PROYECTO BACKEND UNIDAD 6

Edwin Steven Garzón León

c.c. 1010246651

correo: [steven.esql@hotmail.com](mailto:steven.esql@hotmail.com)

Repositorio github del proyecto: <https://github.com/esgarzonl/ecommerce.git>

Servidor del proyecto:

Contenido

[Frameworks, Lenguajes y demás ítems utilizados 1](#_Toc137583949)

[Modelo de la base de datos 2](#_Toc137583950)

[Creación de proyecto 2](#_Toc137583951)

[Crud de productos. 4](#_Toc137583952)

[Carrito De Compras 6](#_Toc137583953)

[Crear tabla pivot 7](#_Toc137583954)

[Modulo pagos con API de paypal 9](#_Toc137583955)

[Módulo de pagos 12](#_Toc137583956)

[Módulo de compras 13](#_Toc137583957)

[Módulo de ordenes (admin) 16](#_Toc137583958)

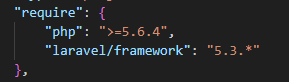
[Autenticaciones 17](#_Toc137583959)

[Subida de imágenes 19](#_Toc137583960)

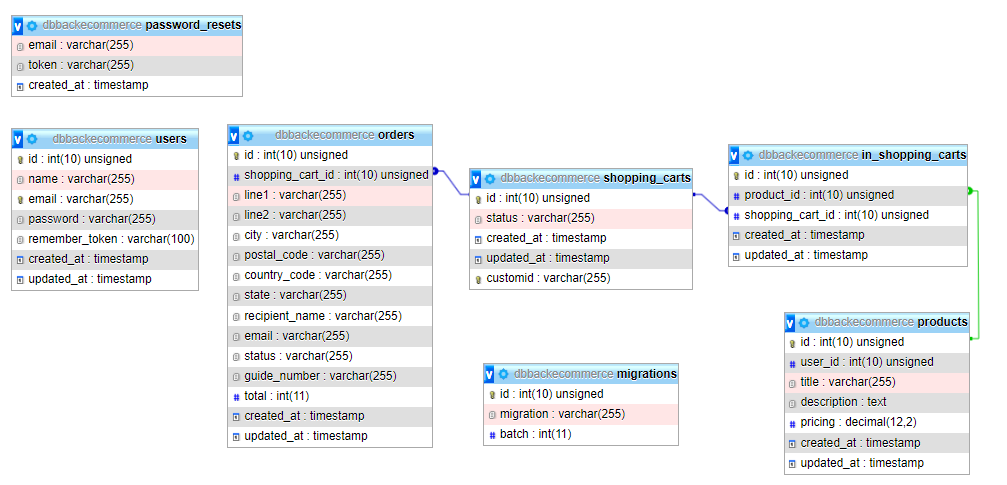
## Frameworks, Lenguajes y demás ítems utilizados

* Usaremos laravel 5.3.\*
* PHP >= 5.6.4
* Base de datos: PhpMyAdmin
* Modelo de base de datos: Relacional
* Bajo la arquitectura Modelo Vista Controlador (MVC)

Como se ve en nuestro json las versiones que usaremos



## Modelo de la base de datos

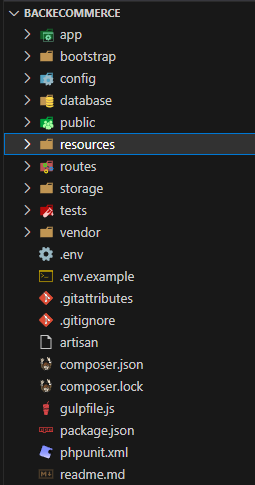


## Creación de proyecto

Creamos el proyecto llamado BACKECOMMERCE dentro del xammp



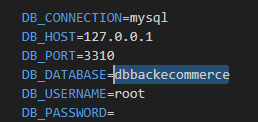
Se ve nuestro proyecto



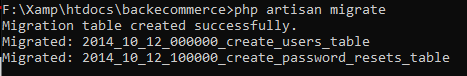
Creamos la base de datos y configuramos variables de entorno en el archivo .env para que nadie mas que el servidor tenga acceso a estas



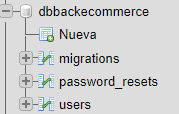
Variables de entorno archivo .env



Migramos para que se generen las dos tablas de usuario y reinicio de contraseña



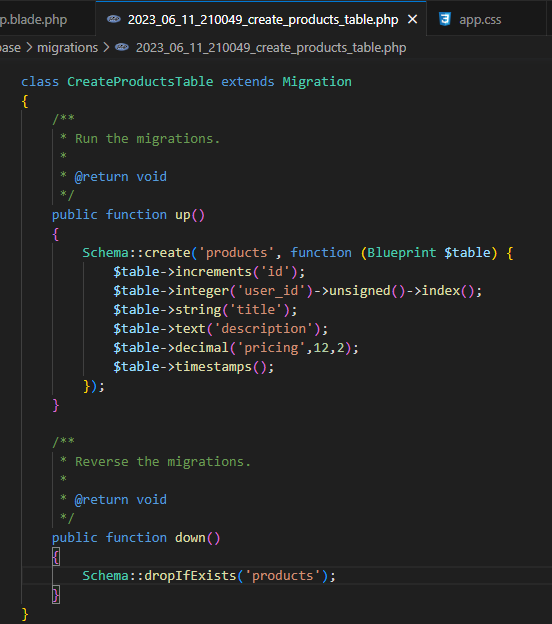
Y vemos creadas las tablas



## Crud de productos.

Creamos la tabla de productos desde la consola de laravel 

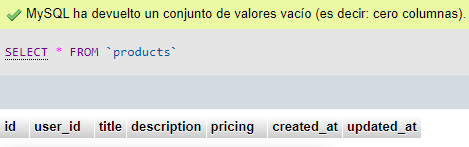
Y nos crea el archivo php de migración al cual le vamos a asignar los campos que tendrá la tabla y las respectivas funciones.



Migramos las declaraciones

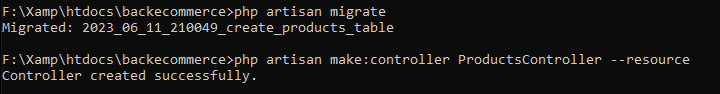


Y vemos los cambios en la base de datos de phpMyAdmin



Ahora crearemos los Controladores para el área de productos, el cual contendrá los métodos para llevar a cabo el CRUD de la tabla products.

Creamos el controlador desde la consola



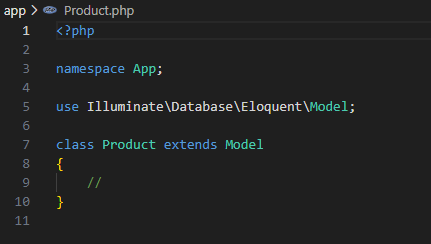
Y así tenemos nuestro controlador.



Ahora creamos el modelo para permitir el manejo de datos de la tabla Products



Y nos crea nuestro modelo



CRUD de productos finalizado:



## Carrito De Compras

El carrito de compras, se guardará para cada usuario, así cuando este salga de su sesión y vuelva a entrar, su carrito seguirá almacenado en la bd, bajo el status de pendiente.

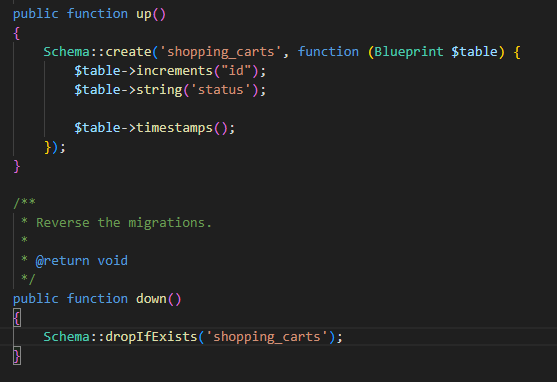
Para crear la tabla shopping\_cart se crea desde laravel



Y creamos el modelo para el manejo de datos



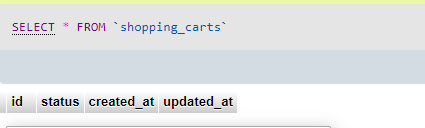
Creamos nuestras funciones



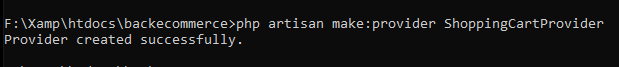
Y migramos esto para que se cree nuestra tabla en la base de datos



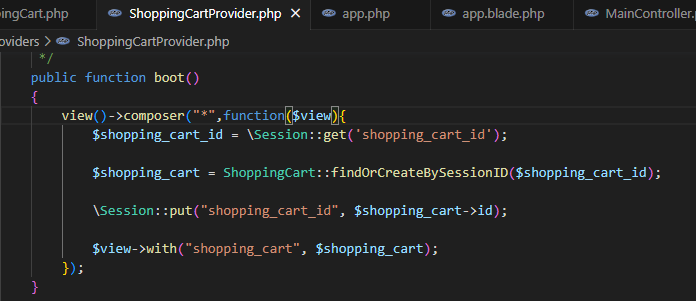
Y tenemos nuestra tabla en la base de datos.



Creamos nuestro provider de la clase ShoppingCart, el cual será el encargado de guardar el id del carrito de compras de la sesión para inyectarlo en las demás vistas.



Configuramos el provider para que el id del carrito quede guardado para cada sesión y se mantenga en todas las vistas de la web.



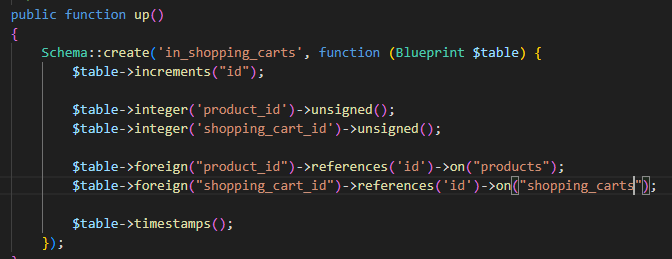
## Crear tabla pivot

Como tenemos dos tablas: products y shopping\_carts, y su relación es muchos a muchos, vamos a crear la tabla pivote que permite relacionar estas dos tablas para así en cada carrito mostrar todos los productos que el usuario a registrado. Esta tabla se llamará In\_shopping\_carts

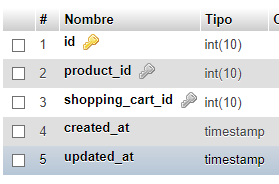


Y creamos el modelo de la misma



Definimos la relación 

Y migramos a la bd para tener nuestra tabla creada donde se ven las llaves foráneas de la relación entre products y shopping\_carts



Creamos el controlador de la tabla in\_shopping\_carts, el cual nos permitirá agregar productos al carrito de cada sesión.



Teniendo esto, creamos el controlador de ShoppingCarts para que el usuario pueda manejar a su gusto el carrito de compras que ha seleccionado.

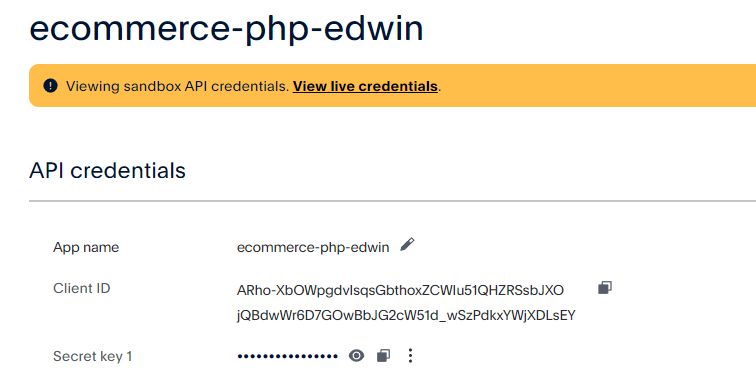


Luego de configurarlo, el usuario podrá ver su carrito de compras de la siguiente manera:

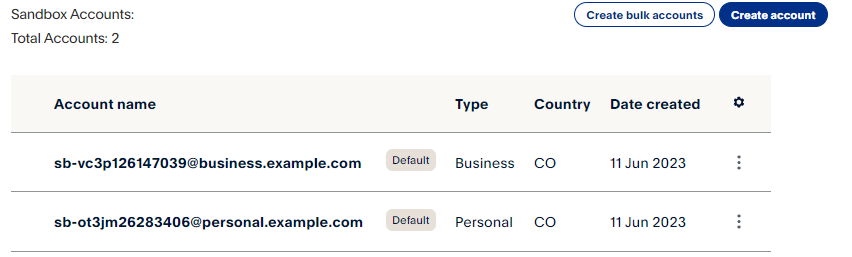


## Modulo pagos con API de paypal

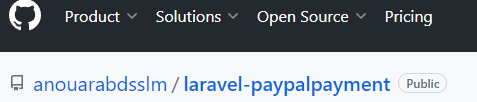
Para desarrollar el modulo de los pagos, se usará el API de paypal, el cual nos permite llevar a cabo el desarrollo e incluso pruebas para nuestro ecommerce, así que generamos los datos para el API junto con las cuentas de pruebas.



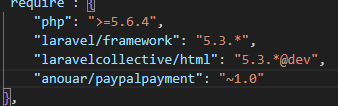
Cuentas de pruebas



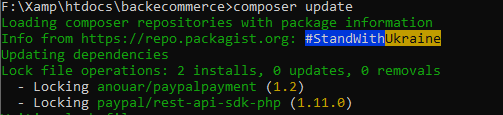
Ahora vamos a incluir la siguiente librería en nuestro laravel.



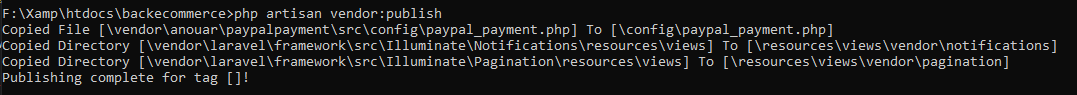
Para lo cual agregamos esta a nuestro composer.json



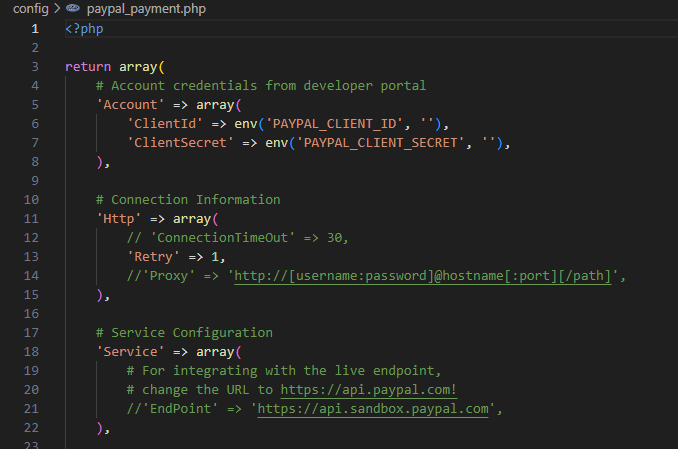
Y actualizamos nuestro composer



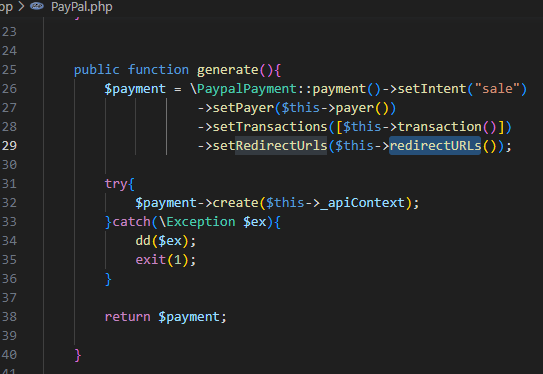
Una vez instalado, generamos el archivo de configuración de la librería



Y nos genera el archivo ‘.php ‘ con el cual se harán los cobros a través del API

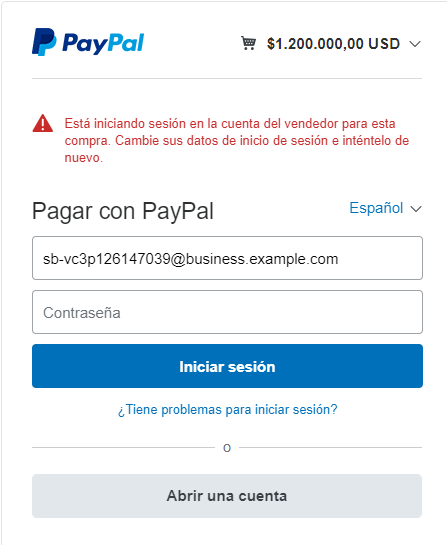


Creamos nuestro archivo modelo para el API y configuramos todos los requerimientos del api.



Una vez hecho esto, al dirigirnos a nuestro carrito debe enviarnos a la pagina de PayPal para hacer el pago, en donde debemos ingresar las credenciales generadas por el API para pruebas.

Además veremos el total a pagar.



Iniciamos sesión y veremos el pago.



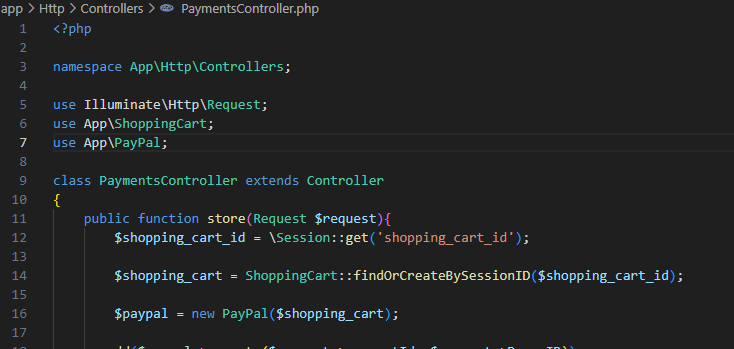
## Módulo de pagos

Ahora crearemos la tabla de Payments para tener el registro de pagos.

Creamos el controller de Payments.



Lo modificamos para realizar los pagos.



Y el API nos valida la compra y transfiere el dinero internamente.

## Módulo de compras

Para que el admin pueda ver el historial de compras, estado de estas, dirección de entrega y demás información, haremos el CRUD de órdenes, comenzando con la tabla en la base de datos la cual tendrá una relación con shopping\_cart para generar el historial.

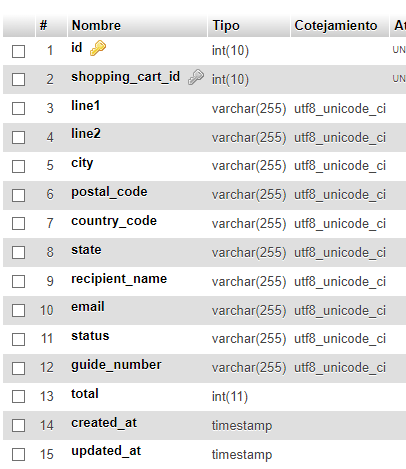


Y ajustamos la función con los campos que tendrá la tabla en la base de datos



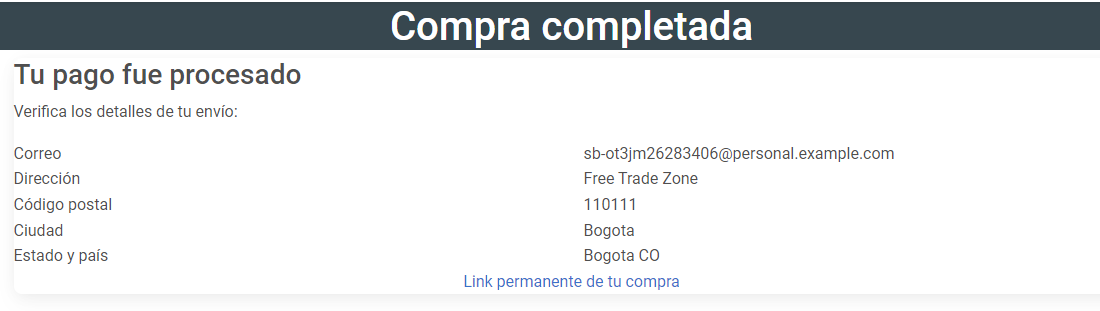
Migramos y vemos nuestra tabla en la base de datos.





Para trabajar sobre orders, creamos el modelo.

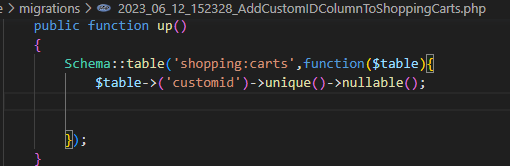


Junto a las dos cuentas Sandbox, personal y business, para que al comprar, nos redirija a una vista de nuestro proyecto con los datos de la compra.

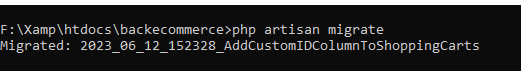
Ahora configuramos el enlace permanente para que se guarde el carrito de compras, para lo cual necesitamos un nuevo campo en la tabla shopping cart, el cual vamos a crear



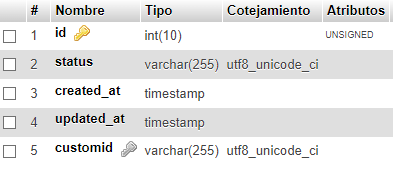
Configuramos la nueva columna de la tabla



Y migramos para que se cree la tabla en la bd



Y la vemos reflejada en nuestra base de datos



Configuramos los modelos necesarios, y al hacer nuestra compra ya nos genera el estado aprobado del carrito.



Y así cada carrito tendrá su url consecutiva cifrada.

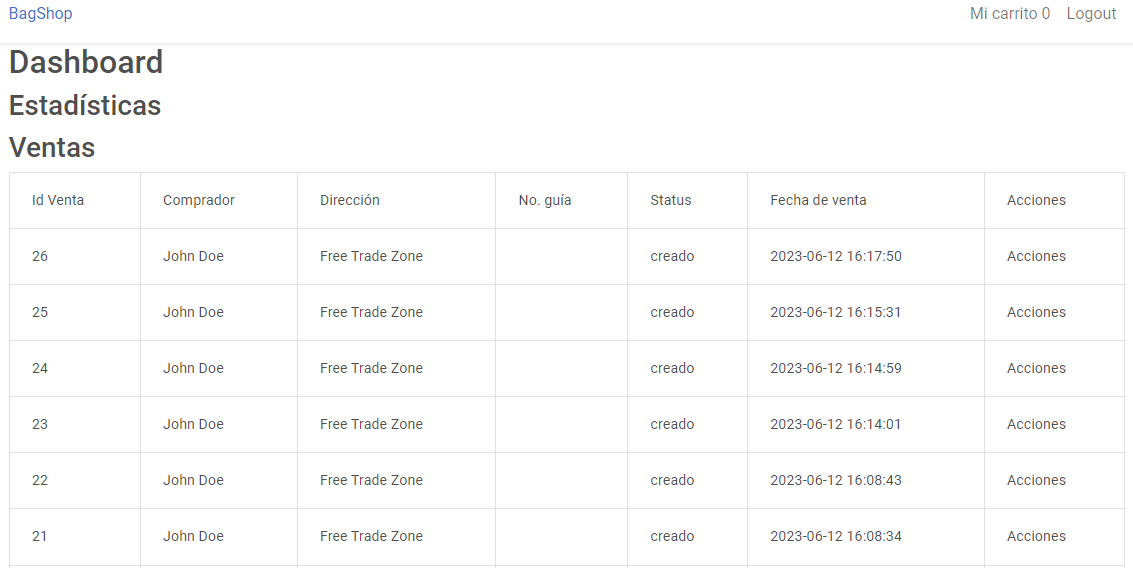


## Módulo de ordenes (admin)

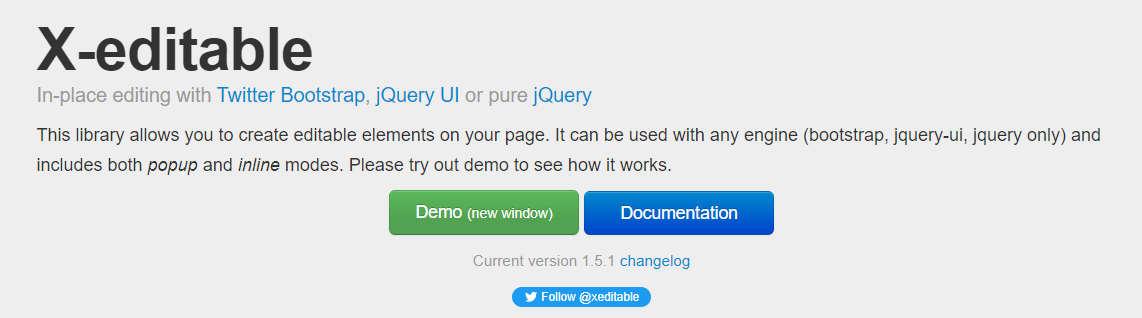
Ahora vamos a crear la vista para que el admin pueda ver todas las órdenes del mes.



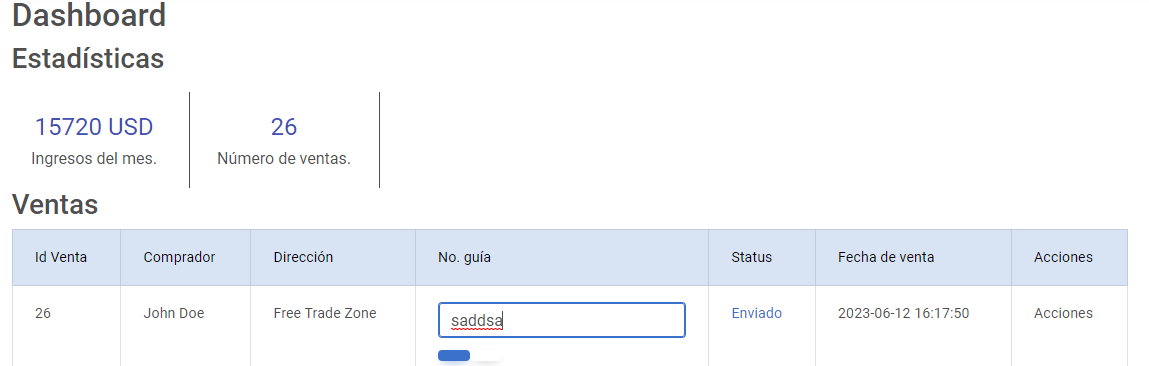
Creamos el controlador de las ordenes junto a todos los métodos y ya tenemos la primer vista de ordenes.

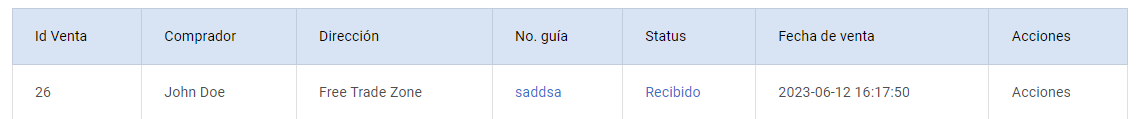


Para las acciones del CRUD de orders vamos a usar la librería XEditable, que permite editar en tiempo real



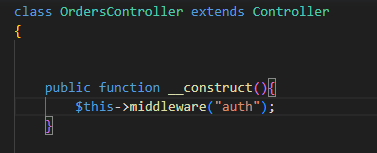
Importamos y ajustamos la vista, para poder editar los campos



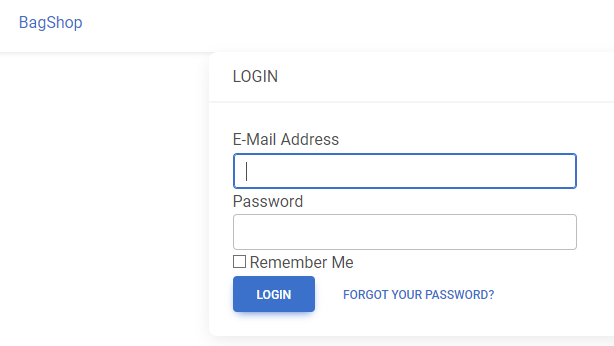


## Autenticaciones

Completados los CRUD de ordenes y de productos, haremos las validaciones de autenticación para que solo los usuarios logeados puedan acceder a estas, creando el middleware de autenticación que tiene por defecto laravel.



Así al intentar entrar a una pagina de Orders sin estar logeado, redirigirá al Login

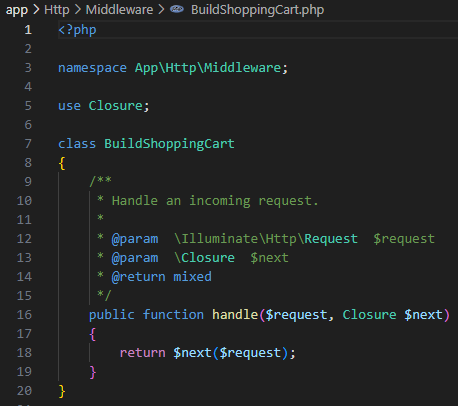


Y hacemos lo mismo para el CRUD de products.

Para continuar con las autenticaciones creamos un middleware para almacenar el carrito y que se pueda acceder a este sin necesidad de estar logeado o estándolo.



Y se crea en nuestro proyecto





Ahora el carrito se guardará para usuarios logeados o no loggeados.

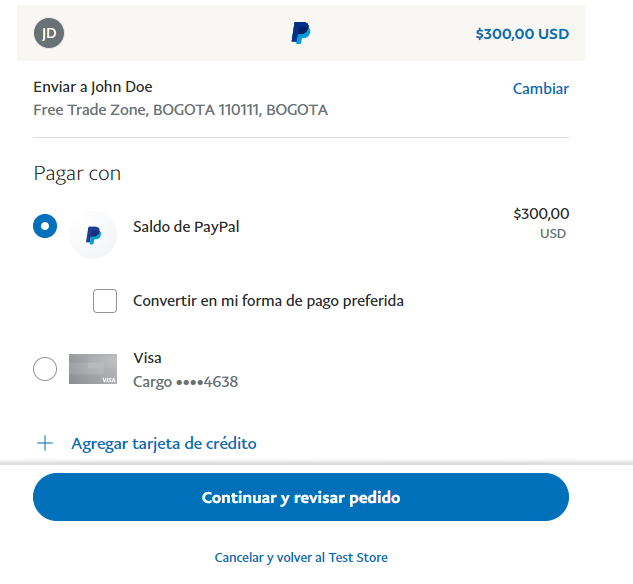
Carrito de un usuario no logeado:



Ahora configuramos un form en el carrito para que al darle a un botón pagar nos conecte con el API de paypal y podamos realizar el pago en la pagina.



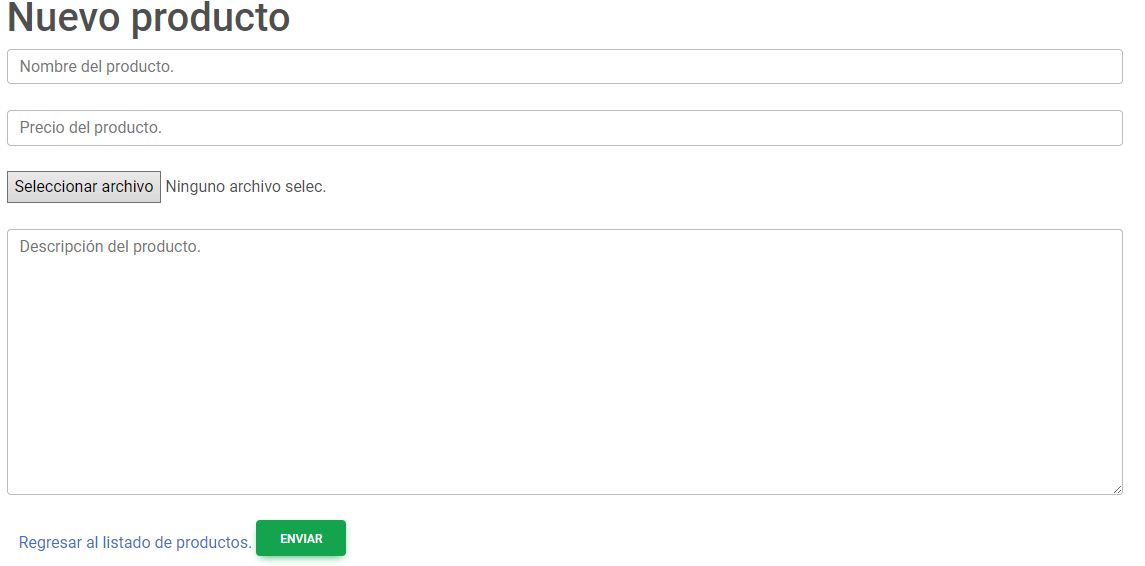
Y al hacer click en pagar:



## Subida de imágenes

Ahora configuraremos la creación de productos para que permita subir una imagen para cada producto.

Configuramos los controladores y el formulario de products, para que solicite una imagen al crear el registro.



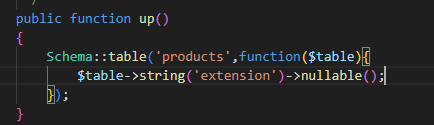
Y en la base de datos agregamos un campo a la tabla products que nos permita almacenar la extensión del archivo para así saber que archivo es, y como traerlo, ya que este se guardará en el framework bajo el nombre de idproducto.extensión.



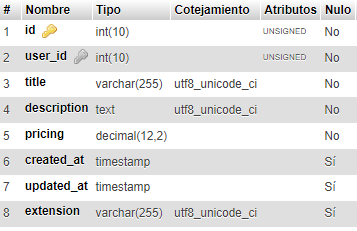
Creamos el campo en la base de datos tabla products.



Creamos la nueva columna



Y migramos para ver el cambio en la bd



Y después de ciertas configuraciones, tenemos nuestros productos con imagen

