PROYECTO FINAL 24-25
2°DAW

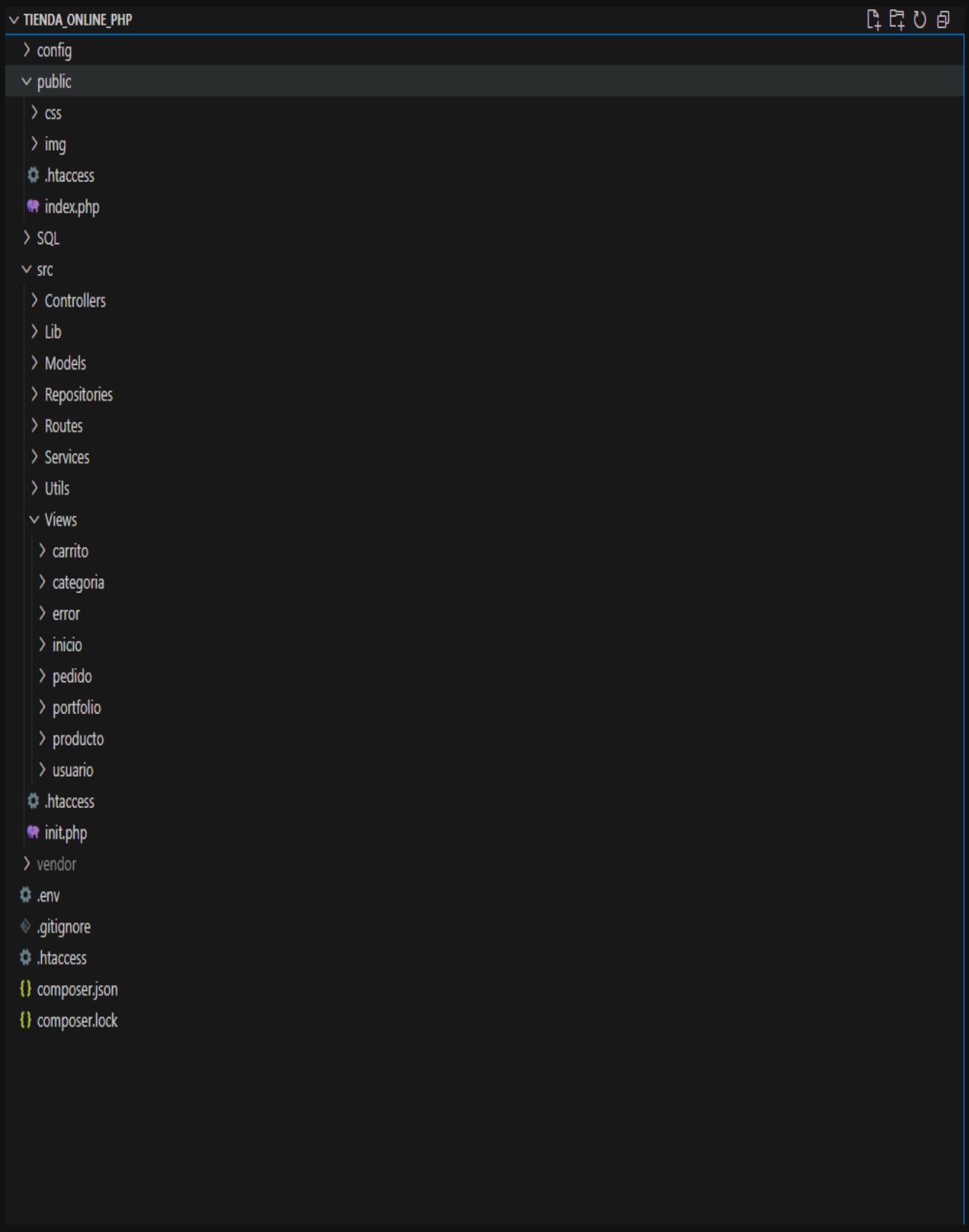
Tienda de Informática

PHP

Jairo Alejandro Gálvez Rodríguez

Índice

Estructura del Proyecto



Breve explicación de cada uno de los ficheros

-Carpeta config:

config.php → El archivo config.php sirve como una especie de punto de encuentro central para configuraciones importantes en mi proyecto de tienda online. En él, he definido una constante llamada BASE_URL. Esta constante es muy útil porque guarda la dirección base del sitio web.

-Carpeta public:

Contiene las carpetas css e img → La carpeta css contiene distintos ficheros .css, que funcionan para darle un diseño más elegante a las distintas vistas de la propia página web.
Al igual que la carpeta img que contiene varías imágenes que he estado usando para la tienda de informática, ya sean imágenes de los propios productos o de los fondos de cada página, etc..

.htaccess → Este archivo .htaccess está configurando algunas reglas importantes para el servidor web Apache que ayudan a manejar cómo se procesan las URLs de mi proyecto: El uso de este archivo facilita el manejo de URLs amigables.

index.php → Actúa como un punto de entrada que delega la inicialización y configuración esencial del proyecto a otro archivo (init.php), así mantengo el index.php limpio y organizado.

-Carpeta SQL:

tienda_online.php → Contiene una serie de instrucciones SQL fundamentales para configurar la base de datos de la tienda. Incluye tanto la Creación y Configuración de la Base de Datos, la Creación y Modificación de Tablas e Inserción de los Datos Iniciales.
-En resumen sirve para establecer la estructura de la base de datos de la tienda online, configurando todo lo necesario para almacenar y gestionar los datos de usuarios, productos, pedidos, etc.

-Carpeta src:

-Dentro tiene las siguientes carpetas y archivos..

-Carpeta Controllers:

CarritoController.php → Este archivo sirve para el manejo de la funcionalidad del carrito de compras de la tienda online. Esta clase "`CarritoController`", interactúa con modelos, servicios, y utilidades para gestionar las acciones relacionadas con el carrito.
-Es decir, esta clase contiene la lógica para gestionar un carrito de compras, incluyendo añadir, eliminar, vaciar, y ajustar cantidades de productos, además de calcular el total. Desde mi punto de vista facilita bastante la interacción del usuario con el carrito de compras.

CategoriaController.php → Lo he utilizado para gestionar todas las operaciones relacionadas con las categorías en la aplicación. Utiliza varios servicios y modelos para manipular datos de categorías.

-osea contiene la creación, visualización, edición, y eliminación de categorías, además de integrar estas operaciones con la visualización de productos asociados a cada categoría.

ErrorController.php → Lo uso para ser llamado cuando el enrutamiento de la aplicación detecte una solicitud a una ruta no definida.

-Por ejemplo, si alguien intenta acceder a una página que no ha sido definida en las rutas del sistema, el sistema de enrutamiento redirige automáticamente al método `error404()` de este controlador para informar de manera amigable que la página solicitada no se encuentra.

InicioController.php → Lo uso para la gestión de la página principal de la aplicación, proporcionando a los usuarios una vista inicial atractiva y dinámica con productos que pueden cambiar con cada carga de página. Utilizo el ProductoController para obtener los datos, para así mantener el principio de separación de preocupaciones dentro del proyecto, donde InicioController se concentra en la lógica de la página de inicio, mientras que ProductoController se encarga de la lógica específica del producto.

PedidoController.php → Aparte de manejar la visualización y gestión de pedidos, también integra funciones como la validación de entrada y comunicación por correo electrónico.

-Basicamente, es un controlador que gestiona todas las operaciones relacionadas con los pedidos. Ya sea mostrar todos los pedidos si el usuario es admin, mostrar los pedidos de cada usuario logueado, validar los datos de envío de un pedido antes de su creación, también permite al administrador crear, editar y eliminar pedidos, además de enviar un email al cliente cuando el admin da el pedido como confirmado, etc..

ProductoController.php → Este controlador maneja diversas funcionalidades desde la recuperación de productos hasta la creación y edición de estos. También permite ver los detalles de un producto en específico. De forma breve, este controlador lo he usado para la gestión de productos.

UsuarioController.php → Funciona para gestionar todas las actividades relacionadas con los usuarios de la tienda online. ofrece una interfaz para el registro, inicio de sesión, visualización, edición y eliminación de usuarios, asegurando que los datos se manejen de forma segura.

-Carpeta Lib:

BaseDatos.php → Permite la interacción con la base de datos para otras partes de la aplicación, asegurando que la conexión y las operaciones de base de datos se manejen de manera más eficiente y segura posible.
-En resumen, es una clase que encapsula la funcionalidad relacionada con la gestión de conexiones y operaciones en una base de datos MySQL utilizando PDO.

Pages.php → El archivo Pages.php es una clase que proporciona la funcionalidad para manejar la renderización de páginas en la aplicación. Y su objetivo es facilitar la inclusión de vistas y manejar la integración de las mismas con plantillas comunes de cabecera y pie de página.

Router.php → Es una implementación esencial de un enrutador para una aplicación web PHP (como es el caso de esta tienda online), que se encarga de dirigir las solicitudes HTTP a los controladores correspondientes basados en la URL. Este enrutador maneja el mapeo de rutas a funciones específicas y la ejecución de estas funciones.

-Carpeta Models:

Categoria.php → Representa un modelo para las categorías de productos, con sus getId() y setId($id): Que son los métodos para obtener y establecer el valor de $id. Estos sirven para manejar la identificación única de cada categoría.
-getNombre() y setNombre($nombre): Métodos para obtener y establecer el valor de $nombre. Los uso para recuperar y actualizar el nombre de la categoría, lpara así hacer operaciones de creación y actualización en la base de datos.

Pedido.php → Modela la estructura de datos y las operaciones de un pedido dentro de la aplicación. Esta clase encapsula todas las propiedades necesarias de un pedido y proporciona métodos para acceder y modificar estas propiedades.
-Las propiedades que uso en la clase Pedido son:

$id: Identificador único del pedido.

$usuario_id: Identificador del usuario que realizó el pedido.

$provincia, $localidad, $direccion: Detalles de la dirección de envío del pedido.

$coste: Coste total del pedido.

$estado: Estado actual del pedido (p.ej., pendiente, enviado, entregado).

$fecha, $hora: Fecha y hora en que se realizó el pedido.

$db: Instancia de BaseDatos para realizar operaciones relacionadas con la base de datos.

Producto.php → Es una representación de los productos de tu tienda en línea. Este modelo sirve para la gestión de los productos en la base de datos, desde la creación y actualización hasta la recuperación de los detalles de los productos.
-En conclusión, facilita la manipulación y gestión de la información del producto.

Usuario.php → Encapsula todo lo relacionado con la gestión de los usuarios en la aplicación, incluyendo la creación y el manejo de información de usuarios dentro de la base de datos.
-Cada propiedad de la clase tiene su correspondiente método getter y setter .Aparte tiene el Método Estático fromArray, que este método facilita la creación de instancias de Usuario a partir de un array asociativo y también tiene el método Desconecta que sirve para cerrar la conexión a la base de datos cuando ya no es necesaria.

-Carpeta Repositories:

CategoriaRepository.php → Es diseñada para manejar todas las operaciones relacionadas con las categorías en la base de datos. Este repositorio actúa como una capa de abstracción que se encarga de interactuar directamente con la base de datos para ejecutar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre la entidad Categoria.

PedidoRepository.php → Funciona para manejar todas las operaciones relacionadas con los pedidos en la base de datos. Utiliza la clase BaseDatos para la conexión y operaciones de base de datos, facilitando la ejecución de consultas SQL y gestionando las transacciones de manera segura.

-Tiene varios métodos como el getAll():

- Recupera todos los pedidos de la base de datos y cierra la conexión.

-getById($id):

- Recupera un pedido específico por su ID.

-getByUsuario($usuarioId):

- Obtiene todos los pedidos asociados a un ID de usuario específico, también usando una consulta parametrizada.

-borrarCategoria($id):

- Elimina las líneas de pedido asociadas a un pedido y luego el pedido en sí.

-editar($id, $coste, $usuario_id):

- Actualiza los detalles específicos de un pedido, como el coste y el ID del usuario asociado.

-getProductosPedido($pedidoId):

- Retorna todos los productos asociados con un pedido específico, utilizando consultas SQL para cruzar información entre tablas de pedidos y líneas de pedidos.

-getCantidadProducto($pedidoId, $productoId):

- Consulta la cantidad de un producto específico en un pedido, útil para operaciones de gestión de inventario y verificación de pedidos.

-guardarCategoria(...):

- Realiza la inserción de un pedido y sus detalles de línea en la base de datos utilizando transacciones para asegurar la integridad de los datos.

ProductoRepository.php -→ Realiza todas las operaciones relacionadas con los productos en la base de datos de la tienda. Para cualquier duda resumo los métodos que hace esta clase:

recuperarTodasCategorias():

- Recupera todos los productos de la base de datos.
- Tendría que haberle puesto recuperarTodosProductos para no ser muy lioso pero bueno me lié mientras ponía los nombres de los métodos en este archivo.

- getRandom():
  - Recupera cinco productos al azar de la base de datos. Lo uso para mostrar algunos productos en la página principal de la tienda.
- getByCategoria($categoriaId):
  - Obtiene todos los productos que pertenecen a una categoría específica, así filtrar productos en la interfaz de usuario según la categoría seleccionada por el usuario.
- categoriaPorId($id):

    - Recupera un producto específico por su ID.
- guardarCategoria(...):
    - Inserta un nuevo producto en la base de datos.
- borrarCategoria($productId):
    - Elimina un producto de la base de datos por su ID.
- actualizarCategoria(...):
    - Actualiza un producto existente en la base de datos, para cuando se necesiten hacer cambios en la información del producto.
- executeQuery($sql, $params = []):
    - Ejecuta consultas de selección y devuelve todos los registros. Este método prepara y ejecuta la consulta SQL, gestionando los parámetros de manera segura para prevenir inyecciones SQL.
- executeUpdate($sql, $params = []):
    - Ejecuta actualizaciones o inserciones y devuelve un booleano indicando el éxito de la operación.

UsuarioRepository.php → Sirve para realizar las operaciones de base de datos relacionadas con usuarios. Y bueno por explicar lo que hacen sus métodos pues...

-create($usuario):

- Inserta un nuevo usuario en la base de datos. Utilizo consultas para evitar la inyección SQL y maneja excepciones para capturar errores durante la inserción.
- -verTodos():
- Recupera todos los usuarios de la base de datos. Utilizo un método para ejecutar una consulta y cerrar la conexión inmediatamente después de recuperar los datos.
- -login($usuario):
- Verifica las credenciales de un usuario para el inicio de sesión, comparando el correo electrónico y la contraseña proporcionados con los almacenados en la base de datos. Utilizo password_verify para comparar la contraseña hash.
- -buscaMail($email):
- Busca un usuario por su email. Retorna el usuario si existe, o null si no existe, manejando también los posibles errores con excepciones.
- -usuarioPorId($id):
- Recupera un usuario específico por su ID. Este método es bueno para operaciones que requieren detalles del usuario a partir de su ID.
- -actualizarUsuario($usuario):

- Actualiza los detalles de un usuario existente en la base de datos. Manejo todos los campos que podrían cambiarse, incluyendo nombre, apellidos, email y rol.
- -borrarUsuario($id):
- Elimina un usuario por su ID.

-Carpeta Routes:

Routes.php → Este archivo utiliza la clase Router del namespace Lib para registrar y manejar las rutas específicas que conectan URLs con los métodos correspondientes en varios controladores. Cada ruta define cómo una solicitud HTTP es procesada y a qué controlador y método debe ser dirigida. Por lo cual, es esencial para la arquitectura de la tienda online porque permite realizar cambios fácilmente gestionables en la estructura del enrutamiento.

-Carpeta Services:

CategoriaService.php → sirve como capa intermedia entre los controladores de la aplicación y el acceso a datos específicos para las categorías, que es gestionado por CategoriaRepository. Este servicio lo uso para lo que esta relacionado con las categorías, permitiendo que los controladores interactúen con los datos de categorías de una manera más organizada.

-Sus métodos son: -recuperarTodasCategorias():

- **Función**: Recupera todas las categorías existentes desde la base de datos.
- **Retorno**: Devuelve un array con la lista de todas las categorías, utilizando el método correspondiente del repositorio.
- **categoriaPorId($id)**:
- **Función**: Busca y retorna los detalles de una categoría específica basada en su ID.
- **Retorno**: Devuelve un array con los datos de la categoría solicitada, si existe; de lo contrario, puede retornar null.
- **guardarCategoria($nombre)**:
- **Función**: Guarda una nueva categoría en la base de datos con el nombre proporcionado.
- **Retorno**: Devuelve true si la operación es exitosa, false si ocurre un fallo durante la inserción.
- **borrarCategoria($id)**:
- **Función**: Elimina una categoría existente identificada por su ID.
- **Retorno**: Devuelve true si la categoría es eliminada exitosamente, false de lo contrario.
- **actualizarCategoria($id, $nombre)**:
- **Función**: Actualiza el nombre de una categoría existente.

PedidoService.php → Bueno pues este servicio se encarga de facilitar las operaciones relacionadas con los pedidos con métodos como:

-recuperarTodosPedidos():

- Retorna todos los pedidos registrados en el sistema. Utilizo el método getAll() del repositorio.
- obtenerPedidoPorId($id):
- Recupera los detalles de un pedido específico por su ID utilizando getById($id) del repositorio.
- obtenerPedidosPorUsuario($usuarioId):
- Obtiene todos los pedidos asociados a un usuario en particular. Algo bastante importante.
- eliminarPedido($id):
- Elimina un pedido por su ID.
- actualizarPedido(...):
- Actualiza información específica de un pedido existente, como el coste y el ID del usuario, lo que permite cambios administrativos o correcciones en los detalles del pedido.
- guardarPedido(...):
- Guarda un nuevo pedido en la base de datos, gestionando detalles como usuario, dirección, coste, y otros datos del pedido, junto con los detalles de los productos pedidos (carrito).

- getProductosPedido($pedidoId):
- Recupera información detallada sobre los productos incluidos en un pedido específico, importante para la revisión y gestión de pedidos.
- getUsuarioPedido($pedidoId):
- Obtiene el ID de usuario asociado a un pedido específico.
- getTotalCarrito($carrito):
- Calcula el total del carrito de compras, lo que es esencial para la finalización del pedido y la revisión del total antes del pago.
- getCantidadProducto($pedidoId, $productoId):
- Obtiene la cantidad específica de un producto en un pedido, útil para ajustes de inventario y confirmación de pedidos.
- confirmarPedido($pedidoId):
- Marca un pedido como confirmado.

ProductoService.php → Proporciona una capa de servicios para manejar operaciones relacionadas con productos a través de un ProductoRepository. Permitiendo recuperar todos los productos disponibles, retornar todos los productos que pertenecen a una categoría en específica, recuperar un producto en específico por su id, guardar un nuevo producto en la base de datos con todos los detalles proporcionados, como nombre, descripción, precio, etc. Al igual que

actualizar los detalles de un producto existente o eliminar un producto por su id.

UsuarioService.php → Es un componente que mejora la organización y mantenibilidad del manejo de usuarios dentro de la aplicación. Teniendo métodos como:

-`create($usuario)`:

- Crea un nuevo usuario utilizando los datos proporcionados. Retorna true si el usuario se crea exitosamente.
- verTodos():
- Recupera todos los usuarios registrados en el sistema.
- login($usuario):
- Maneja el proceso de inicio de sesión verificando los credenciales del usuario. Retorna true si el inicio de sesión es exitoso.
- buscaMail($email):
- Busca un usuario específico por correo electrónico.
- usuarioPorId($id):
- obtiene detalles de un usuario específico por su ID.
- actualizarusuario($usuario):
- Actualiza la información de un usuario existente. Esto incluye detalles como nombre, correo electrónico, y otros datos

relevantes. Retorna true si la actualización es exitosa, facilitando la confirmación al usuario o la corrección de errores.

- borrarUsuario($id):
- Elimina un usuario por su ID.

-Carpeta Utils:

Utils.php → Utilidad y ventajas de la Clase Utils:

- Centralización de Funcionalidades Comunes: Tiene varias operaciones utilitarias que son comunes a muchos componentes de la aplicación, evitando la duplicación de código y facilitando el mantenimiento.
- Facilidad de Uso: Al ser métodos estáticos, pueden ser llamados directamente sin necesidad de instanciar la clase, lo que los hace muy accesibles desde cualquier parte del proyecto.
- Mejora en la organización del Código: Ayuda a mantener el código de los controladores y otros componentes más limpio y enfocado, al externalizar operaciones utilitarias a una clase dedicada.
- Seguridad y Control de Acceso: Provee un método sencillo y centralizado para verificar los privilegios de administrador, lo cual es crítico para la seguridad y el control de acceso en la aplicación.

-Carpeta Views:

-Esta carpeta contiene varias vistas de las distintas secciones de la tienda online de informática; Ya sea del carrito, de las distintas secciones de cada categoría, así como la gestión de categorías desde el administrador. También incluye el inicio de la propia tienda al igual que los headers o el footer que se encuentra siempre visible en cualquier parte de la aplicación. Contiene también las vistas de la gestión de pedidos desde el admin o desde el cliente mostrando sus pedidos. El login, el registro, la gestión de los productos, sus propios detalles, y muchas mas vistas para no poner todas..

-Archivos dentro de la raíz de la carpeta Src:

.htaccess.php → El uso de Options -Indexes en el archivo .htaccess es una práctica recomendada para mejorar la seguridad y la privacidad en aplicaciones web al restringir el acceso directo a los directorios y evitar la exposición no intencionada de archivos y estructuras de directorios en el servidor.

init.php → Desempeña un papel fundamental como inicializador de la aplicación. Su principal función es preparar y establecer el entorno necesario para que la aplicación funcione correctamente.

El archivo init.php es crucial porque:

- Configura el entorno inicial: Prepara todas las dependencias, configuraciones y servicios necesarios para que la aplicación funcione.

- Centraliza la configuración inicial: Agrupa en un único lugar el arranque de la aplicación, facilitando su gestión y mantenimiento.
- Seguridad: Ayuda a gestionar aspectos de seguridad al cargar variables de entorno de manera segura y comenzar la sesión, que es vital para la gestión de autenticaciones y otras operaciones sensibles.

-Carpeta Vendor:

-Esta carpeta se crea y se gestiona automáticamente por Composer y contiene varios componentes importantes como:
-Librería de Terceros
-Autoload de Composer
-Archivos de Composer
Además de las librerías y el autoloader, la carpeta vendor también contiene:

- composer.json: Cada paquete dentro de vendor tiene su propio composer.json que describe el paquete, incluyendo metadatos como el nombre del paquete, versión, dependencias, etc.
- composer.lock: Este archivo no se encuentra dentro de vendor pero está estrechamente relacionado con lo que se instala en esta carpeta. composer.lock especifica las versiones exactas de cada paquete instalado en la última ejecución de Composer, asegurando que todas las instalaciones del proyecto en diferentes entornos sean consistentes.

-Ficheros que están en la propia raíz del proyecto:

.env → Se usa para gestionar configuraciones del entorno en las que se ejecuta la aplicación de PHP, evitando la necesidad de tener configuraciones codificadas directamente en el código fuente.

-Cada una de estas líneas define una variable de entorno que está aplicación PHP puede utilizar para configurarse. Aquí está el detalle de cada una:

1. SERVIDOR: Define la dirección del servidor de base de datos. En este caso, localhost indica que la base de datos se ejecuta en el mismo servidor que la aplicación web.
2. USUARIO: Especifica el usuario de la base de datos. root es el usuario administrador por defecto en muchos sistemas de gestión de bases de datos como MySQL o MariaDB.
3. PASSWORD: Define la contraseña del usuario de la base de datos. Está vacía en este caso, lo cual es común en entornos de desarrollo local pero no recomendado para producción debido a razones de seguridad.
4. BASE_DATOS: Especifica el nombre de la base de datos que la aplicación debe utilizar, en mi caso, tienda_online.

.gitignore → Su propósito es decirle a Git qué archivos o directorios ignorar en un proyecto, es decir, no rastrear ni incluir en el repositorio de Git. En mi caso es la carpeta vendor. Ya que así beneficiaría en la reducción del tamaño del repositorio, entre otras cosas..

.htaccess → El propósito de este .htaccess es asegurar que todas las solicitudes a tu aplicación PHP sean dirigidas a través del directorio public, que generalmente contiene el index.php (el enrutador frontal de muchas aplicaciones PHP modernas como Laravel, Symfony, etc.). Esto mejora la seguridad al asegurar que los archivos y directorios no públicos no sean accesibles directamente a través de la web.

Estas configuraciones ayudan a:

- Centralizar el manejo de solicitudes.
- Mejorar la seguridad al limitar el acceso directo a componentes no públicos.
- Facilitar el uso de URLs amigables para los usuarios y SEO.

Es importante asegurarse de que el servidor Apache tenga habilitado mod_rewrite y permita configuraciones .htaccess (AllowOverride All en la configuración del host virtual), para que este archivo funcione correctamente.