1. \_\_construct

Inicializa la conexión a la base de datos utilizando el método getInstance() de la clase Database.

#### 2. getId

Devuelve el valor del ID del producto.

3. getCategory\_id

Devuelve el ID de la categoría asociada al producto.

4. getNombre

Devuelve el nombre del producto.

5. getDescripcion

Devuelve la descripción del producto.

6. getPrecio

Devuelve el precio del producto.

7. getStock

Devuelve el stock del producto.

8. getOferta

Devuelve si el producto está en oferta.

9. getFecha

Devuelve la fecha de creación del producto.

10. getImagen

Devuelve la imagen del producto.

11. setId

Establece el valor del ID del producto.

12. setCategoria\_id

Establece el ID de la categoría asociada al producto.

13. setNombre

Establece el nombre del producto.

14. setDescripcion

Establece la descripción del producto.

15. setPrecio

Establece el precio del producto.

16. setStock

Establece el stock del producto.

17. setOferta

Establece si el producto está en oferta.

18. setFecha

Establece la fecha de creación del producto.

#### 19. setImagen

Establece la imagen del producto.

#### 20. getAll

Obtiene todos los productos de la base de datos y los devuelve como un array de objetos. Realiza una consulta SQL para seleccionar todos los productos ordenados por ID en orden descendente.

#### 21. getOne

Obtiene un producto específico de la base de datos basado en su ID y lo devuelve como un objeto. Realiza una consulta SQL para seleccionar el producto con el ID especificado.

#### 22. save

Guarda un nuevo producto en la base de datos. Asigna la fecha actual si no se proporcionó una, prepara una consulta SQL para insertar un nuevo producto con los datos proporcionados y ejecuta la consulta.

#### 23. edit

Edita un producto existente en la base de datos. Prepara una consulta SQL para actualizar los datos del producto con el ID especificado, incluyendo la opción de actualizar la imagen si se proporciona, y ejecuta la consulta.

#### 24. delete

Elimina un producto de la base de datos. Prepara una consulta SQL para eliminar el producto con el ID especificado y ejecuta la consulta.

#### 25. getRandom(\$limit)

Obtiene una cantidad específica de productos aleatorios de la base de datos y los devuelve como un array de objetos. Realiza una consulta SQL que selecciona productos en orden aleatorio limitando el resultado según el valor proporcionado.

#### 26. getAllCategory

Obtiene todos los productos de una categoría específica de la base de datos y los devuelve como un array de objetos. Realiza una consulta SQL que une las tablas productos y categorías para obtener los detalles de los productos de la categoría especificada.

### usuario

#### 1. \_\_construct

Inicializa la conexión a la base de datos utilizando el método getInstance() de la clase Database.

2. getId

Devuelve el valor del ID del usuario.

3. setId

Establece el valor del ID del usuario.

4. getNombre

Devuelve el nombre del usuario.

5. setNombre

Establece el nombre del usuario.

6. getApellidos

Devuelve los apellidos del usuario.

7. setApellidos

Establece los apellidos del usuario.

8. getEmail

Devuelve el email del usuario.

9. setEmail

Establece el email del usuario.

10. getPassword

Devuelve la contraseña del usuario hasheada usando password\_hash.

11. setPassword

Establece la contraseña del usuario.

12. getRol

Devuelve el rol del usuario.

13. setRol

Establece el rol del usuario.

16. save

Guarda un nuevo usuario en la base de datos. Aplica un hash a la contraseña antes de insertarla y prepara una consulta SQL para insertar un nuevo usuario con los datos proporcionados, ejecutando la consulta.

17. login

Autentica a un usuario basado en su email y contraseña. Busca un usuario con el email proporcionado, verifica la contraseña usando password\_verify y devuelve el objeto del usuario si la autenticación es exitosa.

#### 18. update

Actualiza los datos de un usuario existente en la base de datos. Prepara una consulta SQL para actualizar los datos del usuario con el ID especificado y ejecuta la consulta.

#### 19. getAll

Obtiene todos los usuarios de la base de datos y los devuelve como un array de objetos. Realiza una consulta SQL para seleccionar todos los usuarios.

#### 20. getOne

Obtiene un usuario específico de la base de datos basado en su ID y lo devuelve como un objeto. Realiza una consulta SQL para seleccionar el usuario con el ID especificado.

#### 21. delete

Elimina un usuario y todos los pedidos y líneas de pedido asociados a él de la base de datos. Primero elimina las líneas de pedido relacionadas, luego elimina los pedidos relacionados con el usuario y finalmente elimina el usuario mismo. Realiza tres consultas SQL: una para eliminar las líneas de pedido, otra para eliminar los pedidos y otra para eliminar el usuario.

### public

- css: Tengo los estilos de la pagina
- img: Tengo el logotipo de la pagina

### uploads

Aquí es donde se descargan las imágenes que se suben a la página

### vendor

Es la carpeta que se crea cuando descargas el composer o el phpmailer

# Views

## Carrito:

- index: muestra el contenido del carrito de la compra en una página web. Si el carrito no está vacío, se crea una tabla que lista cada producto con su imagen, nombre, precio, unidades y opciones para aumentar o disminuir la cantidad o eliminar el producto. También proporciona un botón para vaciar completamente el carrito y muestra el precio total de los productos en el carrito. Finalmente, ofrece un enlace para hacer el pedido. Si el carrito está vacío, simplemente muestra un mensaje indicando que el carrito está vacío y sugiere añadir algún producto.

## categorías:

- cat: muestra una página para gestionar las categorías en una aplicación web. Si hay categorías disponibles, lista cada una con su ID y nombre, y proporciona botones para editar o eliminar la categoría. Si no hay categorías, muestra un mensaje indicando que no hay categorías disponibles. Además, incluye un botón para agregar una nueva categoría.
- crear: muestra una pagina para crear una nueva categoría
- editar : muestra una pagina para editar una categoría en específico
- ver: muestra una pagina para ver una categoría en específico

## layouts :

- footer : es el footer el cual luego se inserta con php
- header : es el header el cual luego se inserta con php
- sidebar: es el sidebar el cual luego se inserta con php

## pedido :

- confirmado : muestra una página de confirmación de pedido en una aplicación web. Si el pedido se ha completado con éxito, muestra un mensaje de confirmación junto con los detalles del pedido, incluyendo el número de pedido, el total a pagar y una tabla con los productos comprados, sus imágenes, nombres, precios y unidades. Si el pedido no se pudo procesar, muestra un mensaje indicando que el pedido no ha podido procesarse.
- detalle : muestra una pagina con los detalles de los pedidos
- hacer: muestra una pagina para que introduzca los datos del envio del pedido
- misPedidos : muestra los pedidos que tiene ese usuario

## productos :

- crear: Muestra una pagina para crear un producto
- edit: Muestra una pagina para editar un producto
- gestion: Muestra una pagina que solo pueden ver los administradores que es para gestionar todos los productos
- ver : muestra una pagina para ver un producto en concreto

## usuario :

- editar : Muestra una pagina para editar los datos de los usuarios
- gestion : Muestra todos los usuarios y desde aqui puedes editarlos o eliminarlos
- modificar : Muestra una pagina para modificar los datos de tu cuenta
- registro : Muestra una pagina para que los usuarios se registren

## .env

Aqui tengo las variables para la base de datos

## .gitignore

Todos los archivos que quiero q no se suban a github

## .htaccess

Este código de configuración de Apache activa el módulo `mod_rewrite`, establece una página personalizada para errores 404, limita el acceso a directorios y archivos que no existen, y define reglas para reescribir URLs en el formato `index.php?controller=$1&action=$2`, permitiendo una estructura de URL más limpia y amigable.

## autoload.php

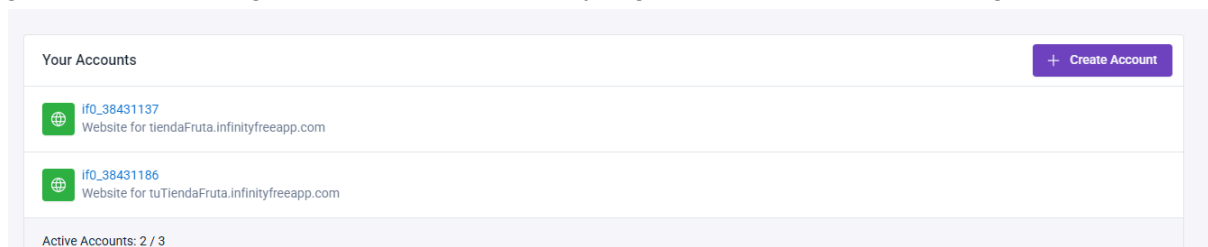
Incluye automáticamente archivos PHP desde el directorio `controllers/` basándose en el nombre de la clase proporcionado. Esto permite que las clases de controladores se carguen dinámicamente cuando son necesarias, mejorando la organización y mantenimiento del código.

## index.php

Es el punto de entrada principal de una aplicación MVC, gestionando las solicitudes del usuario. Inicia la sesión, carga archivos necesarios y crea una instancia de la base de datos. Luego, determina el controlador y acción a ejecutar basándose en los parámetros de la URL o usar valores predeterminados si no se proporcionan. Si el controlador y la acción existen, crea una instancia del controlador y llama al método correspondiente; de lo contrario, muestra un error. Finalmente, incluye las vistas de encabezado, barra lateral y pie de página para completar la respuesta.

## Despliegue de la aplicación :

Para desplegar la aplicación he usado infinity free el cual permite subir una aplicación gratuitamente. Luego de crearte una cuenta y registrarte le tienes q dar al siguiente boton



Luego de crearlo para subir los archivos a este servidor yo he usado filezilla



ya q arriba simplemente poniendo estos datos

MYSQL USERNAME

ifo\_38431186

MYSQL PASSWORD

\*\*\*\*\* Show/Hide

MYSQL DATABASE NAME

ifo\_38431186\_XXX  
(see below)

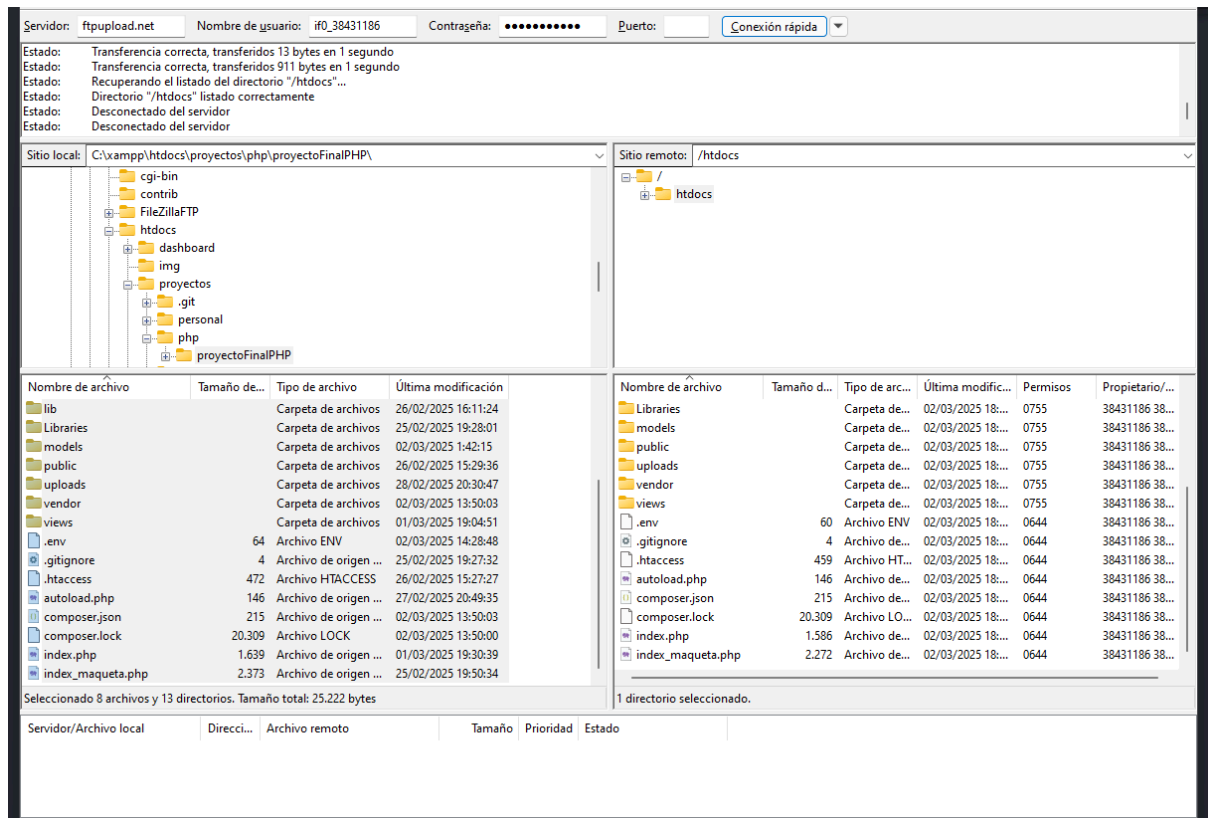
MYSQL HOSTNAME

sql101.infinityfree.com

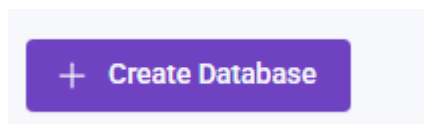
MYSQL PORT (OPTIONAL)

3306

Se pasan facilmente



Para la base de datos simplemente le das a este boton



y pones la base de datos importada de tu phpMyAdmin y ya simplemente buscando

[tuTiendaFruta.infinityfreeapp.com](http://tuTiendaFruta.infinityfreeapp.com)

aparecerá la página web desplegada. Para eso los archivos de la base de datos de la ruta amigable las he tenido q cambiar para q funcionara

