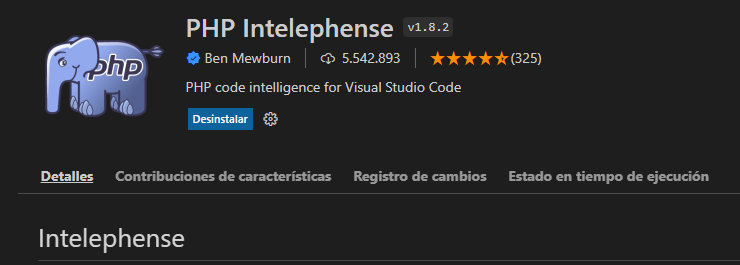
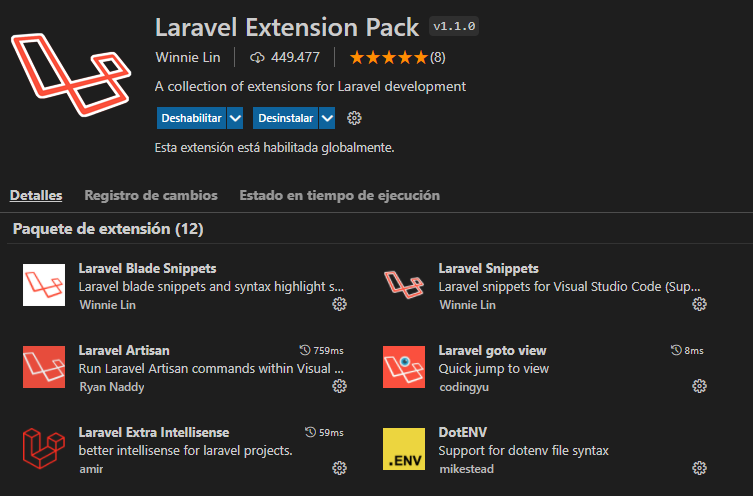
**LARAVEL APP CMS - JETSTREAM - ALPINE**

1. **COMPLEMENTOS PARA VISUAL CODE**

****

****

**Tailwind Css IntelliSense**

****

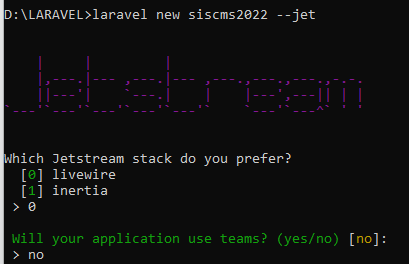
1. **CREAMOS EL PROYECTO**

Es necesario **composer**, **node** y **composer global require laravel/installer**

Creamos el proyecto: **laravel new siscms2022 --jet**

opción 0: **livewire**

trabajar con equipos (teams): **no**

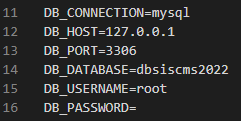


Luego de crear el proyecto ejecutamos: **npm install && npm run dev**

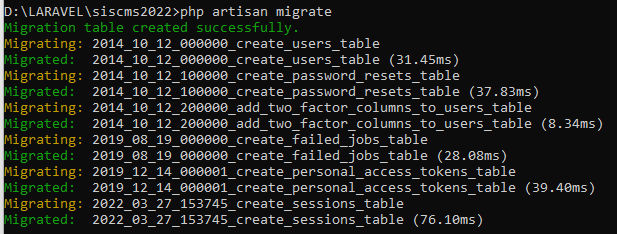
Accedemos a la ruta del proyecto: **cd siscms2022**

Abrimos con visual code: **code .**

Creamos la base de datos con el nombre correspondiente en **.env**



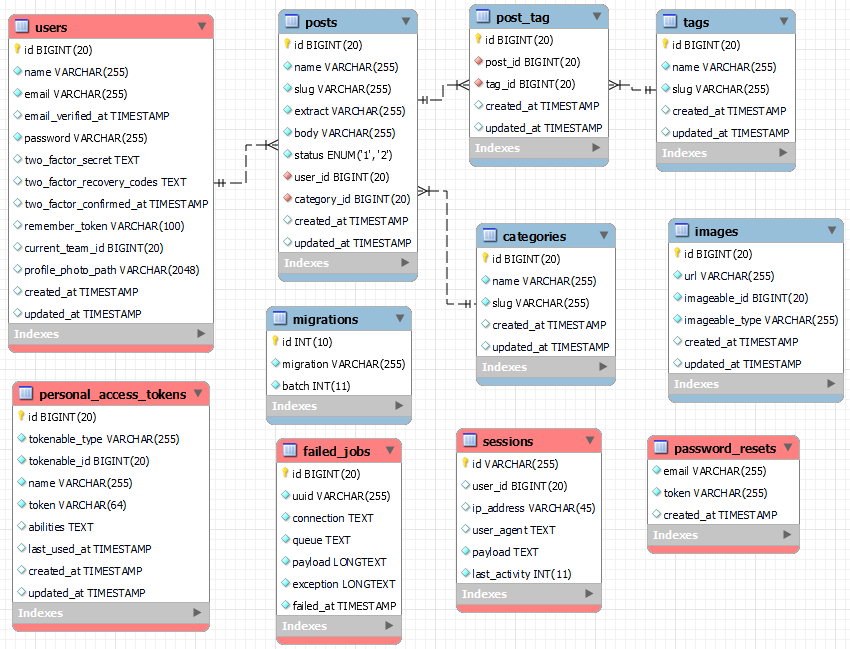
Finalmente realizamos las migraciones: **php artisan migrate**



Levanta el proyecto con: **php artisan serve**

1. **MODELO FÍSICO DE LA BASE DE DATOS**

Las tablas rojas son propias de laravel jetstream, las tablas azules debemos implementarlas usando migraciones.



Creamos las migraciones:

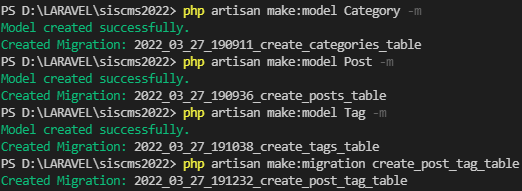
**php artisan make:model Category -m**

**php artisan make:model Post -m**

**php artisan make:model Tag -m**

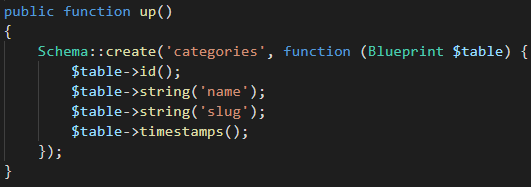
**php artisan make:migration create\_post\_tag\_table**

**php artisan make:model Image -m**

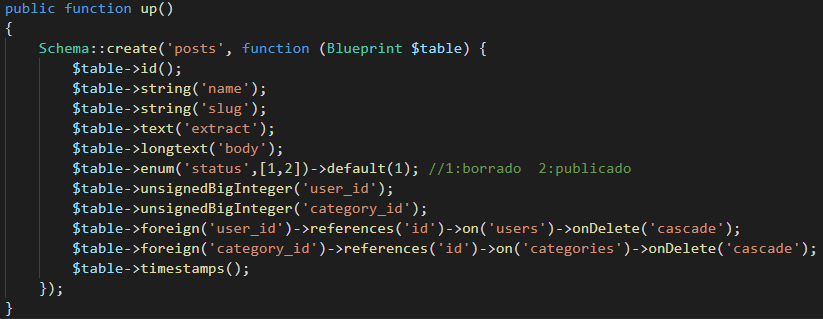


-m: Permite crear un modelo con su respectiva migración.

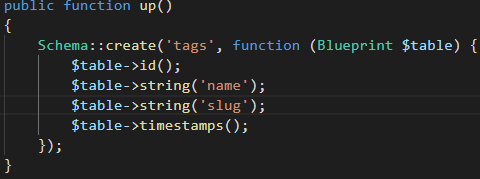
Abrimos el archivo: database/migrations/2022\_XX\_XX\_create\_categories\_table.php

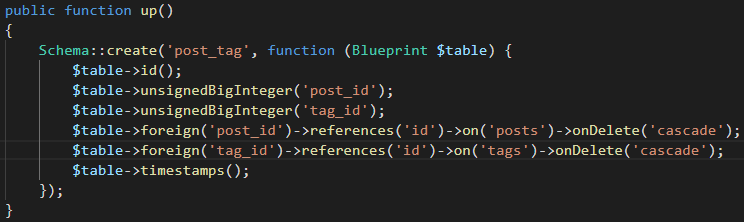


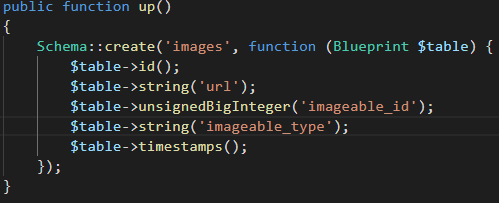
Codificamos las siguientes migraciones:

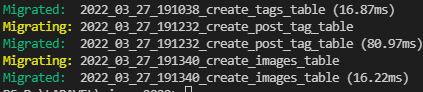


| public function up()  {  Schema::create('posts', function (Blueprint $table) {  $table->id();  $table->string('name');  $table->string('slug');  $table->text('extract');  $table->longtext('body');  $table->enum('status',[1,2])->default(1); //1:borrado 2:publicado  $table->unsignedBigInteger('user\_id');  $table->unsignedBigInteger('category\_id');  $table->foreign('user\_id')->references('id')->on('users')->onDelete('cascade');  $table->foreign('category\_id')->references('id')->on('categories')->onDelete('cascade');  $table->timestamps();  });  } |
| --- |



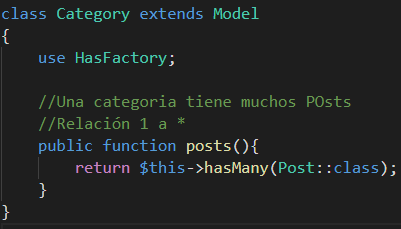




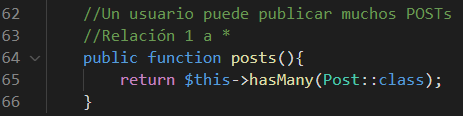
Finalmente ejecutamos: **php artisan migrate**

1. Relaciones entre modelos

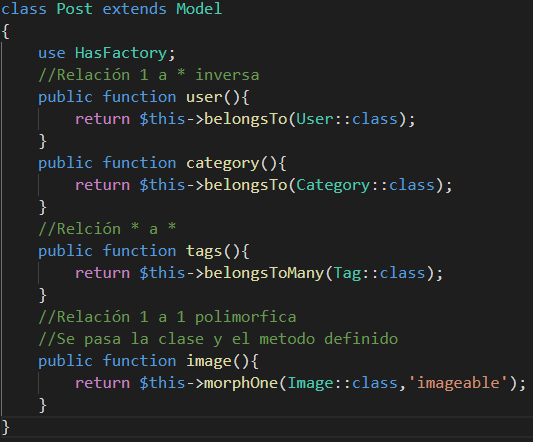
**app/Models/Category.php**

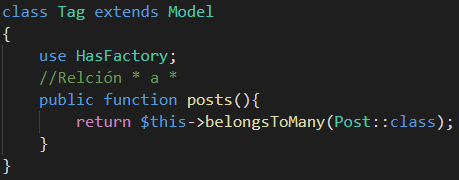


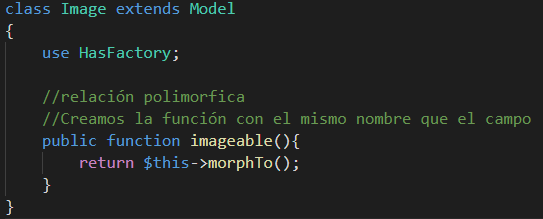
**app/Models/User.php**



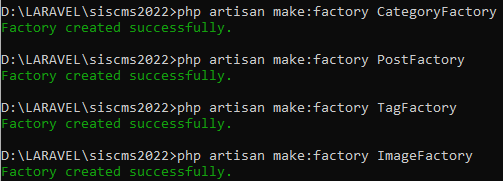
**app/Models/Post.php**





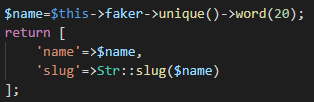


1. **Creamos Factories**



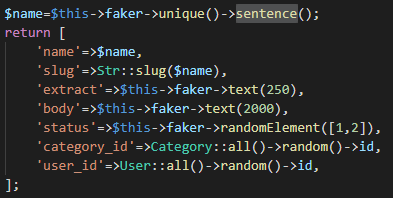
Se crean en database/factories/…

**CategoryFactory**

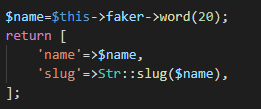


**use Illuminate\Support\Str;**

**PostFactory**



**TagFactory**

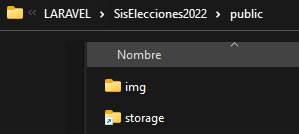


Cambiar el “local” por “public” en **config/filesystems.php**

esto para que las imágenes se suban a la ruta storage/app/public



**Es necesario el link a storage si este no se encuentra: php artisan storage:link**

****

**ImageFactory**



parámetros: ruta de alojamiento, ancho, alto, categoría, forma de almacenamiento

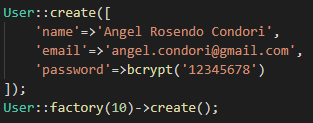
true: public/storage/posts/nombre\_imagen.jpg

false: nombre\_imagen.jpg

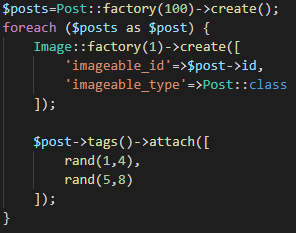
Para ejecutar los factories lo llamamos desde: **database/seeders/DatabaseSeeder.php**

Primero creamos dos seeder:

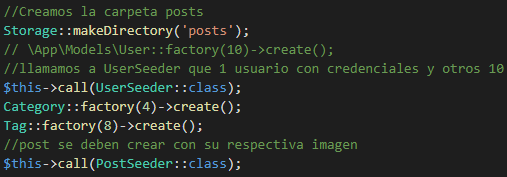




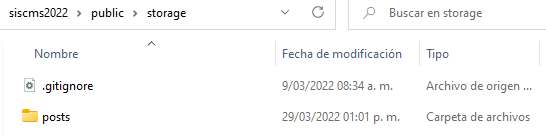




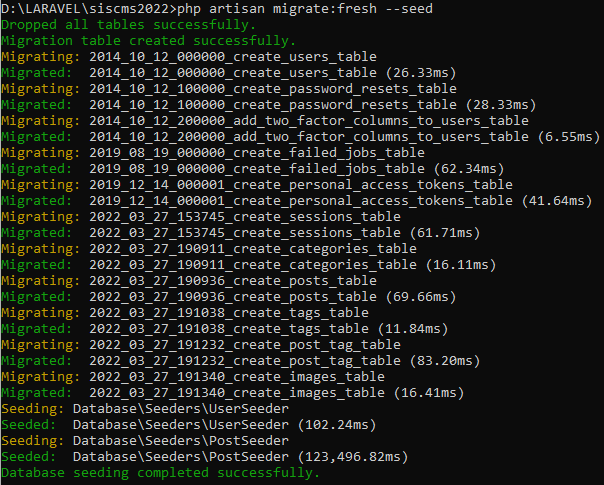
En **DatabaseSeeder.php**



Creamos la carpeta posts donde se generan las imágenes



Finalmente ejecutamos el comando: **php artisan migrate:fresh --seed**



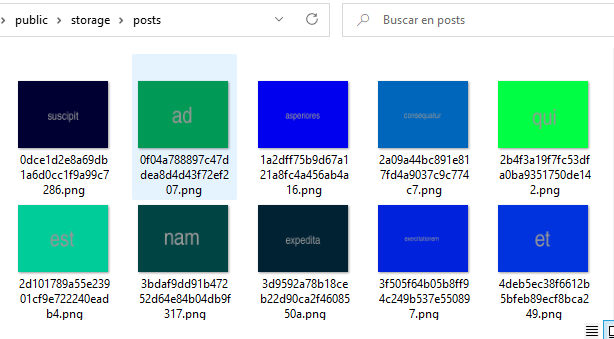
php artisan migrate:fresh --seed, es una abreviación de:

**php artisan migrate:fresh**

**php artisan db:seed**

**Este último comando reemplaza a los anteriores**

**php artisan migrate:fresh --seed**

****

El servicio de <http://lorempixel.com> que actualmente está inactivo y Faker usa este sitio lorempixel para generar las imágenes.

Para corregir este error se puede cambiar la base url del servidor de imagenes

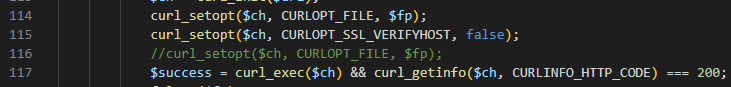
En la ruta: **vendor\fakerphp\faker\src\Faker\Provider\Image.php**

En la línea 13



public const BASE\_URL = 'https://placehold.jp';

En la línea 144



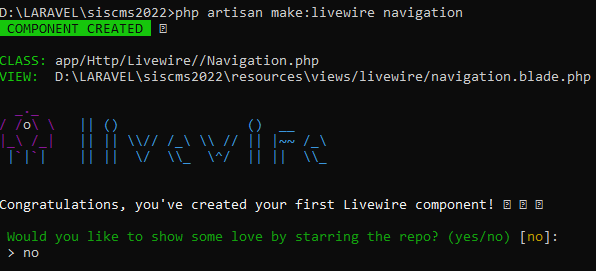
| curl\_setopt($ch, CURLOPT\_FILE, $fp);  **curl\_setopt($ch, CURLOPT\_SSL\_VERIFYHOST, false);** |
| --- |

1. **Tailwind**: <https://tailwindui.com/preview>

Framework CSS, en el enlace se puede revisar componentes gratuitos que tenemos a disposición.

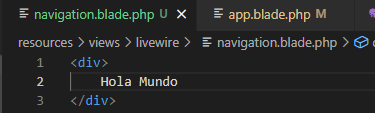
Creamos un componente para la barra de navegación:

**php artisan make:livewire navigation**

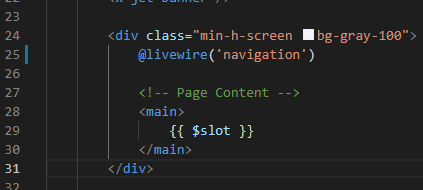
****

Usamos el componente:

views/livewire/navigation.blade.php

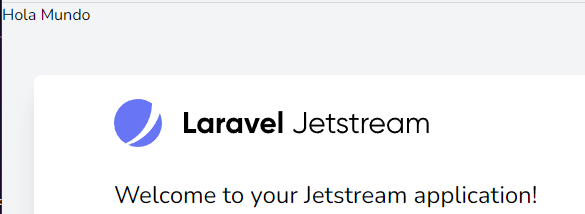


Realizar los cambios en **views/layouts/app.blade.php**



Iniciar sesión con el usuario creado en los factories

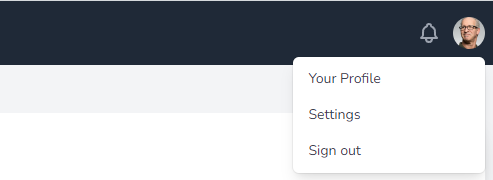
<https://tailwindui.com/components/application-ui/navigation/navbars>



Reemplazar el código por el menú disponible en tailwindcss



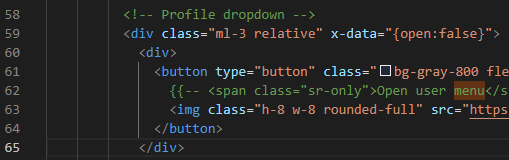
* Copiar código html y pegar en navigation.blade.php
* Revisar y adecuar el código para que se muestre correctamente.
* Para darle funcionalidad a los botones se puede usar alpine.js que viene incorporado en jetstream

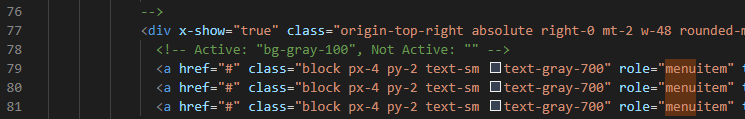


Ubicamos el div que contiene al dropdown para agregarle la funcionalidad mediante Alpine.js

<https://alpinejs.dev/>

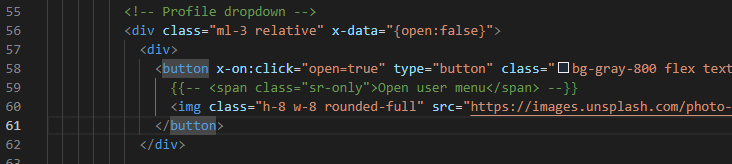
<https://javifont.medium.com/alpinejs-una-alternativa-ligera-a-vue-3a39f3802c5d>

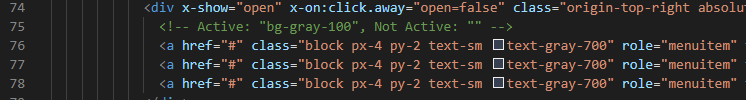


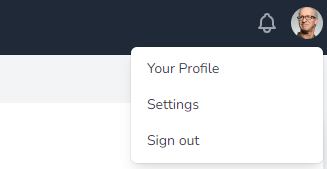


A cambiar true, por false se cierra y abre el menú

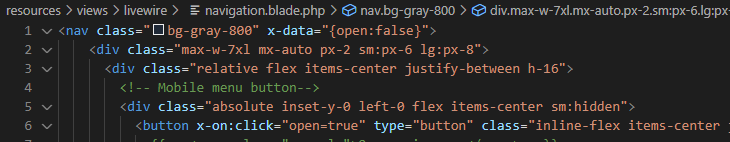
entonces pasamos el valor por variable con el evento click en el botón

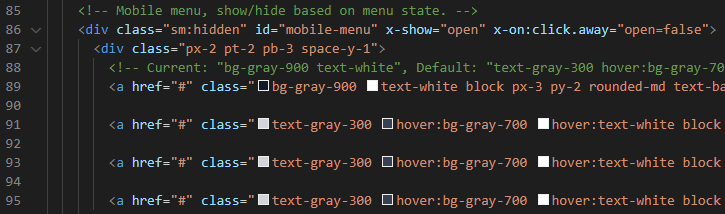






Realizamos lo mismo para el menú en caso de móviles



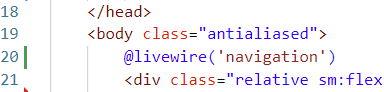


1. **Web pública (index)**

Creamos un controlador: **php artisan make:controller PostController**

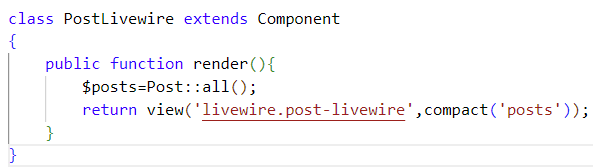
Creamos un componente: **php artisan make:livewire PostLivewire**

En el archivo de welcome.blade.php, realizamos los cambios para incluir el componente navigation



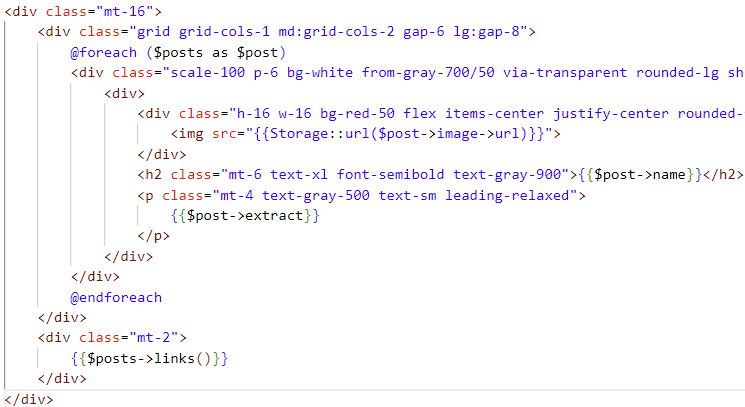
1. Mostrar los post en la web pública

http/Livewire/PostLivewire.php (Controlador)



Se puede agregar paginación cambiando all() por paginate()

resources/views/livewire/post-livewire.blade.php (vista)



El código se copia de la vista welcome.blade.php, agregando el @foreach

Pasamos las variables de los campos de la base de datos que queremos mostrar

Resultado

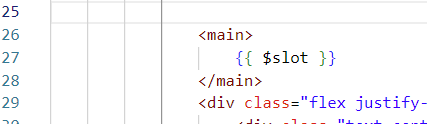


1. Creamos index layout

Para reutilizar las plantillas creamos un layout en resources/views/layouts

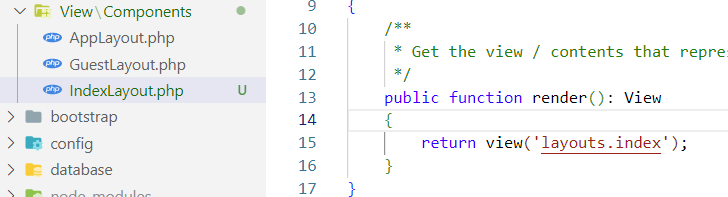
index.blade.php

Copiamos todo el código de welcome.blade.php



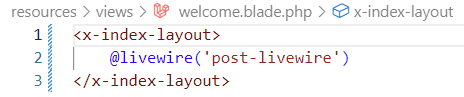
Agregamos $slot para renderizar las páginas dentro del layout

Creamos en App/view/components/IndexLayout.php



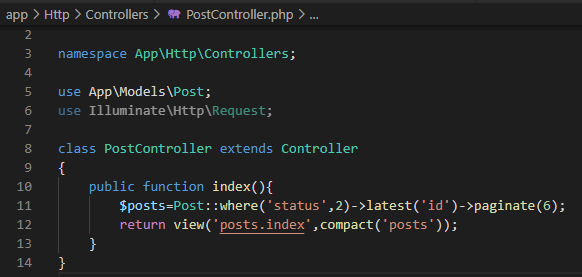
Copiamos de AppLayout el código y lo modificamos

Finalmente el welcome.blade.php



1. **Vista de Posts (Web pública)**

Mejoramos el resultado de posts con una consulta que solo muestre los posts publicados



**Agregamos la paginación con:**

****

**Algunos templates que podemos usar**

[**https://github.com/tailwindtoolbox/Nordic-Store**](https://github.com/tailwindtoolbox/Nordic-Store)

[**https://github.com/davidgrzyb/tailwind-blog-template**](https://github.com/davidgrzyb/tailwind-blog-template)

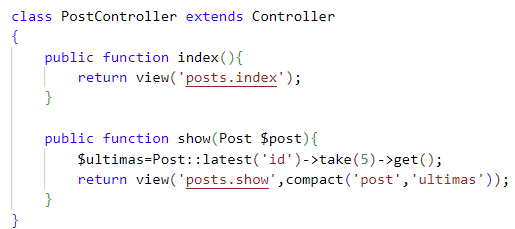
[**https://github.com/aribudin/tailmater**](https://github.com/aribudin/tailmater)

1. **DETALLE DE POST**

Primero agregamos la ruta:



Creamos el controlador en PostController



Creamos resources/views/posts/show.blade.php

| <x-index-layout>  <div>  <div class="border-y-2 border-indigo-600 py-3">  <h3>{{$post->category->name}}</h3>  </div>  <div>  <h1 class="text-2xl font-bold py-6">{{$post->name}}</h1>  </div>  <div class="grid grid-cols-3 gap-6 w-full">  <div class="col-span-2">  <div>  <img class="w-full" src="{{Storage::url($post->image->url)}}" alt="">  </div>  <div>  <p class="font-bold border-y-2 border-indigo-600 border-dotted py-3 my-4">Publicado el {{$post->created\_at}}</p>  <p class="text-sm">{{$post->body}}</p>  </div>  </div>  <div>  <h3 class="bg-indigo-600 text-white p-2">Ultimas noticias</h3>  @foreach ($ultimas as $item)  <div class="grid grid-cols-2 my-4 border-b-2 border-indigo-600 pb-4">  <div>  <img src="{{Storage::url($item->image->url)}}" alt="">  </div>  <div class="ml-4">  <h4>{{$item->name}}</h4>  </div>  </div>  @endforeach  </div>  </div>  </div>  </x-index-layout> |
| --- |

Resultado:



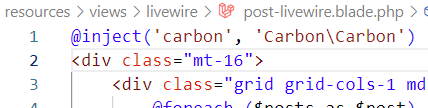
* Agregar filtros de posts al seleccionar una categoría en el menú
* Usar las url de tipo slug

Para activar las URL Slug debemos agregar el siguiente código en el modelos post y category

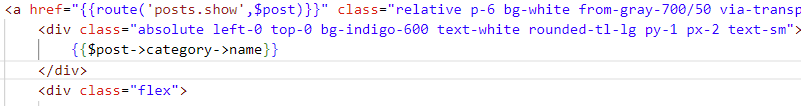
| public function getRouteKeyName(){  return 'slug';  } |
| --- |

Agregando categoría de post y fecha en los cards

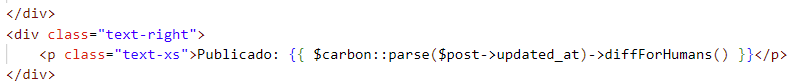
En livewire/post-livewire.blade.php

****

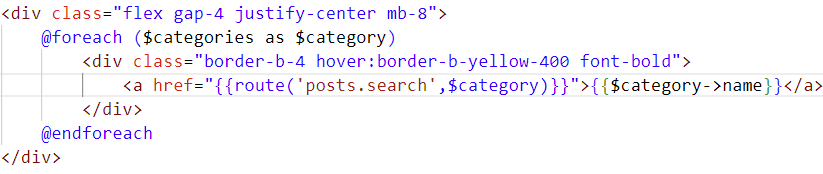
Para mostrar la categoría



Para agregar el tiempo de publicación



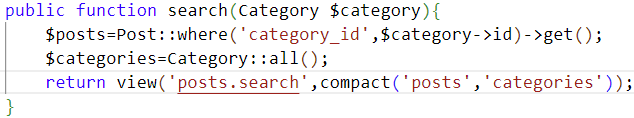
Agregaremos al menú categorías la ruta



Creamos la ruta: routes/web.php

| Route::get('category/{category}',[PostController::class,'search'])->name('posts.search'); |
| --- |

En el controlador App/Http/Controllers/PostController.php creamos el método

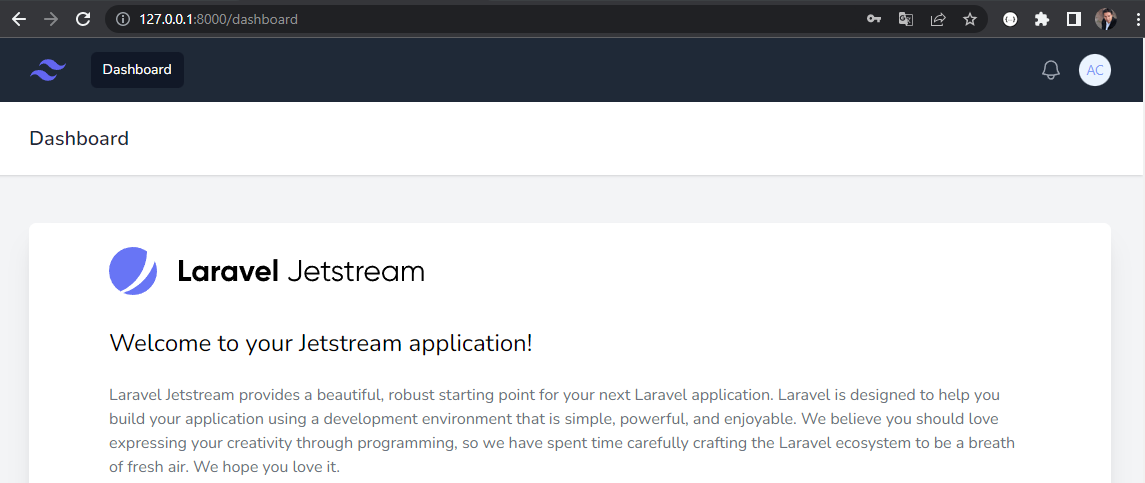


Creamos la vista en la carpeta views/posts/search.blade.php

| @inject('carbon', 'Carbon\Carbon')  <x-index-layout>  <div class="flex gap-4 justify-center mb-8">  @foreach ($categories as $category)  <div class="border-b-4 hover:border-b-yellow-400 font-bold">  <a href="{{route('posts.search',$category)}}">{{$category->name}}</a>  </div>  @endforeach  </div>  <div class="grid grid-cols-1 md:grid-cols-2 gap-6 lg:gap-8">  @foreach ($posts as $post)  <a href="{{route('posts.show',$post)}}" class="relative p-6 bg-white from-gray-700/50 via-transparent rounded-lg shadow-2xl shadow-gray-500/20 motion-safe:hover:scale-[1.01] transition-all duration-250 focus:outline focus:outline-2 focus:outline-red-500">  <div class="absolute left-0 top-0 bg-indigo-600 text-white rounded-tl-lg py-1 px-2 text-sm">  {{$post->category->name}}  </div>  <div class="flex">  <div class="bg-slate-50 flex items-center justify-center">  <img src="{{Storage::url($post->image->url)}}">  </div>  <div class="w-80 ml-4">  <h2 class="mt-6 text-xl font-semibold text-gray-900">{{$post->name}}</h2>  <p class="mt-4 text-gray-500 text-sm leading-relaxed">  {{$post->extract}}  </p>  </div>  </div>  <div class="text-right">  <p class="text-xs text-gray-400">Publicado: {{ $carbon::parse($post->updated\_at)->diffForHumans() }}</p>  </div>  </a>  @endforeach  </div>  </x-index-layout> |
| --- |

1. **CRUD CATEGORÍAS CON TAILWIND:**

**Utilizaremos el dashboard que nos proporciona Laravel Jetstream, esta incluye Alpinejs y Tailwind**

****

Para crear un CRUD podemos hacerlo de 2 formas:

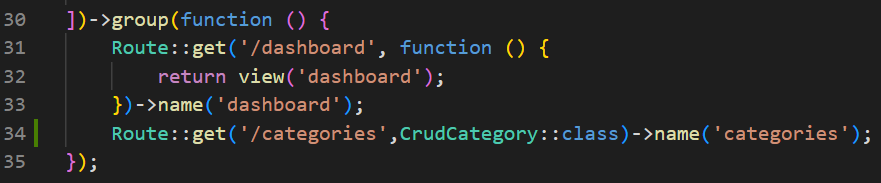
* Usar los controladores normales de laravel, similar al ítem 12 de la guía.
* Usar componentes Jetstream, usaremos esta forma

Agregamos la opción Categorías al Menu, para esto creamos un component de tipo Livewire

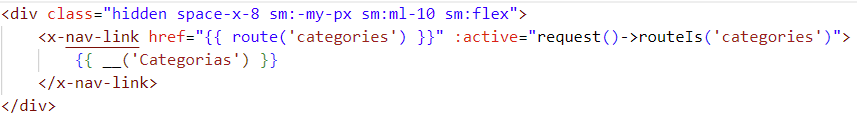
**php artisan make:livewire CrudCategory**

Esto crea un controlador de tipo livewire y una vista blade, que están enlazadas para usar databinding

Creamos la ruta dentro del middleware que apunte al componente CrudCategory

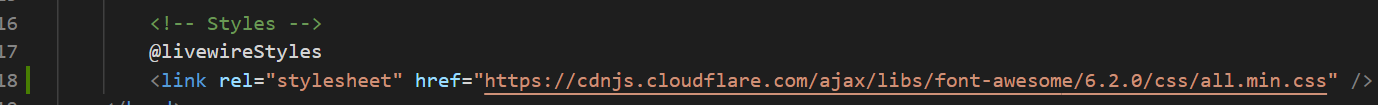


Agregamos el ítem categoría en el menú de navegación: navigation-menu.blade.php



Para que se muestren los iconos necesitamos fontawesome, debemos agregar el cdn en views/layout/app.blade.php

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.2.0/css/all.min.css" />



**En crud-category.blade.php (LISTAR)**

| <div class="py-5">  <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">  <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg px-4 py-4">  <div class="flex items-center justify-between">  <!--Input de busqueda -->  <div class="flex items-center p-2 rounded-md flex-1">  <label class="w-full relative text-gray-400 focus-within:text-gray-600 block">  <svg class="pointer-events-none w-8 h-8 absolute top-1/2 transform -translate-y-1/2 left-3" viewBox="0 0 25 25" fill="none" stroke="currentColor" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" viewBox="0 0 24 24" class="w-6 h-6"><path d="M21 21l-6-6m2-5a7 7 0 11-14 0 7 7 0 0114 0z"></path></svg>  <x-jet-input type="text" wire:model="search" class="w-full block pl-14" placeholder="Buscar equipo..."/>  </label>  </div>  <!--Boton nuevo -->  <div class="lg:ml-40 ml-10 space-x-8">  <button wire:click="create()" class="bg-indigo-600 px-4 py-2 rounded-md text-white font-semibold tracking-wide cursor-pointer" >  <i class="fa-solid fa-plus"></i> Nuevo  </button>  @if($isOpen)  @include('livewire.category-create')  @endif  </div>  </div>  <!--Tabla lista de items -->  <div class="shadow overflow-hidden border-b border-gray-200 sm:rounded-lg">  <table class="w-full divide-y divide-gray-200 table-auto">  <thead class="bg-indigo-500 text-white">  <tr class="text-left text-xs font-bold uppercase">  <td scope="col" class="px-6 py-3">ID</td>  <td scope="col" class="px-6 py-3">Categoría</td>  <td scope="col" class="px-6 py-3">Slug</td>  <td scope="col" class="px-6 py-3">Creado</td>  <td scope="col" class="px-6 py-3">Actualizado</td>  <td scope="col" class="px-6 py-3">Opciones</td>  </tr>  </thead>  <tbody class="divide-y divide-gray-200">  @foreach ($categories as $item)  <tr class="text-sm font-medium text-gray-900">  <td class="px-6 py-4">  <span class="px-2 inline-flex text-xs leading-5 font-semibold rounded-full bg-indigo-500 text-white">  {{$item->id}}  </span>  </td>  <td class="px-6 py-4">{{$item->name}}</td>  <td class="px-6 py-4">{{$item->slug}}</td>  <td class="px-6 py-4">{{$item->created\_at}}</td>  <td class="px-6 py-4">{{$item->updated\_at}}</td>  <td class="px-6 py-4">  {{-- @livewire('cliente-edit',['cliente'=>$item],key($item->id)) --}}  <x-jet-button wire:click="edit({{$item}})">  <i class="fas fa-edit"></i>  </x-jet-button>  <x-jet-danger-button wire:click="$emit('deleteItem',{{$item->id}})"> <!-- Usamos metodos magicos -->  <i class="fas fa-trash"></i>  </x-jet-danger-button>  </td>  </tr>  @endforeach  </tbody>  </table>  </div>  @if(!$categories->count())  No existe ningun registro conincidente  @endif  @if($categories->hasPages())  <div class="px-6 py-3">  {{$categories->links()}}  </div>  @endif  </div>  </div>  <!--Scripts - Sweetalert -->  @push('js')  <script>  Livewire.on('deleteItem',id=>{  Swal.fire({  title: 'Are you sure?',  text: "You won't be able to revert this!",  icon: 'warning',  showCancelButton: true,  confirmButtonColor: '#3085d6',  cancelButtonColor: '#d33',  confirmButtonText: 'Yes, delete it!'  }).then((result) => {  if (result.isConfirmed) {  //alert("del");  Livewire.emitTo('crud-team','delete',id);  Swal.fire(  'Deleted!',  'Your file has been deleted.',  'success'  )  }  })  });  </script>  @endpush  </div> |
| --- |

**Integrando SweetAlert para mostrar mensajes:**

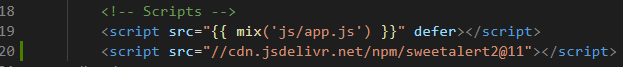
SweetAlert se lanzó bajo la licencia MIT, lo que significa que es totalmente gratuito. Está hecho a partir de un archivo JavaScript y CSS y no tiene dependencias. Comenzar es bastante fácil: además del archivo JavaScript, debe incluir un archivo de hoja de estilo, que contiene todos los estilos de alerta, en el documento HTML. Luego puede llamar a varias alertas a través de «swal()».

<https://sweetalert2.github.io/>

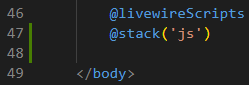
Copiamos el CDN desde la opcion install

| <script src="//cdn.jsdelivr.net/npm/sweetalert2@11"></script> |
| --- |

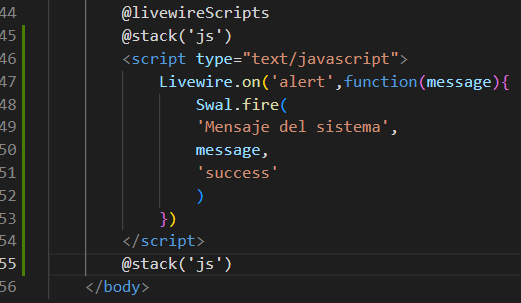
Lo pegamos en app.blade.php



En el mismo archivo agregamos el **stack** para usar javascript y el script para mostrar mensajes de tipo alerta

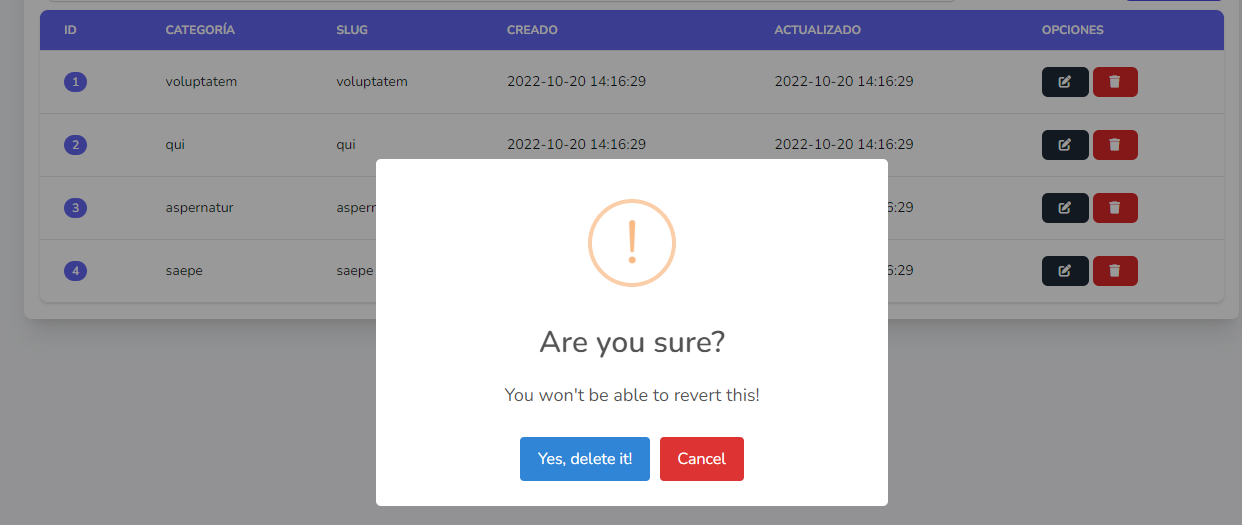


| <script type="text/javascript">  Livewire.on('alert',function(message){  Swal.fire(  'Mensaje del sistema',  message,  'success'  )  })  </script>  @stack('js') |
| --- |



En el archivo de la vista.blade.php agregamos la directiva @push y los scripts para mostrar el mensaje.

Para comprobar el funcionamiento, presionamos el botón de eliminar un registro, se debe mostrar un mensaje de confirmación.



**CREATE, UPDATE Y DELETE:**

CrudCategory.php

| <?php  namespace App\Http\Livewire;  use App\Models\Category;  use Livewire\Component;  use Illuminate\Support\Str;  class CategoryLivewire extends Component{  public $isOpen=false;  public $search;  public $category;  protected $listeners=['render','delete'=>'delete'];  protected $rules=[  'category.name'=>'required',  'category.slug'=>'required',  ];  public function messages(){  return [  'category.name' => 'Ingrese un nombre a la categoría',  'category.slug' => 'Falta el Slug a la categoría',  ];  }  public function render(){  $categories=Category::where('name','like','%'.$this->search.'%')->paginate();  return view('livewire.category-livewire',compact('categories'));  }  public function create(){  $this->isOpen=true;  }  public function store(){  $this->validate();  //dd($this->category);  if(!isset($this->category['id'])){  Category::create($this->category);  }else{  $category=Category::find($this->category['id']);  $category->name=$this->category['name'];  $category->slug=$this->category['slug'];  $category->save();  }  $this->reset(['isOpen','category']);  $this->emitTo('CrudCategory','render');  $this->emit('alert','Registro creado satisfactoriamente');  }  public function edit($category){  //dd($category);  $this->category=$category;  $this->isOpen=true;  }  public function delete($id){  Category::find($id)->delete();  }  public function updatedCategoryName(){  $this->category['slug']=Str::slug($this->category['name']);  }  } |
| --- |

Creamos un vista dentro de views/livewire que se incluya dentro de la vista de listar

**category-create.blade.php**

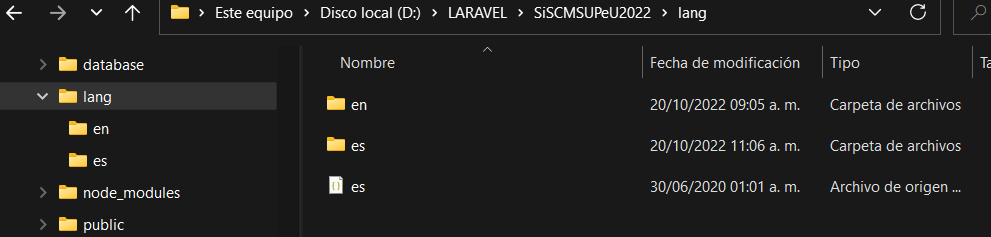
| <div>  <x-jet-dialog-modal wire:model="isOpen">  <x-slot name="title">  <h3>Registro nueva categoría</h3>  </x-slot>  <x-slot name="content">  <form>  <div class="flex justify-between mx-2 mb-6">  <div class="mb-2 md:mr-2 md:mb-0 w-full">  <x-jet-label value="Nombre categoría" class="font-bold"/>  <x-jet-input type="text" wire:model.defer="category.name" class="w-full"/>  <x-jet-input-error for="category.name"/>  </div>  </div>  <div class="flex justify-between mx-2 mb-6">  <div class="mb-2 md:mr-2 md:mb-0 w-full">  <x-jet-label value="Slug del categoría" class="font-bold"/>  <x-jet-input type="text" wire:model.defer="category.slug" class="w-full"/>  <x-jet-input-error for="category.slug"/>  </div>  </div>  </form>  </x-slot>  <x-slot name="footer">  <x-jet-secondary-button wire:click="$set('isOpen',false)" class="mx-2">Cancelar</x-jet-secondary-button>  <x-jet-button wire:click.prevent="store()" wire:loading.attr="disabled" wire:target="store" class="disabled:opacity-25">  Registrar  </x-jet-button>  </x-slot>  </x-jet-dialog-modal>  </div> |
| --- |

En model/Category agregar fillable

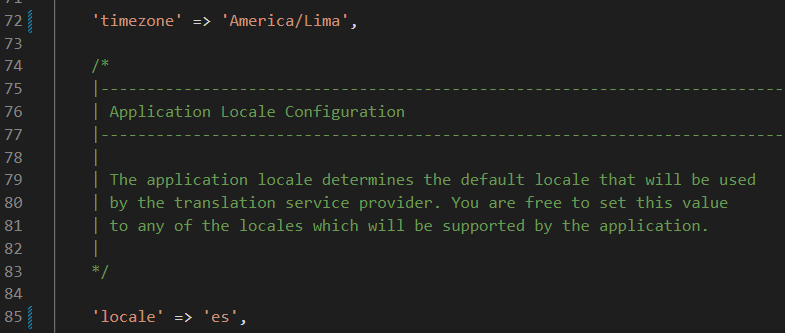
| protected $fillable=['name','slug']; |
| --- |

Finalmente para usar español en las validaciones y otros elementos de laravel podemos descargar: <https://github.com/Laraveles/spanish>

copiamos la carpeta resources/lang/es



En el archivo config.app.php, indicamos el idioma que usaremos



También cambiamos la zona horaria a “America/Lima”



\*Implemente la funcionalidad de generar Slug al crear una categoría

| use Illuminate\Support\Str;  public function updatedCategoryName(){  $this->category['slug']=Str::slug($this->category['name']);  } |
| --- |

1. CRUD Posts
2. Desactivar el registro con acceso público

