larave<http://www.cristalab.com/tutoriales/como-instalar-laravel-y-composer-c111363l/>

sugerencia: debe tener instalado el git y también debe estar como variable global

<https://getcomposer.org/download/> entra a comoser e instala, en la dirección donde esta el xamp php

luego en cmd teclea

composer global require “laravel/installer”

para crear una carpeta debe estar en la carpeta htdoc en el cmd cuando estemos en la carpeta teclear

laravel new ecommerce

y empezara a crear todos los archivos y dependencias

una vez hecho para acceder a la pagina en laravel

al activar el xamp vamos a

<http://localhost/ecommerce/public/>

el archivo .env –es donde se establece la conexión con la base de datos  
La configuración varia según el usuario fuera de los datos que se compartan

Este archivo nos puede ayudar para cuando querramos cambiar de pruebas a produccion

para conectarse con la base de datos de xamp

(nombre base de datos) debemos crear un phpadmin la tabla

DB\_DATABASE=products\_course

DB\_USERNAME=root

DB\_PASSWORD=

Para configurar de forma segura el aplicativo es la clave de aplicación

La ejecutamos php artisan key:generate

El la carpeta databse en migraciones lo que hace esc rear un table en la base de datos

Remember token le da características

Timestamps crear 2 campos que guarda fecha y hora de creación del registro y la ultima vez que se actualizo

Vamos a la dirección en donde esta nuestro proyecto en cmd

C:\xampp2\htdocs\ecommerce

y ejecutamos “php artisan migrate” que ejecuta el archivo artisan que tiene varios comandos

lo que hace es ejecutar los archivos que no se hayan migrado previamente y estén en la carpeta de migrations, ojo en env debe estar la base de datos y también debe estar creada en phpadmin

la tabla migreate tiene la tabla de migración y si fue migrada aquí se están que archivos ya fueron ejecutados y no volver a ejecutarlos

Estructura del proyecto  
app – contiene el código principal del aplicativo  
config – Archivos de configuración: base de datos, cache, correos, sesiones, configuración en general  
database – Aquí se incluye todo lo relacionado con definición de la base de datos, tres carpetas factores, migrations, sedes   
public – Unica carpeta publica visible, todas las peticiones pasan por allí, el index inicia todo el proceso de ejecución del framework, se aloja css, javascript, imágenes  
Resources – carpetas assets, views y lang  
resoruces/lang – se guardan archivos php que contiene arrays con textos  
resources/assets – almacena fuentes tipo less o sass  
storage – se alamcena toda la información interna para la ejecución, como archivos, cache, compilación de las vistas, meta información y logs de sistema, normalmente solo se utiliza para ver logs  
test - fichero de pruebas automatizdas proceso de pruebas con PHPUnit  
vendor – alojan las libreris y dependencias no es necesario modificar  
.env – se utiliza para la configuración de la maquina   
composer .json – instalación de paquetes

Mas de una ruta

Route::match(array(‘GET’,’POST’), ‘/’ , function()

{

Return ‘¡hola mudno’

});

App

App/Http/Controllers – controladores sirven para interactuar con los modelo, vistas y maneja la lógica de la aplicación

App/Http/Middleware – filtros y clases intermedias que podemos utilizar para determinadas acciones , validación de permisos, antes y después de a ejecución de la aplicación

App/Http/routes – Define todas las rutas de nuestro sitio web

Ejecucion de cache – solo funciona con route controller

Php artisan route:cache

Aceleramos el proceso de registro de rutas, en preferible que se realice cuando enviemos a produccion

Php artisan route:clear

el comando “php artisan make:auth”

make es un comando que permite genrear diversas cosas archivos que sirven con el modulos de autenticación

en nuestra carpeta app/http/Controllers/Auth se generaron archivos

controladores que nos sirven para autenticar a los usuarios

también en la carpeta resources/views/auth/passwords

son formularios de autenticación y utilizan bootstrap y utilizan en la carpeta resources/views/ layout app.blade.php

ir a <http://fezvrasta.github.io/bootstrap-material-design/>

o

agregar la versión cloud es bueno en caso de que en otra pagina ya tenga la dirección y entramos a ella ya tenemos cachedo la direccion

<link rel="stylesheet" type="text/css" href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

Ahora hay que importer bootstrap material design

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-material-design/0.3.0/css/material-fullpalette.min.css">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-material-design/0.3.0/css/ripples.min.css">

Abajo de script agregar

Se puede obtener en code.jquery.com ir por la versión mimificada

<script src="https://code.jquery.com/jquery-3.1.0.min.js" integrity="sha256-cCueBR6CsyA4/9szpPfrX3s49M9vUU5BgtiJj06wt/s=" crossorigin="anonymous"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

Si se quiere inicializar el material design se debe

<script>

$.material.init();

</script>

Pero antes hay que agregar

<script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-material-design/0.3.0/js/material.min.js"></script>

Y ripple – el efecto de onda de material design

<script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-material-design/0.3.0/js/ripples.min.js"></script>

Layout – es un diseño de tu estructura web que se va a repetir

(Arroba)yield(‘content’) - se encarga de importar el contenido de otras vistas o seccion

(Arroba)extends(‘layouts.app’) la vista va a utilizar la carpeta el archivo app de la carpeta layouts

(Arroba)(‘content’) – aquí se define el contenido de la sección

La funcionalidad de vistas mencionadas anteriormente viene del mortor de vistas preconfiado de laravel llamado blade, por eso algunos archivos tiene el nombre blade.

Tiene dos caracterias todo lo dentro de las dobles llaves es un impresión

{{ config(‘app.name’, ‘Laravel’) }}  
es igual a   
<?php echo (config(‘app.name’, ‘Laravel’)); ?>

(Arroba) todo lo que empiece arroba y que el motor lo entiende cuando llega una petición al servidor se ejecutar como código y se ejecutar como html al navegador

Nuestra primera vista

Para cada nueva pagina debemos crear un controlador

Debe ir a la carpeta app/http/controllers/Auth - y crear el archivo MainController

Como sugerencia sigue el formato de letras para cada grupo de carpetas

Namespace es como una carpeta cada clase debe tener un nombre de espacio para que no conflictue con nombres repetidos

<?php  
marca la ruta donde esta el archivo

namespace App\Http\Controllers;

hacemos nuestro main controller herede a la clase controller

class MainController extends Controller{

}

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;

class MainController extends Controller{

public function home(){

}

}

Ahora vamos a routes/ web.php

La ruta de crear poniendo el nombre del controlado un arroba y su funcion

Route::get('/', 'MainController@home');

Auth::routes();

Route::get('/home', 'HomeController@index');

Para mostrar un vista

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;

class MainController extends Controller{

public function home(){  
 regresa una vista

return view('main.home',[“name ” => “Carlos”]);

} crear un vista dentro de la carpeta main que se llame home

return view('main.home',[]) – lo que hay dentro [] son los datos que va a recibir el usuario del controlador

Creamos una carpeta main en resources/view y dentro de esa carpeta un archivo home.blade.php

Agregamos

{{$name}} para imprimir la variable name

Dejamos con

{{$name}}

@extends('layouts.app')

@section('title', 'Productos Facilito')

@section('content')

<h1>Bienvenidos a esta tienda</h1>

@endsectionA nuestro [home.blade.php](http://home.blade.php)

{{$name}}

Buscar la carpeta layouts en view el archivo app

@extends('layouts.app')

Allí en la sección con le etiqueta titile tendrá

@section('title', 'Productos Facilito')

Y la etiqueta content tendra

@section('content')

<h1>Bienvenidos a esta tienda</h1>

@endsectionA nuestro home.blade.php

Para gulp

Entramos

Resources/assets/sass/app.scs

El procesamiento de los archivos sass se hace mediante gulp autimatizador del flujo de trabajo

El archivo gulpfile.js

elixir((mix) => {

mix.sass('app.scss')

.webpack('app.js');

});

Conviente el archivo app.scss en app.css

Para eso necesitamos descarga el node.js

Nos dirigimos a nuestra carpeta donde tenemos nuestra carpeta ecomerce que ya tiene el archivos gulp.js y packeage.json aquí es donde tiene todos los elementos de necesario para la instalación solo necesitamos hacer npm install y automáticamente instala todos los elementos

Si hay algun error npm install -g npm@latest para actualiza el npm

npm install -g npm

Debemo  
npm install gulp - instalar gulp  
 npm install –g gulp - instalar gulp de forma global  
 instala de manera global

Npm install laravel-elixir en caso de que gulp necesite el modulo puede ser global o no   
Npm install laravel-elixir –g - global  
Npm install laravel-elixir-vue-2 - local  
Npm install laravel-elixir-vue-2 –g - global  
npm install --no-bin-links -g - global evitar el error

cannot read property mergeCOnfigs of undefined

npm install install webpack

npm install [minimatch@3.0.2](mailto:minimatch@3.0.2)

o puede haber problema con el git

npm install git

o buscar entre el path de variable global, buscar la dirección donde esta instalado nuestro git (debe estar instalado)

1. Right click on Computer.
2. Click Advanced System Settings
3. Click Environment Variables inside the Advanced Menu
4. Under System Variables, scroll to PATH
5. Add ;"C:\path\to\git\bin";"C:\path\to\git\cmd"

Y gulp watch para ver los cambios

En app.blade php

Cambiams la ruta del css a <link href="{{ url('/css/app.css') }}" rel="stylesheet">

Datos a tener en cuenta

{

En caso de que no exista agregar name y version

"name": "ecommerce",

"version": "0.0.1",

"private": true,

"scripts": {

"prod": "gulp --production",

"dev": "gulp watch"

},

"devDependencies": {

"bootstrap-sass": "^3.3.7",

"gulp": "^3.9.1",

"jquery": "^3.1.0",

"laravel-elixir": "^6.0.0-14",

"laravel-elixir-vue-2": "^0.2.0",

"laravel-elixir-webpack-official": "^1.0.2",

"lodash": "^4.16.2",

"vue": "^2.0.1",

"vue-resource": "^1.0.3"

}

}

REST – tipo de arquitectura como diesñas la arq de tu app web, características es enfoque en recursos no intereccion

Con rest si desarrollas el formulario a donde tiene que ir para que se pueda crear   
verbos estandarizados POST/DELETE la url no describe la interaccion sino el recursos

Basdo HTTP arquietctura worldwideweb

La Uri representa recursos no acciones

Las acciones lo definen los verbos http

Un recursos puede tener multiples representaciones

Los verbos que se utilizan son,

POST = Crear  
GET = Mostrar  
PUT/PATCH = Actualizar   
DELETE = Eliminar

Iniciar session es un peticion POST porque estamos creando una sesión

Cerrar sesión es una petición DELETE eliminar la sesión

Pagar el carrito de compras –pude ser post si enfocamos que creamos un pago o puede ser put o patch en el punto de vista de actualizar

Utilizar el bash de git o cerrar el node y abrirlo

Escribimos en nuestra carpeta y ejecutamos el siguiente comando para crear una migración para crear algo comenzamos con Php artisan make: y depende de la accion

Php artisan make:migration create\_products\_table

Una vez ejecutada se creara en nuestra capeta/database/migrations/fecha y código de la migración

En una migración existe dos métodos up y downd

Función up realiza cambios y down revertir los cambios

\* @return void

\*/

public function up()

{

//

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

//

}

Los cambios se realizan mediante

Schema:: que tiene varios métodos

Schema::create('products',function(Blueprint $tabla){

$tabla->increments("id");

$tabla->integer('user\_id')->unsigned()->index();

$tabla->string('title');

$tabla->text('description');

$tabla->decimal('pricing',9,2);

$tabla->timestamps();

});

El primer parámetro es el nombre y después una función bluprints

La tabla siempre debe tener un id incremental y un timestamps para saber cuando se realizaron los cambios, user es un fk de la tabla usuario, string es para texto corto y text es para una gran cantidad de caracteres como los de un blog, decimal tiene tres valor y nombre de la columna el numero de números y tercero el numero de decimales

public function down()

{

//

Schema::drop('products');

}

Para nuestra función down que elimina todo lo de up

Una vez escrito la funciones de nuestra migración

En nuestro comand ejecutamos

php artisan migrate - y se creara en nuestra base de datos

en caso de que nos hayamos equivocado utilizamos el comando

php artisan migrate:rollback – ejeuctar la función down

esto permite llevar un registro de los cambios

Resources controller

Para crear un controller ejecutamos y para que sea un resource controller le agramos --resource

Php artisan make:controller ProductsController –resource

Se creara el controller en app/Http/Controllers

La funciones son

Index() - muestra la colleccion de recursos puede todos o algunos

Create() - despliega la vista para poder crear un nuevo producto

Store() – al desplegar la vista de compra, store es el encargado de guardar el nuevo producto

Show (id) – muestra el producto con su id

Edit(id) – muestra un formulario para editar el producto

Update(request,id) - actualiza todo lo que se envio en edit al id

Destroy(id) – eliminar el id (el producto con el id)

AL principio nuestro controlador esta desconectado de nuestra plataforma para ello vamos a la carpeta routes /web.php y agregar nuestro controlador como como controlador de recursos

Route::resource('products','ProductsController')

El primer argumento define el recuros con el que vamoa a trabajar es importante por que

Route::resource('products','ProductsController') - crea la siguientes rutas

GET /products => index  
POST /products => store  
GET /products / create => formulario para crear

GET /products /:id => index Mostrar un prodcuto con id  
GET /products /:id/edit  
PUT /PATCH/ products /:id  
DELETE /products /:id

Nuestro controlador debe estar guardado tambien en un view, vamos Resources/views y creamos una carpeta products

Ahora en nuestro controller en la función index()

public function index()

{

return view("products.index")

}

Ahora vamos a reurces/view/products/ y creamos un archivos index.blade.php

Agregar

@extends("layouts.app");

@section("content")

<div class="big-padding text-center blue-grey white-text">

<h1>Productos</h1>

</div>

<div class="container">

<table class="table table-bordered">

<thead>

<tr>

<td>ID</td>

<td>Titulo</td>

<td>Descripcion</td>

<td>Precio</td>

<td>Acciones</td>

</tr>

</thead>

<tbody>

</tbody>

</table>

</div>

@endsection

Ahora necesitamos un modelo para ello vamos a la sección de comandos

Php artisan make:model Product

El modelo se crea en app/Product.php de preferencia debe llamarse en singular de la tabla

En nuestro ProductsController debemos importar el modelo

Use App\Http\Requests;

Use App\Product;

Y en index()

Escribimos

$products = Product::all(); para obetener todos los productos de la tabla

return view("products.index",["products"=> $products ]);

colocamos como parametro para enviar un nombre product y la lista de productos (variable)

de nuestra archivo

resources/views/products/index.blade.php

debe estar asi

<tbody>

@foreach ($products as $product)

<tr>

<td>{{ $product->id }}</td>

<td>{{ $product->title }}</td>

<td>{{ $product->description }}</td>

<td>{{ $product->pricing }}</td>

<td> Acciones</td>

</tr>

@endforeach

</tbody>

Insertamos un nuevo registro en nuestra tabla products

Si se quiere actualizar los assets

Se busca una sección donde haya problemas con los css, mira en sass/app.sass y se modifica la seccion

Para crear nuevos productos vamos a la función create() de ProductsControllers

public function create()

{

//

return view("products.create");

}

Y vamos a resources/views/products y creamos el archivo create.blade.php

@extends("layouts.app");

@section("content")

<div class="container white"></div>

@endsection

Para aprovechar una librería nos vamos a composer.json

En la sección agregamos

"require": {

"php": ">=5.6.4",

"laravel/framework": "5.3.\*"

“laravelcollective/html”: [5.3.\*@dev](mailto:5.3.*@dev), “agregamos”

},

Esta librería son un conjunto de helpers fuciones de clases y método que permiten generar html

Una vez agregados vamos a comandos ejeuctamos “composer update”

Esperar un poco

Después ir a config/ app.php e ir a la sección de provider

Agregamos Collective\Html\HtmlServiceProvider::class,

Y despues avamos a la seccion aliases agregamos

'Form' => Collective\Html\FormFacade::class,   
'Html' => Collective\Html\HtmlFacade::class,

Para poder llamar al provider mediante un alias, y podremos crear nuestro formularios

Se utilizan el siguiente tipo de código para crear form es porque no todos lo navegadores crean los con todos los métodos, en el create.blade.php

<div class="container white">

<h1>Nuevo producto</h1>

{!! Form::open(['url' => '/products', 'method' => 'POST']) !!}

<div class="form-group">

{{Form::text('title','',['class' => 'form-control',

'placeholder'=>'Titulo...']) }}

</div>

<div class="form-group">

{{Form::number('pricing','',['class' => 'form-control',

'placeholder'=>'Precio de tu producto en centavos de dolar']) }}

</div>

<div class="form-group">

{{Form::textarea('description','',['class' => 'form-control',

'placeholder'=>'Describe tu producto...']) }}

</div>

<div class="form-group text-right">

<a href="{{url('/products')}}"> Regresar al listado de productos </a>

<input type="submit" value="Enviar" class="btn btn-success">

</div>

{!! Form::close() !!}

</div>

Datos en paypal hay una desventaja de payapal cobro por el api rest esta limitado a que moneda acepta solo puede ser el dólares o tu moneda local y el backend transfórmalo a dólares

En app.scsss, agregar

.floating{

position:fixed;

bottom:2em;

right:2em;

z-index:100;

}

En index.blade.php,agregar para que se cree un botón para crear nuevos productos

<div class="floating">

<a href="{{url('/products/create')}}" class="btn btn-primary btn-fab">

<i class="material-icons">add</i>

</a>

</div>

En app.blade.php agregar

<link rel="stylesheet" type="text/css" href="//fonts.googleapis.com/icon?family=Material+Icons">

Para poder utilizer los iconos d material desiegn

Ahora para guardar el producto debemos ir a ProductController e ir a la función store()

Agregamos

use Illuminate\Support\Facades\Auth;

public function store(Request $request)

{

//

$product = new Product;

$product->title = $request->title;

$product->description = $request->description;

$product->pricing = $request->pricing;

$product->user\_id = Auth::user()->id;

if($product->save()){

return redirect("/products");

}else{

return view("products.create");

};

}

Los títulos deben coincideir con los datos del formulario, el user id será tomado de la autenticación del login, si graba se redirecciona a /products y si no vuelve a la vista products.create

Para la autenticación debe tener

use Illuminate\Support\Facades\Auth;

creamos un edit.blade.php en views/products

también creamos un form.blade.php el cual guardaremos nuestro form tanto de edit como create en ambos llamamos al form con un

@include('products.form',['product' => $product])

Mientra que en nuestro metodo create y edit agregamos o cambiamos

public function edit($id)

{

//

$product = Product::find($id);

return view("products.edit",["product" => $product]);

}

Product::find($id) - busca el product con el id

Create.blade.php

@extends("layouts.app");

@section("content")

<div class="container white">

<h1>Nuevo producto</h1>

@include('products.form',['product' => $product, 'url' => '/products', 'method' => 'POST'])

</div>

@endsection

Edit.blade.php

@extends("layouts.app");

@section("content")

<div class="container white">

<h1>Nuevo producto</h1>

@include('products.form',['product' => $product, 'url' => '/products/'.$product->id, 'method' => 'PATCH'])

</div>

@endsection

Form.blade.php

{!! Form::open(['url' => $url, 'method' => $method]) !!}

<div class="form-group">

{{Form::text('title',$product->title,['class' => 'form-control',

'placeholder'=>'Titulo...']) }}

</div>

<div class="form-group">

{{Form::number('pricing',$product->pricing,['class' => 'form-control',

'placeholder'=>'Precio de tu producto en centavos de dolar']) }}

</div>

<div class="form-group">

{{Form::textarea('description',$product->description,['class' => 'form-control',

'placeholder'=>'Describe tu producto...']) }}

</div>

<div class="form-group text-right">

<a href="{{url('/products')}}"> Regresar al listado de productos </a>

<input type="submit" value="Enviar" class="btn btn-success">

</div>

{!! Form::close() !!}

Index.blade.php

El foreach cambiamos a

@foreach ($products as $product)

<tr>

<td>{{ $product->id }}</td>

<td>{{ $product->title }}</td>

<td>{{ $product->description }}</td>

<td>{{ $product->pricing }}</td>

<td>

<a href="{{url('/products/'.$product->id.'/edit')}}">

Editar

</a>

</td>

</tr>

@endforeach

Para actualizar se utiliza la función update()

public function update(Request $request, $id)

{

//

$product = Product::find($id);

$product->title = $request->title;

$product->description = $request->description;

$product->pricing = $request->pricing;

if($product->save()){

return redirect("/products");

}else{

return view("products.edit",["product" => $product]);

};

}

Para eliminar

@foreach ($products as $product)

<tr>

<td>{{ $product->id }}</td>

<td>{{ $product->title }}</td>

<td>{{ $product->description }}</td>

<td>{{ $product->pricing }}</td>

<td>

<a href="{{url('/products/'.$product->id.'/edit')}}">

Editar

</a>

@include('products.delete',['product' => $product])

</td>

</tr>

@endforeach

En la carpeta products/ creamos un delete.blade.php

{!! Form::open(['url' => '/products/'.$product->id, 'method' => 'DELETE', 'class' => 'inline-block' ]) !!}

<input type="submit" class="btn btn-link red-text no-padding no-margin no-transform" value="Eliminar">

{!! Form::close() !!}

En product controller en destroy

public function destroy($id)

{

//

Product::destroy($id);

return redirect('/products');

}

El método show()

public function show($id)

{

$product = Product::find($id);

return view("products.show",["product" => $product] );

}

Creamos un show.blade.php

@extends('layouts.app')

@section('content')

<div class="container text-center">

<div class="card product text-left">

@if(Auth::check() && $product->user\_id == Auth::user()->id)

<div class="absolute actions">

<a href="{{url('/products/'.$product->id.'/edit')}}">

Editar

</a>

@include('products.delete',['product' => $product])

</div>

@endif

<h1>{{$product->title}}</h1>

<div class="row">

<div class="col-sm-6 col-xs-12"></div>

<div class="col-sm-6 col-xs-12">

<p>

<strong>Descripcion</strong>

</p>

<p>

{{$product->description}}

</p>

<p>

<a href="" class="btn btn-sucess"> Agregar al carrito </a>

</p>

</div>

</div>

</div>

</div>

@endsection

La seccion la condicion verifica verificar si el usuario esta logeado y si el producto fue creado por el usuario logueado le permite modificarlo o eliminarlo

@if(Auth::check() && $product->user\_id == Auth::user()->id)

<div class="absolute actions">  
<a href="{{url('/products/'.$product->id.'/edit')}}">

Editar

</a>

@include('products.delete',['product' => $product])

</div>

@endif

El carrito de compras no esta enlazado con un usuario esta con la sesión, aunque no este registrado se guarda un identificador de un carrito de compras en tu computadora datos de navegador, esto nos permite puede egreagr productos isn pasar un registro, cuando vendemos debemos quitar todos las dificultades y dejar solo los pasos necesarios, no formulario de varios campos, diseñamos la base de datos para eso utilizamos unamigracion.

Ejecutamos el comando

Php artisan make:migration create\_shopping\_carts\_table

Y tambien creamos nuestro modelo

Php artisan make:model ShoppingCart

En la migración

Schema::create('shopping\_carts',function(Blueprint $table){

$table->increments("id");

$table->string('status');

$table->timestamps();

});

public function down()

{

Schema::drop('shopping\_carts');

}

Y ejecutamos

php artisan migrate

para crear el modelo

en nuestro modelo

los métodos integradores – integrar funcionalidad de otros, y método que realizan las cosas

buena practica los nombres deben escribir lo que quieren hacer del método

en los carritos de compras muestra cuando el usuario entre nosotros mostramos el carrito que ya tenia o si no tiene uno crearle uno.

public static function findOrCreateBySessionID($shopping\_cart\_id){

if($shopping\_cart\_id)

return ShoppingCart::findBySession($shopping\_cart\_id);

else

return ShoppingCart::createWithoutSession($shopping\_cart\_id);

}

Verifica si existe el carrito si no crea uno

public static function createWithoutSession(){

return ShoppingCart::create([

"status" => "incompleted"

]);

}

Permite crear un carrito con un parámetro con la función create

Vamos a MainController en el cual al inicio crearemos un carrito de compra

// primero llamara buscara es su session el id del carrtio de compras

$shopping\_cart\_id = \Session::get('shopping\_cart\_id');

//la primera vez no existe asi que dara nullo pero si ya estas logueado retomara su carrito con su id para eso llama a la funcion findOrCreateBySessionID

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);

// aqui si tomra el id del carrito

\Session::put("$shopping\_cart\_id", $shopping\_cart->id);

return view('main.home',["shopping\_cart" => $shopping\_cart]);Agregamos importamos el modelo

use App\ShoppingCart;

En nuestro Model

Método estatico es llamado desde la clase y un método dinamico lo llamas desde el objeto

Vamos a nuestro resources/views/layouts/app.blade.php

Agregamos

<li>

<a href="">

Mi carrito

<span class="circle-shopping-cart">

{{$shopping\_cart->productsSize()}}

</span>

</a>

</li>

El carrito de compras tiene un incomveninete y es que no esta disponible en las otras paginas.

Para ello utilizamos un viewComposer via un provider

Para ello ejecutamos el comando

Php artisan make:provider ShoppingCartProvider

El archivo se crea en app/Providers

Función boot() - para provider boot se ejecuta

Function register()

Viewcomposer permite injectar variables en las vistas primero es a a que vistas se injecta las vistas

use App\ShoppingCart; no olvidarse del modelo

public function boot()

{

//

// tomara todos lov views

View()->composer("\*",function($view){

$shopping\_cart\_id = \Session::get('shopping\_cart\_id');

//la primera vez no existe asi que dara nullo pero si ya estas logueado retomara su carrito con su id para eso llama a la funcion findOrCreateBySessionID

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);

// aqui si tomra el id del carrito

\Session::put("$shopping\_cart\_id", $shopping\_cart->id);

// enviar la variable a los view

$view->with("shopping\_cart", $shopping\_cart);

});

}

Lo que necesita ahora es que laravel se entere de la existencia de este provider

Ir a Config/app

A la sección de providers

Ya agregar nuestra dirección App\Providers\ShoppingCartProvider::class;

Cuando uno tiene producto y carrito de compra necesita una tabla intermedia

Uno que haga relación con y una con otra

Primero ejecutamos el comando

Php artisan make:migration create\_in\_shopping\_carts\_tables

Y su modelo

Php artisan make:model InShoppingCart

Están en database/migrations

Y en App respectivamente

Agregamos

public function up()

{

//

Schema::create('in\_shopping\_carts',function(Blueprint $table){

$table->increments("id");

$table->integer('product\_id')->unsigned();

$table->integer('shopping\_cart\_id')->unsigned();

$table->foreign("product\_id")->references("id")->on("products");

$table->foreign("shopping\_cart\_id")->references("id")->on("shopping\_carts");

$table->timestamps();

});

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

//

Schema::drop('in\_shopping\_carts');

}

Y ejecutamos todo

Php artisan migrate

En Shopping CArt agregamos

public function inShoppingCarts(){

return $this->hasMany("App\InShoppingCart");

}

Para que nos regrese todos los registro de la tabla in shopping cart

public function products(){

return $this->belongsToMany('App\Product','in\_shopping\_carts');

}

Y primero damos el modelo y luego la tabla donde se hace la realacio

Creamos el controller del carrito de compras tipo resource ejecutamos

Php artisan make:controller InShoppingCartsController –resource

Al crear el controller automaticamente creamos todos lo metodos por default pero Habra ocasiones en las que solo querremos aplicar unos cuantos, en la sección carrito de compras solo utilizaremos destroy y store

Vamos a routes/web.php

Route::resource('in\_shopping\_carts','InShoppingCartsController'[

'only' => ['store', 'destroy']

]);

El recursos o tabla será nuestra tabla intermediaria, segundo el controller, y el tercero serán los métodos que utilizaremos

Agregar a productos a un carrito de compra

Primero creamos un formulario para en nuestra tabla view creamos un folde en Resources/views/ in\_shopping\_carts y creamos el archivo form.blade.php

Agregamos – la url es donde enviaremos , el metodo para enviar tiene que se rpost

{!! Form::open(['url' => '/in\_shopping\_carts', 'method' => 'POST', "class" => "inline-block" ]) !!}

<input type="hidden" name="product\_id" value="{{$product->id}}">

<input type="submit" value="Agregar al carrito" class="btn btn-info">

{!! Form::close() !!}

Despues vamos a view/products/show.blade.show

E incluimos en una seccion

@include("in\_shopping\_carts.form",["product" => $product])

Tomara el producto

Ahora nos vamos al controller InShoppingCartsController

En la función store ()

$shopping\_cart\_id = \Session::get('shopping\_cart\_id');

//la primera vez no existe asi que dara nullo pero si ya estas logueado retomara su carrito con su id para eso llama a la funcion findOrCreateBySessionID

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);

$response = InShoppingCart::create([

"shopping\_cart\_id" => $shopping\_cart->id,

"product\_id" => $request->product\_id

]);

if($response){

return redirect('/carrito');

}else{

return back();

}

NOTA CCUANDO UTILIZAMOS EL METODO CREATE Y PASAMO UN HASH DE DATOS, TENEMOS QUE COLOCAR UN CAMPO PROTECTED EN NUESTRO MODELO LLAMADO FILLABLE EL CAMPO NOS INDICA LOS CAMPOS UE PODEMOS ALMACENAR EN MASA PARA QUE EL CREATE FUNCION SE DEBEN COLOCAR ACA ARA QUE NO SE LLENER CAMBIOS QUE NO QUIERAS QUE SE LLENEN POR SEGURIDAD

Agregamos en nues modelo InShoppingCart

protected $fillable = ["product\_id", "shopping\_cart\_id"];

creamos un controller para el carrito de compra

cremoas el controller con el comando

php artisan make:controller ShoppingCartsController

y en nuestro controller agregamos la función del código de provider del carrito

public function index(){

$shopping\_cart\_id = \Session::get('shopping\_cart\_id');

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);

//retorna todos los productos del carrito

$products = $shopping\_cart->products()->get();

// el manejo de la inforamcion siempre tiene que estar en el modelo no aqui

// $shopping\_cart->products()->sum("pricing");

// los controladores tiene que ser delgados y los modelos deben tener solo codigo gordos

$total = $shopping\_cart->total();

return view("shopping\_carts.index", ["products" => $products,"

total" => $total]);

}

Creamos una carpeta view/ shopping\_carts

Y creamos index.blade.php

Escribirmos

@extends("layout.app")

@section("content")

<hi>{{$total}}</hi>

@endsection

En routes/web.php

Agregamos

// no se crea como un ruta de resources para que se mas facil de recordar

Route::get('/carrito', 'ShoppingCartsController@index');

Y en layouts/app.blade.php

Se agrega

<li>

<a href="{{url('/carrito')}}">

Mi carrito

<span class="circle-shopping-cart">

{{$shopping\_cart->productsSize()}}

</span>

</a>

</li>

En shopping carts index.blade.php

@extends("layouts.app")

@section("content")

<div class="big-padding text-center blue-grey white-text">

<hi>{{$total}}</hi>

</div>

<div class="container">

<table class="table table-bordered">

<thead>

<tr>

<td>Producto</td>

<td>Precio</td>

</tr>

</thead>

<tbody>

@foreach($products as $product)

<tr>

<td>{{$product->title}}</td>

<td>{{$product->pricing}}</td>

</tr>

@endforeach

<tr>

<td>Total</td>

<td>{{$total}}</td>

</tr>

</tbody>

</table>

</div>

@endsection

Pagos en paypal

Ir a <https://developer.paypal.com/>

Tenemos que utilizar el api rest de paypal y una librería que a su vez intergra el sdk para comunicarnos con la api

Tenemos que tener una cuenta paypal la que utilizamos para el dinero

DashBoard/ My Apps & Credencials/ Create App / dale un nombre y debes seleccionar una cuenta de vendedor si no tienes hay que crear una

Para crear una cuenta ir a Accounts / seleccionar una cuneta Bussiness (Merchant Account) esta será una cuenta de prueba

Una vez creado tendremos nuestra cuente nuestro cliente Id

En composer.json ir a requiere

"anouar/paypalpayment": "~1.0"

Y luego composer update

Luego vamos a config/app.php

Ir a ala sección de provider

Anouar\Paypalpayment\PaypalpaymentServiceProvider::class,

Creamos un alias para llamar a la clase

‘PaypalPayment’ => Anouar\Paypalpayment\Facades\PaypalPayment::class,

Ahora tenemos que genrar un archive de configuración

Php artisan vendor:publish – se crea un paypal\_payment

Archivo para configuración paypal

El Endpoint indica donde se van a dirigir la peticiones

Descomenta el archivo de log ‘FileName’ => ‘../Paypal.log’,

En Account descomenta CliendID y Client Secret

Lo reemplazamos con nuestra información que creamos

Cobrar debemos contruir la petición bajo su formato todo debe estar perfectamente cordinado

En app/ creamos un archivo Paypal.php

<?php

namespace App;

class PayPal{

private $\_apiContext;

private $shopping\_cart;

private $\_ClientId = '';

private $\_ClientSecret = '';

public function \_\_contruct($shopping\_cart){

$this->\_apiContext = \PaypalPayment::ApiContext($this->\_ClientId

, $this->\_ClientSecret);

$config = config("paypal\_payment");

$flatConfig = array\_dot($config);

$this->\_apiContext->setConfig($flatConfig);

$this->shopping\_cart = $shopping\_cart;

}

}

Vamos a shoppingcartscontroller

Se cambia a public function index(){

$shopping\_cart\_id = \Session::get('shopping\_cart\_id');

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);

$paypal = new PayPal($shopping\_cart);

return "";

/\* $products = $shopping\_cart->products()->get();

$total = $shopping\_cart->total();

return view("shopping\_carts.index", ["products" => $products, "total" => $total ]);\*/

}

En paypal.php

<?php

namespace App;

class PayPal{

private $\_apiContext;

private $shopping\_cart;

private $\_ClientId = '';

private $\_ClientSecret = '';

public function \_\_construct($shopping\_cart){

// utilizamos el alias que creamos del prvodier

$this->\_apiContext = \PaypalPayment::ApiContext($this->\_ClientId

, $this->\_ClientSecret);

// utilizamos utilizamos el config

$config = config("paypal\_payment");

$flatConfig = array\_dot($config);

// utilizamos la configuracion flat porque el metodo asi lo espera

$this->\_apiContext->setConfig($flatConfig);

// utilizamos el carrito de compras

$this->shopping\_cart = $shopping\_cart;

}

public function generate(){

$payment = \PaypalPayment::payment()->setIntent("sale")

->setPayer($this->payer())

->setTransactions([$this->transaction()])

->setRedirectUrls($this->redirectURLs());

}

public function payer(){

// Return payment info

return \PaypalPayment::payer()->setPaymentMethod("paypal");

}

public function redirectURLs(){

//Returns transaccion info

$baseURL = url('url');

return \PaypalPayment::redirectUrls()

->setReturnUrl("$baseURL/payements/store");

->setCancelUrl("$baseURL/carrito");

}

public function transaction(){

return \PaypalPayment::transaction()

->setAmount($this->amount())

->setItemList($this->items())

->setDescription("Tu compra de ProductsFacilito")

->setInvoiceNumber(uniqid()))

}

public function items(){

items = [];

$products = $this->shopping\_cart->products()->get();

foreach ($products as $product) {

array\_push($items, $product->paypalItem());

}

return \PaypalPayment::itemList()->setItems($items);

}

public function amount(){

return \PaypalPayment::amount()->setCurrency("USD")

-setTotal($this->shopping\_cart->total());

}

}

Y Model Product

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Product extends Model

{

//

public function paypalItem(){

return \PaypalPayment::item()->\_\_setName($product->title)

->setDescription($product->description)

->setCurrency('USD')

->setQuantity(1)

->setPrice($this->pricing);

}

}

ShoppingCartsController

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;

use App\ShoppingCart;

use App\PayPal;

class ShoppingCartsController extends Controller

{

//

public function index(){

$shopping\_cart\_id = \Session::get('shopping\_cart\_id');

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);

$paypal = new PayPal($shopping\_cart);

$paypal->generate();

return "";

/\* $products = $shopping\_cart->products()->get();

$total = $shopping\_cart->total();

return view("shopping\_carts.index", ["products" => $products, "total" => $total ]);\*/

}

}

Cremoas una ruta route/web.php

Route::get('/payments/store', 'PaymentsController@store');

En la linea de commando ejecutamos

Php artisan make:controller PaymentsController

class PaymentsController extends Controller

{

//

public function store(Request $request){

$shopping\_cart\_id = \Session::get('shopping\_cart\_id');

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);

$paypal = new Paypal($shopping\_cart);

}

}

En el archivo paypal.php agregamos

public function execute(){

$payment = \PaypalPayment::getById();

}

El PaymentsController

public function store(Request $request){

$shopping\_cart\_id = \Session::get('shopping\_cart\_id');

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);

$paypal = new Paypal($shopping\_cart);

$paypal->execute($request->paymentId,$request->PayerID);

}

Paypal.php

public function execute($paymentId,$payerId){

$payment = \PaypalPayment::getById($paymentId,$this->\_apiContext);

$execution = \PaypalPayment::PaymentExecution()->setPayerId($payerId);

// quien lo va a pagar

return $payment->execute($execution,$this->\_apiContext);

}

paymentsController

public function store(Request $request){

$shopping\_cart\_id = \Session::get('shopping\_cart\_id');

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);

$paypal = new Paypal($shopping\_cart);

dd($paypal->execute($request->paymentId,$request->PayerID));

}

Agregar en ShoppingCart

public function totalUSD(){

return $this->products()->sum("pricing") / 100;

}

PaymentsController importar ShoppingCart y Paypal

Falta información del usuario

Creamos una nueva mgracion para las ordenes

Php artisan make:migration create\_orders\_table

En nuestra migracion agregamos

public function up()

{

//

Schema::create('orders',function(Blueprint $table){

$table->increments("id");

$table->integer("shopping\_cart\_id")->unsigned();

$table->foreign("shopping\_cart\_id")->references("id")->on("shopping\_carts");

$table->string("line1");

$table->string("line2")->nullable(true);

$table->string("city");

$table->string("postal\_code");

$table->string("country\_code");

$table->string("state");

$table->string("recipient\_name");

$table->string("email");

$table->string("status")->default("creado");

$table->string("guide\_number")->nullable(true);

$table->integer("total");

$table->timestamps();

});

}

Y ejecutamos php artisan migrate y creamos nuestro modelo

Php artisan make:model Order

Vamos a PaymentsController,

$reponse = $paypal->execute($request->paymentId,$request->PayerID);

if($response->state == "approved"){

$details = Order::createFromPayPalResponse($response,$shopping\_cart);

}

Agregamos para verificar los response si fueron corecctamente enviados

class PaymentsController extends Controller

{

//

public function store(Request $request){

$shopping\_cart\_id = \Session::get('shopping\_cart\_id');

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);

$paypal = new Paypal($shopping\_cart);

$reponse = $paypal->execute($request->paymentId,$request->PayerID);

if($response->state == "approved"){

$details = Order::createFromPayPalResponse($response,$shopping\_cart);

}

}

}

Agregamos a PaymentsController.php

En nuestro modelo Order

class Order extends Model, recupera la inforamcion del pagador y los coloca en u array, se agrega por separado el email porque se encuntra en otra sección de la información del payery guardamos el id del carrito

{

//

public static function createFromPayPalResponse($response,$shopping\_cart){

$payer = $response->payer;

$orderData = (array) $payer->payer\_info->shipping\_address;

$orderData["email"] => $payer->payer\_info->email;

$data["shopping\_cart\_id"] = $shopping\_cart->id;

Return Order::create($orderData);

}

}

Volvemos a PaymentsController debe quedar asi, cuando se utiliza un create se crea una variable fillable con las columnas que podemos almacenar via create

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Request;

use App\ShoppingCart;

use App\Paypal;

use App\Order;

class Order extends Model

{

//

protected $fillable = ['recipient\_name','line1','line2','city','country\_code','state','postal\_code','email','shopping\_cart\_id','status','total','guide\_number']

public static function createFromPayPalResponse($response,$shopping\_cart){

$payer = $response->payer;

$orderData = (array) $payer->payer\_info->shipping\_address;

$orderData = $orderData[key($orderData)];

$orderData["email"] => $payer->payer\_info->email;

$orderData["total"] => $shopping\_cart->total();

$orderData["shopping\_cart\_id"] = $shopping\_cart->id;

return Order::create($orderData);

}

}

Cuando agregamos un product se envía al paypal para pagar

En PaymentsController, cambiamos para ir a una vista

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Request;

use App\ShoppingCart;

use App\Paypal;

use App\Order;

class PaymentsController extends Controller

{

//

public function store(Request $request){

$shopping\_cart\_id = \Session::get('shopping\_cart\_id');

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);

$paypal = new Paypal($shopping\_cart);

$reponse = $paypal->execute($request->paymentId,$request->PayerID);

if($response->state == "approved"){

$order = Order::createFromPayPalResponse($response,$shopping\_cart);

}

//dd($order);

return view("shopping\_carts.completed",["shopping\_cart" => $shopping\_cart, "order" => $order]);

}

}

Agregamos un archivo resources/view/shopping\_carts/ se crea completed.blade.php para visualizer las ordenes

@extends("layouts.app")

@section("content")

<header class="big-padding text-center blue-grey white-text">

<h1>Compra completada</h1>

</header>

<div class="container">

<div class="card large-padding">

<h3>Tu pago fue procesado <span class="{{$order->status}}">{{$order->status}}</span></h3>

<p>Corrobora los detalles de tu envio:</p>

<div class="row">

<div class="col-xs-6">Correo</div>

<div class="col-xs-6">{{$order->email}}</div>

</div>

<div class="row">

<div class="col-xs-6">Direccion</div>

<div class="col-xs-6">{{$order->address()}}</div>

</div>

<div class="row">

<div class="col-xs-6">Codigo Postal</div>

<div class="col-xs-6">{{$order->postal\_code}}</div>

</div>

<div class="row">

<div class="col-xs-6">Ciudad</div>

<div class="col-xs-6">{{$order->city}}</div>

</div>

<div class="row">

<div class="col-xs-6">Estado y pais</div>

<div class="col-xs-6">{{ "$order->state $order->country\_code" }}</div>

</div>

<div class="text-center">

<a href="/{{$shopping\_cart->id}}">Link permanente de tu compra</a>

</div>

</div>

</div>

@endsection

Agregamos en order

public function address(){

return "$this->line1 $this->line2"

}

Vamos a shoppingCart agregamos 2 funciones para generar un id del carrito de compra

public function generateCustomID(){

}

public function updateCustomID(){

}

Crear una migración

Php artisan make:migration AddCustomIDColumnToShoppingCarts

Si se agrega <?php

use Illuminate\Support\Facades\Schema;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class AddCustomIDColumnToShoppingCarts extends Migration

{

/\*\*

\* Run the migrations.

\*

\* @return void

\*/

public function up()

{

//se agrega un unique que es un index para que query sean mas rapido, se recomienda se utilizan para buscar elemetnos solo cuanod se compre se genere

Schema::table('shopping\_carts',function($table){

$table->string('customid')->unique()->nullable();

});

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

//

}

}

Ejecutamos php artisan migrate

Md5() función de has no es bueno para password pero si para carrito, se utiliza el ID + ultima fecha de actualización y tiempo actual

ShoppingCart aumentamos

public function generateCustomID(){

return md5("$this->id $this->updated\_at");

}

public function updateCustomID(){

$this->customid = $this->generateCustomID();

$this->save();

}

En PaymentController

Se modifica

if($response->state == "approved"){

$order = Order::createFromPayPalResponse($response,$shopping\_cart);

$shopping\_cart->approve();

}

ShoppingCart se agrega

public function approve(){

$this->updateCustomIDAndStatus();

}

public function generateCustomID(){

return md5("$this->id $this->updated\_at");

}

public function updateCustomIDAndStatus(){

$this->status = "approved";

$this->customid = $this->generateCustomID();

$this->save();

}

UNa vez que tengamos nuestro link no consecutivo el carrito debería vaciarse para evitar

quese acumulen

En PaymentsContrellers.php

if($response->state == "approved"){

//esto nos permite desacernos del Id de la session del carrito

\Session::remove("shopping\_cart\_id");

$order = Order::createFromPayPalResponse($response,$shopping\_cart);

$shopping\_cart->approve();

}

En completed.blade.php

Modificamos para que muestre nuestro customid

<div class="text-center top-space">

<a href="{{url('/compras/'.$shopping\_cart->customid)}}">Link permanente de tu compra</a>

</div>

Creamos una nueva ruta en routes/web

Para mostrar los carrito (solo mostrar)

Route::resource('compras','ShoppingCartsController',[

'only' => ['show']

]);

En ShoppingCart

Agregamos la function

public function order(){

return $this->hasOne("App\Order");

}

En ShoppingCartsCOntroller

Agregamos

public function show($id){

$shopping\_cart = ShoppingCart::where('customid',$id)->first();

$order = $shopping\_cart->order();

return view("shopping\_carts.completed",["shopping\_cart" => $shopping\_cart, "order" => $order]);

}

EL administrador tenga un registro de las ordenes para actualizara visualizar ver cuales faltan en entregar y logros del mes

En la web, agregamos una nueva routa (Nota antes de crear la ruta se debe crear el controller)

Php artisan make:controller OrdersController --resource

Route::resources('orders','OrdersController',[

'only' => ['index','update']

]);

Una vez creada borrar las funciones que no necesitemos

Importamos

use App\Http\Requests;

use App\Order;

Metodos Scope – para crear un scope debe ser declarado como método de objeto o instanceia no de la clase y el principio debe ser scope, recibe un parámetro y es un query que representa la primera condición del objeto where()

Los método de clase - Order::latest();

Metodo de objeto – Order::where()->latest();

Agregamos en Order.php

public function scopeLatest($query){

return $query->orderID()->monthly();

}

public function scopeOrderID($query){

return $query->orderBy("id","desc");

}

Nota es preferible hacer la búsqueda por id (llame primaria -index)son mas rapidas no por created add

public function scopeMonthly($query){

return $query-> whereMonth ("created\_at","=",date('m'));

}

En OrdersCOntroller.php

public function index()

{

$orders = Order::latest()->get();

return view('orders.index',['orders' => $orders]);

}

En views creamos un folder orders y crear un index.blade.php

@extends("layouts.app")

@section("content")

<div class="container">

<div class="panel panel-default">

<div class="panel-heading">

<h2>Dashboard</h2>

</div>

<div class="panel-body">

<h3>Estadisticas</h3>

<h3>Ventas</h3>

<table class="table table-bordered">

<thead>

<tr>

<td>ID. venta</td>

<td>Comprador</td>

<td>Direccion</td>

<td>No. guia</td>

<td>Status</td>

<td>fecha de venta</td>

<td>Acciones</td>

</tr>

</thead>

<tbody>

@foreach($orders as $order)

<tr>

<td>{{$order->id}}</td>

<td>{{$order->recipient\_name}}</td>

<td>{{$order->address()}}</td>

<td>{{$order->guide\_number}}</td>

<td>{{$order->status}}</td>

<td>{{$order->created\_at}}</td>

<td>Acciones</td>

</tr>

@endforeach

</tbody>

</table>

</div>

</div>

</div>

@endsection

Visualiamos la lista de ordenes

ORder agregamos – para saber el total de las ordenes

public static function totalMonth(){

return Order::monthly()->sum("total");

}

public static function totalMonthCount(){

return Order::monthly()->count();

}

En OrdersCOntroller se modifica

public function index()

{

// get obtiene una lista de consultas

$orders = Order::latest()->get();

$totalMonth = Order::totalMonth();

$totalMonthCount = Order::totalMonthCount();

return view('orders.index',['orders' => $orders, "totalMonth" => $totalMonth,

"totalMonthCount" => $totalMonthCount]);

}

View/Ordes/index

Agregar

<div class="row top-space">

<div class="col-xs-4 col-md-3 col-lg-2 sale-data">

<span>{{$totalMonth}}USD</span>

Ingresos del mes

</div>

<div class="col-xs-4 col-md-3 col-lg-2 sale-data">

<span>{{$totalMonthCount}}</span>

Ingresos del mes

</div>

</div>

Para editar los elementos para el administrador en layouts agregamos las direcciones

<link href="//cdnjs.cloudflare.com/ajax/libs/x-editable/1.5.0/bootstrap3-editable/css/bootstrap-editable.css" rel="stylesheet"/>

<script src="//cdnjs.cloudflare.com/ajax/libs/x-editable/1.5.0/bootstrap3-editable/js/bootstrap-editable.min.js"></script>

En resources/assets/js/components/app.js

Borramos todo porque no utilizamo Vue.js

Debemos definir los valores globales inline

Agregamos

Método que se envía las peticiones

$.fn.editable.defaults.mode = 'inline';

$.fn.editable.defaults.ajaxOptions = {type: 'PUT'};

$(document).ready(function(){

});

Resources/views/orders/index.blade.php

Cambiamos a

@foreach($orders as $order)

<tr>

<td>{{$order->id}}</td>

<td>{{$order->recipient\_name}}</td>

<td>{{$order->address()}}</td>

<td>

<a href="#"

data-type="text"

data-pk="{{$order->id}}"

data-url="{{url("/orders/$order->id")}}"

data-title="Numero de guia"

data-value="{{$order->guide\_number}}"

class="set-guide-number"

data-name="guide\_number"></a>

</td>

<!-- <td>{{$order->guide\_number}}</td> -->

<td>

<a href="#"

data-type="select"

data-pk="{{$order->id}}"

data-url="{{url("/orders/$order->id")}}"

data-title="Numero de guia"

data-value="{{$order->guide\_number}}"

class="select-status"

data-name="status"></a>

</td>

<td>{{$order->created\_at}}</td>

<td>Acciones</td>

</tr>

@endforeach

Views/layouts/app.blade.php

<link href="{{ url('/css/app.css') }}" rel="stylesheet">

<script src="{{ url('/js/app.js') }}"></script>

Resources/assets/js/app.js

$.fn.editable.defaults.mode = 'inline';

$.fn.editable.defaults.ajaxOptions = {type: 'PUT'};

$(document).ready(function(){

$(".set-guide-number").editable();

$(".select-status").editable({

source: [

{value:"creado", text: "Creado"},

{value:"enviado", text: "Enviado"},

{value:"recibido", text: "Recibido"}

]

});

});

OrdersController

Se agrega a update

public function update(Request $request, $id)

{

//

$order = Order::find($id);

$field = $request->name;

$order->$field = $request->value;

$order->save();

return $order->$field;

}

EL token evita que las personas externar envien peticiones isn autorización

App/Http/Middleware/VerifyCsrfToken

protected $except = [

//hace que tengamos que validar todas las urls que tengan orders

'orders/\*'

];

Middleware

Debemos proteger la pagina products y orders solo debemos llegar a esas paginas cuando estemos autenticados

En orderController

Laravel puedes colocar middleware a nivel de las rutas o a nivel de los controladores

Este middleware ya esta creado y solo podemos llamar al constructor si estamos autenticados

Es parte del modulo del que laravel genera, cuando entrar si logueare te redirige a login

public function \_\_contruct(){

$this->middleware("auth");

}

Para productsController

Agregamos

public function \_\_contruct(){

$this->middleware("auth",["except" => "show"]);

}

El parámetro de except indica que sin autenticación solo podemos acces al product controller show

En shoppingscarts controller

Cambiamos

public function index(){

$shopping\_cart\_id = \Session::get('shopping\_cart\_id');

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);

/\* $paypal = new PayPal($shopping\_cart);

$payment = $paypal->generate();

return redirect($payment->getApprovalLink());\*/

//return "";

$products = $shopping\_cart->products()->get();

$total = $shopping\_cart->total();

return view("shopping\_carts.index", ["products" => $products, "total" => $total ]);

middleware nos puede ayudar para la busqueda del carrito de compras

en las secciones donde llamamos a nuestro carrito de compras debemos moverlos a un lugar en común, utilizamos el middlware

Un middleware es una función que se ejecuta en medio del ciclo donde llega la perticion del navegador y se enviar la respuesta, cuando se recibe una petición se ejecutar el middleware registrado el método y respuesta

creamos php artisan make:middleware BuildShoppingCar

una vez creado debemos tomar la logica de busqueda del carrito de compras

$shopping\_cart\_id = \Session::get('shopping\_cart\_id');

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);

use App\ShoppingCart; agregar el modelo del carrito

public function handle($request, Closure $next)

{

$shopping\_cart\_id = \Session::get('shopping\_cart\_id');

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);

//al carrito de compras lo colocamos en una variable request ya que los controladores tiene esta variable por default y al ponerlo en una variable request ya lo estamo poniendo a su disposicion

$request->shopping\_cart = $shopping\_cart;

return $next($request);

}

Vamos al archive http/kernel

Vamos a routemiddleware

Aqui podemos ver los middle y nombres y a las clases que llaman

agregamos

'shoppingcart' => \App\Http\Middleware\BuildShoppingCart::class,

En ShoppingCartsController

Borramos la parte del index que llamaba al carrito y ahora colocamos la función

public function \_\_contruct(){

$this->middleware("shoppingcart");

}

Y modificamos el index

Cuando este en el autenticador, envie el request con el carrito el index tomara como parametro el request con el carrito

public function index(Request $request){

$shopping\_cart = $request->shopping\_cart;

/\* $paypal = new PayPal($shopping\_cart);

$payment = $paypal->generate();

return redirect($payment->getApprovalLink());\*/

//return "";

$products = $shopping\_cart->products()->get();

$total = $shopping\_cart->total();

return view("shopping\_carts.index", ["products" => $products, "total" => $total ]);

}

Se debe copiar la lógica en los controller que utiliza

Modificamos InShoppingCarts

class InShoppingCartsController extends Controller

{

public function \_\_contruct(){

$this->middleware("shoppingcart");

}

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\Response

\*/

public function store(Request $request)

{

/\* $shopping\_cart\_id = \Session::get('shopping\_cart\_id');

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);\*/

$shopping\_cart = $request->shopping\_cart;

$response = InShoppingCart::create([

"shopping\_cart\_id" => $shopping\_cart->id,

"product\_id" => $request->product\_id

]);

if($response){

return redirect('/carrito');

}else{

return back();

}

}

/\*\*

\* Remove the specified resource from storage.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function destroy($id)

{

//

}

}

También en payments controllers modificamos

class PaymentsController extends Controller

{

//

public function \_\_contruct(){

$this->middleware("shoppingcart");

}

public function store(Request $request){

/\* $shopping\_cart\_id = \Session::get('shopping\_cart\_id');

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);\*/

$shopping\_cart = $request->shopping\_cart;

$paypal = new Paypal($shopping\_cart);

$response = $paypal->execute($request->paymentId,$request->PayerID);

if($response->state == "approved"){

//esto nos permite desacernos del Id de la session del carrito

\Session::remove("shopping\_cart\_id");

$order = Order::createFromPayPalResponse($response,$shopping\_cart);

$shopping\_cart->approve();

}

//dd($order);

return view("shopping\_carts.completed",["shopping\_cart" => $shopping\_cart, "order" => $order]);

}

}

Proteger registro de usuarios

Debemos crear un nuevo middleware

Como Wordpress el registro solo esta abierto si eres el primer usuario, una vez que ya exista un usuario el registro se cierra, osea si ya estas logueado se desactiva la sección de registro, a partir de allí solo yo como adminsitrador podre crear nuevos usuarios

Cremos un middleware para validar

Php artisan make:middleware ValidateFirstUserSignUp

Importamos

use App\User;

use Illuminate\Support\Facades\Auth;

el façade nos devul¿elves i ya hay un usuario que ya haya iniciado session o no y el modelo del usuario

En ValidateFIrstUserSignUp

Agregamo

public function handle($request, Closure $next)

{

$usersCount = User::count();

if($usersCount > 0 && !Auth::check()){

return redirect("/");

}

return $next($request);

}

}y en kernel

Agregamos

'first.user' => \App\Http\Middleware\ValidateFirstUserSignUp::class,

---------------

En RegisterController

Vamos a la sección de contructor

Solo permite crear cuenta cuando inicias sesión, una vez registrado y como único usuario ya puedes crear cuenta para otras personas

**Configuracion de servidor de correo**

En el archivo env

MAIL\_DRIVER=smtp

MAIL\_HOST=mailtrap.io

MAIL\_PORT=2525

MAIL\_USERNAME=null

MAIL\_PASSWORD=null

MAIL\_ENCRYPTION=null

Debemos poner los datos

Creamos un controller, cuando el usuario paqgue la orden que pago

Php artisan make:mail OrderCreated

Indicamos desde donde queremos enviar el correo

public function build()

{

return $this->from(“uriel@codigofacilito.com”)

->(‘mailers.order’);

}

Creamos en resoources una cerpeta mailers y orders.blade.php

<!DOCTYPE html>

<html>

<head>

<title></title>

<meta charset="utf-8">

</head>

<body>

<h1>Hola mundo</h1>

</body>

</html>

En ShoppingcartsCOntroller importamos la clase

use App\Mail\OrderCreated;

use Illuminate\Support\Facades\Mail;

facades permite utilizer el mailesr

en index

public function index(Request $request){

Mail::to("uriel@codigofacilito.com")->send();

$shopping\_cart = $request->shopping\_cart;

/\* $paypal = new PayPal($shopping\_cart);

$payment = $paypal->generate();

return redirect($payment->getApprovalLink());\*/

//return "";

$products = $shopping\_cart->products()->get();

$total = $shopping\_cart->total();

return view("shopping\_carts.index", ["products" => $products, "total" => $total ]);

}agremoas

En shopping cartcontroller la lógica del envio de correos la enviamos a order

Mail::to("carloscruzado89@gmail.com")->send(new OrderCreated());

En order agregamos

public function sendMail(){

Mail::to("carloscruzado89@gmail.com")->send(new OrderCreated());

}

Importamos

use Illuminate\Support\Facades\Mail;

use App\Mail\OrderCreated;

o podemos cambiar, para que envie directo al comprador

public function sendMail(){

Mail::to(this->email)->send(new OrderCreated());

}

Para pode poner toda la informacion de nuestros productos debemos ir OrderCreated

Modificamos el constructor

public function \_\_construct($order)

{

//

$this->order = $order;

$this->products = [];

}

Asi recibe la orden y creamos un array para que tenga los productos

También debemos hacer las variables public

public $order;

public $products;

para utilizarlars en otros métodos

En shopping carts controller

Debemos modificar el método index

Mail::to("carloscruzado89@gmail.com")->send(new OrderCreated($this));

Con un this esto tomara al objeto actual

Entramos al Order y agregamos un método

public function shopping\_cart(){

return $this->belongsTo('App\ShoppingCart');

}

Modificamos

public function \_\_construct($order)

{

//

$this->order = $order;

$this->products = $order->shopping\_cart()->products();

}

Para llamar a todos los products del carrito de nuestra order

En order.blade.php cambiamos

<!DOCTYPE html>

<html>

<head>

<title></title>

<meta charset="utf-8">

</head>

<body>

<h1>Hola mundo</h1>

<p>Te enviamos los datos de tu compra realizada den ProductsFacilito</p>

<table>

<thead>

<tr>

<th>Producto</th>

<th>Costo</th>

</tr>

</thead>

<tbody>

@foreach($products as $product)

<tr>

<td>{{$product->title}}</td>

<td>{{$product->pricing}}</td>

</tr>

@endforeach

<tr>

<td>Total</td>

<td>{{$order->total}}</td>

</tr>

</tbody>

</table>

</body>

</html>

El nombre total es tomado de array fill

Ahora en payments controller

En la función store

Agregamos

if($response->state == "approved"){

//esto nos permite desacernos del Id de la session del carrito

\Session::remove("shopping\_cart\_id");

$order = Order::createFromPayPalResponse($response,$shopping\_cart);

$shopping\_cart->approve();

$order->sendMail();

}

$order->sendMail(); que envia el mensaje cuando ya se ha aprobado la compra

En shopping carts controller regresamos la lógica de direccionamiento

public function index(Request $request){

//$mailer = new OrderCreated();

Mail::to("carloscruzado89@gmail.com")->send(new OrderCreated($this));

$shopping\_cart = $request->shopping\_cart;

$paypal = new PayPal($shopping\_cart);

$payment = $paypal->generate();

return redirect($payment->getApprovalLink());

//return "";

$products = $shopping\_cart->products()->get();

$total = $shopping\_cart->total();

return view("shopping\_carts.index", ["products" => $products, "total" => $total ]);

}

En OrderCreated

public function \_\_construct($order)

{

//

$this->order = $order;

$this->products = $order->shopping\_cart->products()->get();

}

Shopping carts controller

public function index(Request $request){

//$mailer = new OrderCreated();

$order = Order::all()->last();

Mail::to("carloscruzado89@gmail.com")->send(new OrderCreated($this));

/\* $shopping\_cart = $request->shopping\_cart;

$paypal = new PayPal($shopping\_cart);

$payment = $paypal->generate();\*/

return redirect($payment->getApprovalLink());

//return "";

$products = $shopping\_cart->products()->get();

$total = $shopping\_cart->total();

return view("shopping\_carts.index", ["products" => $products, "total" => $total ]);

}

El método all carga todos los datos de la carga del utlimo datos de la tabla pero no es muy recomendable cuandp se tiene muchos registros

Blueprints - puedes conseguirlo buscando email blueprints templates, puedes descargarlo

Tomamos un template y los reemmplazamos agregándole el formato blade php

En caso de que este descuadrado , vamos a prefrences / settings / syntax specific

{

“Word\_wrap”: “false”

}

Y guardamos, agregarmos el foreach

EN Order.php

Agregamos

public function shoppingCartID(){

return $this->shopping\_cart()->customid;

}

Para devolver el id del carrito

Nota algunos comentarios pueden causar problema ya que utilzian arroba eliminarlos

Como enviar mensaje de que su producto se ha actualizado

Vamos al modelo de la orden

Agregar el método en ORder

public function sendUpdateMail(){

Mail::to("carloscruzado89@gmail.com")->send(new OrderUpdated($this));

}

Para crear otro Mailer

Creamos php artisan make:mail OrderUpdated

En nuestro Order Updated agregamos

<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;

use Illuminate\Mail\Mailable;

use Illuminate\Queue\SerializesModels;

use Illuminate\Contracts\Queue\ShouldQueue;

class OrderUpdated extends Mailable

{

use Queueable, SerializesModels;

public $order;

/\*\*

\* Create a new message instance.

\*

\* @return void

\*/

public function \_\_construct($order)

{

//

$this->order = $order;

}

/\*\*

\* Build the message.

\*

\* @return $this

\*/

public function build()

{

$this->from("carloscruzado89@gmail.com")

->view('mailers.order\_updated');

}

}

En Order importamos orderUpdated,

use App\Mail\OrderUpdated;

Eventos de modelo – enviar cooreo cada de vez que se actualize tu orden

Ir a EventServiceProvider se registra cualquier evento de su aplicación

Imporatamos order

use App\Order;

y en aagremaos

public function boot()

{

parent::boot();

Order::created(function($order){

$order->sendMail();

});

Order::updated(function($order){

$order->sendUpdateMail();

});

//

}

Para enviar los correos cuanodo se cree un metodo de una orden

Aliminamos el envio de shopping controle y payments controller

Dejamos eñ indexa si

public function index(Request $request){

$shopping\_cart = $request->shopping\_cart;

$paypal = new PayPal($shopping\_cart);

$payment = $paypal->generate();

return redirect($payment->getApprovalLink());

//return "";

$products = $shopping\_cart->products()->get();

$total = $shopping\_cart->total();

return view("shopping\_carts.index", ["products" => $products, "total" => $total ]);

}

Dejamos

Paymets asi

public function store(Request $request){

/\* $shopping\_cart\_id = \Session::get('shopping\_cart\_id');

$shopping\_cart = ShoppingCart::findOrCreateBySessionID($shopping\_cart\_id);\*/

$shopping\_cart = $request->shopping\_cart;

$paypal = new Paypal($shopping\_cart);

$response = $paypal->execute($request->paymentId,$request->PayerID);

if($response->state == "approved"){

//esto nos permite desacernos del Id de la session del carrito

\Session::remove("shopping\_cart\_id");

$order = Order::createFromPayPalResponse($response,$shopping\_cart);

$shopping\_cart->approve();

}

//dd($order);

return view("shopping\_carts.completed",["shopping\_cart" => $shopping\_cart, "order" => $order]);

}

**Crear botón de compra**

Ir a ShoppingcartController – comentar el paypal que redirige

public function index(Request $request){

$shopping\_cart = $request->shopping\_cart;

// $paypal = new PayPal($shopping\_cart);

// $payment = $paypal->generate();

// return redirect($payment->getApprovalLink());

//return "";

$products = $shopping\_cart->products()->get();

$total = $shopping\_cart->total();

return view("shopping\_carts.index", ["products" => $products, "total" => $total ]);

}

Ir a resources/view/shoppingcarts/index.blade.php

{!!Form::open(['url' => '/carrito', 'method' => 'POST', 'class'=> 'inline-block']) !!}

<input type="submit" class="btn btn-success" value="Pagar">

{!!Form::close([]) !!}

Agregar en Shoppingcartscontroller

public function checkout(Request $request){

$shopping\_cart = $request->shopping\_cart;

$paypal = new PayPal($shopping\_cart);

$payment = $paypal->generate();

return redirect($payment->getApprovalLink());

}

En routes – web.php

Agregar

Route::post('/carrito', 'ShoppingCartsController@checkout');

Para agregar al carrito con post y checkout para que vaya a la function checkout

Toma el request

**Subir imágenes**

Ir a resources/View/products/form.blade.php

Agregar

<div class="form-group">

{{ Form::file('cover') }}

</div>

Y agregar la seccion files true para poder utilizar archivos

{!! Form::open(['url' => $url, 'method' => $method, 'files' => true]) !!}

Cover sera el nombre del archivo

ProductsController

En la función store

Agregamos

$hasFile = $request->hasFile('cover') && $request->cover->isValid();

Verifica si el archivo se pudo subir guardarlo en una capeta temporal

Debe quedar asi

public function store(Request $request)

{

//

$hasFile = $request->hasFile('cover') && $request->cover->isValid();

$product = new Product;

$product->title = $request->title;

$product->description = $request->description;

$product->pricing = $request->pricing;

$product->user\_id = Auth::user()->id;

if($hasFile){

$extension = $request->cover->extension();

$product->extension = $extension;

}

if($product->save()){

if($hasFile){

$request->cover->storeAs('images',"$product->id.$extension");

}

return redirect("/products");

}else{

return view("products.create",["product" => $product]);

};

}

La ruta es Storage/ app / images se creara

Creamos php artisan make:migration add\_column\_extension\_to\_products

Agregar

public function up()

{

//

Schema::table('products',function($table){

$table->string('extension')->nullable();

});

}

Y ejecutar php artisan migrate

Resources/view/products/index.blade.php

Agregar la parte para visualizar

<a href="{{url('/products/$product->id')}}">

Ver

</a>

Resources/view/products/show.blade.php

Agregar

<div class="col-sm-6 col-xs-12">

@if($product->extension)

<img src="">

@endif

</div>

En problema con en donde almacenamos las imagines es que no estan en la carpeta public, debemos desarrollar un método para acceder a esos archivos

Routes/web.php

Agregamos {filename} se agrega en la seccion function filename

Route::get('products/images/{filename}',function($filename){

$path = storage\_path("app/images/$filename");

if(!\File::exists($path)) abort(404);

$file = \File::get($path);

$type = \File::mimeType($path);

$response = Response::make($file,200);

$response->header("Content-Type",$type);

return $response;

});

View/products/products/show.blade.php

Agregar

@if($product->extension)

<img src="{{url("/products/images/$product->id.$product->extension")}}" class="product-avatar">

@endif

</div>

En resources/assets/sass/app.scss

Agregar

.product-avatar{

max-width: 100%;

}

La clase de la seccion de imagines que se ajustara al maxmo tamaño de la seccion

En app/Http/Controllers/Auth/MainCOntroller

Actualizamos – para obtener todos las imagenes

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;

use App\Product;

class MainController extends Controller{

public function home(){

$products = Product::all();

return view('main.home');

}

}

Vamos a resources/views/main/home.blade.php

Vamos a resources/views/products/show.blade.php

Debe quedar

@extends('layouts.app')

@section('content')

<div class="container text-center">

@include("products.product",["product" => $product])

</div>

@endsection

Y creamos en la carpeta products/product.blade.php

<div class="card product text-left">

@if(Auth::check() && $product->user\_id == Auth::user()->id)

<div class="absolute actions">

<a href="{{url('/products/'.$product->id.'/edit')}}">

Editar

</a>

@include('products.delete',['product' => $product])

</div>

@endif

<h1>{{$product->title}}</h1>

<div class="row">

<div class="col-sm-6 col-xs-12">

@if($product->extension)

<img src="{{url("/products/images/$product->id.$product->extension")}}" class="product-avatar">

@endif

</div>

<div class="col-sm-6 col-xs-12">

<p>

<strong>Descripcion</strong>

</p>

<p>

{{$product->description}}

</p>

<p>

@include("in\_shopping\_carts.form",["product" => $product])

</p>

</div>

</div>

</div>

En la carpeta views/main/home.blade.php

@extends('layouts.app')

@section('title', 'Productos Facilito')

@section('content')

<div class="container text-center products-container">

@foreach($products as $product)

@include("products.product",["product" => $product])

@endforeach

</div>

@endsection

MainController.php – agregamos la seccion para tener los productos

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;

use App\Product;

class MainController extends Controller{

public function home(){

$products = Product::all();

return view('main.home',["products" => $products]);

}

}

Resources/assets/sass/app.scss

Agregamos

.products-container{

.product{

margin-top: 1em;

}

}

**Paginacion**

Ir al modelo Product - agregar para ordenar

public function scopeLatest($query){

return $query->orderBy("id","desc");

}

En maincontroller

class MainController extends Controller{

public function home(){

$products = Product::latest()->get();

return view('main.home',["products" => $products]);

}

}

$products = Product::latest()->get();

Para que tengamos la paginacion cambiamos

$products = Product::latest()->simplePaginate(2);

Que nos permite controlar la paginación ya que laravel tiene el método

Ir a resources/assets/view/main/home.blade.php

@section('content')

<div class="container text-center products-container">

@foreach($products as $product)

@include("products.product",["product" => $product])

@endforeach

<div>

{{$products->links()}}

</div>

</div>

@endsectionPara agregar la seccion que permite cambiar de pagina

Para agregar php artisan vendor:publish --tag=laravel-pagination

Los creara en viewS/vendor/simple-default.blade.php

Aquí podemos modificar el diseño del pagination

**AJAX**

Cambiar para que pueda ser utilizado Ajax en vez con un refhesr para hace petición completa al servidor

Resoruces/views/in\_shopping\_carts/form.blade.php – agregar una clase

{!! Form::open(['url' => '/in\_shopping\_carts', 'class' => 'add-to-cart', 'method' => 'POST', "class" => "inline-block" ]) !!}

<input type="hidden" name="product\_id" value="{{$product->id}}">

<input type="submit" value="Agregar al carrito" class="btn btn-info">

{!! Form::close() !!}

Ir a Resources/assets/js/app.js

Agregar

$(".add-to-cart").on("submit",function(ev){

ev.preventDefault();

var $form = $(this);

var $button = $form.find("[type='submit']");

$.ajax({

url: $form.attr("action"),

method: $form.attr("method"),

data: $form.serialize(),

dataType: "JSON",

beforeSend: function(){

$button.val("Cargando...");

},

success: function(){

$button.css("background-color","#00c853").val("Agregado");

//actualize el numero de carrito

$(".circle-shopping-cart").html(data.products\_count);

setTimeout(function(){

restartButton($button);

},2000);

},

error: function(err){

console.log(err);

$button.css("background-color","#d50000").val("Hubo un error.");

setTimeout(function(){

restartButton($button);

},2000);

}

});

return false;

});

function restartButton($button){

$button.val("Agregar al carrito").attr("style","");

}

Ir a InShoppingCartsController

if($request->ajax()){

return response()->json([

'products\_count' => InShoppingCart::productsCount($shopping\_cart\_id)

]);

}

Verifica si es ajax

Vamos a InShoppingCart

Agregamos

public static function productsCount($shopping\_cart\_id){

return InShoppingCart::where("shopping\_cart\_id",$shopping\_cart\_id)

->count();

}

Para contar los productos en el carrito

Ir a Resources/views/layouts/app.blade

<span class="circle-shopping-cart hightlight">

{{$productsCount}}

</span>

Ir a resources/assets/sass

Agregar

.highlight{

-webkit-animation: hightlight 1.5s;

animation: hightlight 1.5s;

}

@-webkit-keyframes hightlight {

50% {

background-color: #ffff00;

color:black;

}

100% {

background-color: inherit;

}

}

@keyframes hightlight {

50% {

background-color: #ffff00;

color: black;

}

100% {

background-color: inherit;

}

}

.circle-shopping-cart{

background-color: #232980;

border-radius:50%;

padding:1em 1em;

}

En resources/assets/js/app.js

Modificar para que ejecute la niacion

$(".circle-shopping-cart").html(data.products\_count).addClass("highlight");

Y cambiar para remove la clase

function restartButton($button){

$button.val("Agregar al carrito").attr("style","");

$(".circle-shopping-cart").removeClass("highlight");

}

Digital ocean

<http://www.putty.org/>

cliente para conectarse va ssh

putty.exe

indicamos el host del digital ocean

de preferencia el puerto 22

el ususario es root

el unix password es del correo

sudo apt –get update

sudo add-apt-repository ppa:ondrej/php

sudo apt –get update

sudo apt –get install php7.0-fpm

sudo apt-get install php-mbstring

sudo apt-get install php7.0-curl

sudo apt-get install php-xml

sudo apt-get install git

buscar el archive de configuracion php

cd /etc/php/ls cd 7.0 cd fpm

ls php.ini

nano php.ini

sudo nano php.ini

busca fix\_pathinfo

descomentamos y cambiamos a cgi.fix\_pathinfo=0 medida de seguridad

para git utiliza gitlab

ejecutamos git init en la carpeta del proyecto, crear un gitgnore

git add .

git commit –m “init”

git remote add origin git\_\_\_\_\_ el nombre de la dirección

git push origin master

y ya esta en nuestro repositorio

en tu sevidor remoto git clone la dirección del repositorio

en la carpeta que bajamos debemos instalar composer

en curl –sS <https://getcomposer.org/installer> | php

sudp mv composer.phar /usr/local/bin/composer

ve a tu carpeta de proyecto y exibimos

también apt-get install unzip

composer install

**instalar maria db**

agregar la llave publica utilizado por el manejador de paquetes se encuentraen la sección de comunidad

luego agregar el repositorio de mariadb

actualizar apt get

instalamos maria db sudo apt-get install mariadb-server

colorcar su password

ahora nos logueamos como mysql

mysql –u root –p

le damos un password

Creamos la base datos

CREATE DATABASE productos;

**Configurar las variables de entorno**

Vamos a las variables de entorno

Al archivo env.

Cambiar

APP\_KEY

APP\_ENV

Modificar la base de datos, el root,

O modificar APP\_URL= de su dominio

En terminal nano .env

Lo copiado en el archivo .env se peqga aquí

Y enter

Para verificar que todo este bien ejeuctamos las migracione

Php artisan migrate, si no pudo encontra el driver que conecta php con sql

Sudo apt-get install php7-mysql

Sudo service php7.0-fpm restart reiniciar el php

Php artisan migrate

**Configurar Nginx**

Es para recibir las peticiones para eso necesita que se configure

Cd /etc/nginx/

Cd sites-enabled

Sudo nano default

Con ctrl k borrar las linesas que están comentdas

La sección

Server {} es el seridor por default

Modifica el root donde esta el proyecto ecommerce/public

Y el index.html

Server\_name localhost o el nombre de tu dominio

Location /{

Try\_files $uri $uri/ /index.php?query\_string;

Fastcgi\_split\_path\_info (.+\.php)(/.+)$;

Fastcgi\_pass unix:/var/run/php/php7.0-fpm.sock;

Fastcgi\_index index.php;

Fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;

Include fastcgi\_params;

}

Sudo nano default

Sudo nginx restart

Para permisos

Is al proyecto cd /var

Mkdir www

Mv /home/ecommerce-curso/ /var/www/ecommerce-curso para mover

En www ya esta el proyecto

Sudo nano /etc/nginx/sites-enabled/default

Cambiamos el root a

/var/www/ecommerce-curso/public

Reiniciamos nginx sudo service nginx restart

Para cambiar el propietario de un archivo o un folder

Sudo chown –R “$USER”:www-data ./ parte de nginx

Sudo chmod –R 755 public/

Sudo chmod –R 755 storage/

Sudo chmod –R 755 bootstrap/

**Como subir nuestra aplicacion a un hosting compartido**

Si ya tienes un hosting compartido

Cpanel usuario y password

Ve a FTP accounts

Crear los usuarios - una vez hecho eso

Ir a FileZilla-project.org

Descarga filezilla cliente, instala tu versión

Colocamos la url del servidor nombre y usuario

Creamos una carpeta fuera de los folder nuevo directorio para almacenar todo de nuestro proyecto, evitar tomar el .DSStore .env .env.example .gitattributes . gitgnore composer.lock para evitar el exceso de carga y .git

Va a demorar

**Subir y configurar la carpeta public**

Toda la capeta de nuestro proyecto public se va a moder a la carpeta public de nuestro servidor html, por eso no es necesario hacer public a loa anterior una vez movido ,

Vamos al proyecto

Public index.php

Debemos redirigir a nuestro proyecto en nuestro sevidor remoto donde esta el pryecto public principal

**Para configurar nuestra base de datos**

Vamos a nuestra base de datos local phpadmin a products

Exportamos nuestra base de datos

En cpanel vamos al home ir a phpmyadmin y nos envía a base de datos de nuestro servidor remoto, antes debes crear MYsql databases

Borra la database pro default y el usuario

Creamos la ase de datos y el usuario

Le damos permisos

Ahora entramos a phpadmin de nuestro servidor remoto e importamos el archivo que esxporatamos ir a operaciones vaciar todo truncate de las tablas que se necesiten vacias, hosting conpartido no va a funcionar env.

Para ello vamos a config app.php y database.php

En app.php

Ir a la sección key

Para ello tomas la clave APP\_KEY de tu proyecto configurado al servidor de env.

Y la pegar

‘key’ => env(‘APP\_KEY’,’la clave APP\_KEY’)

Guardamos y en database

Ir a mysql colocamos los datos que creamos de nuestra base de datos el usuario que creamos en cpanel y se guarda y se sube via filezila

Y al app.php en ‘env’ cambiar a local

Para poder ver errores cambiamos el appdebug de app.php a true

**Obtener credenciales producción paypal**

Developer payapal ir a dashboard

Para cambiar nuestra api de sandbox a live

Copiamos el sandbox account client id y el secret y los reemplazamos en nuestro poyecto

Los reemplazamos en config / paypalpayment

Y cambiamos el endpoint

<https://api.paypal.com>

y en ap Paypal.php también cambiamos los subimos

**hosting compartido** ir a filezila y mover el archivo donde hicieron los cambios

si hay error

en nuestro proyecto ir a

‘mode’ => ‘live’,

**Hosting cloud**

Vamos al servidor de nginx sudo nano default

Agregamos un nuevo location pero para los archivos normales

Colocamos

Location /{

Try\_files $uri $uri/ /index.php?query\_string;

}

Al otro hacemos un cambio

Location ~ \.php$

los que parecen php

una vez hecho sudo service nginx restart

una vez git commit –am “SE cambio a producción”

al git

agregamos a paypal\_payment.php de nuestro proyecto

‘mode’ => ‘live’,

Y hacerle el push para subir los cambios