**Introducción**

Siguiendo el vídeo no puede realizar la instalación

Lo consigo instalando Composer y después ejecutando el comando

composer create-project laravel/laravel publicaciones desde la carpeta htdocs.

Composer require laravel/breeze --dev

Php artisan breeze:install

Ahora instalamos Node.js para poder continuar con los comandos npm e instalar los elementos necesarios para el proyecto de laravel

Npm install

Npm run dev

Finaliza de forma distinta al vídeo y no crea los css en la carpeta public. Todo lo demás parece que está correcto.

Base de datos inicial

Creamos una vbase de datos vacía con el nombre de “publicaciones” con nuestro cliente de mysql que es PhpmyAdmin.

Establecemos en el fichero ,env las variables de entorno de la base de datos(nombre, usuario, password)

Php aristan migrate

Y se crean las tablas que se definen en los archivos de la carpeta migrations.

**Datos**

php artisan make:model Post

database\migrations\2024\_01\_04\_103911\_create\_posts\_table.php

Con esto estamos creando el esquema para la creación de la tabla “posts”

Añadimos las instrucciones para la creación de un campo user\_id en la tabla posts y establecemos una clave ajena en el campo user\_id que se relaciona con el campo “id” de la tabla “users”.

$table->unsignedBigInteger('user\_id');

            $table->foreign('user\_id')->references('id')->on('users');

            $table->text('body');

Ejecutamos de nuevo php artisan migrate y se crea la tabla en la base de datos.

Comprobamos que se ha creado la tabla y que no nos permite introducir datos si no hay un usuario debido a las restricciones de la clave ajena.

Php artisan migrate:refresh

Genera todas las tablas de la base de datos destruyendo las anteriores.

Añadimos un usuario mediante el formulario Register y comprobamos en la Base de Datos que se puede añadir registros en la tabla “posts” ahora que ya existe un usuario.

 $table->foreign('user\_id')->references('id')->on('users')->onDelete('cascade');

Php artisan migrate:refresh

Se crean de nuevo las tablas. Creamos un usuario mediante el formulario register y luego un post en la base de datos. Comprobamos que se puede borrar el usuario y se eliminan los posts asociados a éste.

Lo siguiente es establecer en Laravel la multiplicidad entre las clases Post y User ya que un usuario pude tener muchos posts pero cada post sólo puede pertenecer a un usuario.

Para ello modificamos la clase User de la siguiente manera:

Añadimos la función posts()

public function posts()

    {

        return $this->hasMany(Post::class);

    }

En la clase post

public function user()

    {

        return $this->belongsTo(User::class);

    }

Y añadimos en Post

protected $fillable = ['body'];

para permitir la asignación en masa en el campo body.

Desarrollo

Ruta y controlador

Escribimos el comando php artisan make:controller PostController para crear el archivo del controlador PostController.php

class PostController extends Controller

{

    public function index()

    {

        return 'index';

    }

    public function store()

    {

        //guardar;

    }

    public function destroy()

    {

        //eliminar

    }

}

Añadimos las funciones index, store y destroy y pasamos a crear las rutas en el archivo routes/web.php

Añadimos

use App\Http\Controllers\PostController;

para importar el controlador que hemos definido.

Y añadimos las rutas, una para cada método que hemos creado.

Route::get('posts',[PostController::class, 'index'])->name('posts.index');

Route::post('posts',[PostController::class, 'store'])->name('posts.store');

Route::delete('posts/{post}', [PostController::class, 'destroy'])->name('posts.index.destroy');

Añadimos en la vista una entrada en la barra de navegación que redirige a la ruta posts.index

En el archivo resources/views/layout/navigation.blade.php

Añadimos en Navigation links

  <x-nav-link :href="route('posts.index')" :active="request()->routeIs('posts.index')">

                        {{ \_\_('Post') }}

                    </x-nav-link>

y terminamos añadiendo una entrada para el botón de hamburguesa

  <x-responsive-nav-link :href="route('posts.index')">

                        {{ \_\_('Posts') }}

                    </x-responsive-nav-link>

Estructura de un formulario

Modificamos la clase PostsController para que el método store() devuelva una vista.

 public function index()

    {

        return view('posts.index');

    }

Creamos el archivo de la vista en resources/views/posts/index.blade.php

Tomamos como plantilla el contenido del archivo resources/views/dashboard.blade.php y lo modificamos.

Quedará de la siguiente forma

<x-app-layout>

    <x-slot name="header">

        <h2 class="font-semibold text-xl text-gray-800 leading-tight">

            {{ \_\_('Posts') }}

        </h2>

    </x-slot>

    <div class="py-12">

        <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">

            <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg">

                <div class="p-6 text-gray-900">

                    Formulario

                </div>

            </div>

        </div>

    </div>

</x-app-layout>

Añadimos una directiva para hacer el formulario seguro

  <form action="{{route('posts.store')}}" method="POST" >

                    @csrf

                    FORMULARIO

                   </form>

La directiva es @csrf y genera un campo oculto con un token.

Diseño de un formulario

Creamos un archivo en resources/views/components/textarea.blade.php para definir el campo body en el formulario

<textarea

    rows="2"

    {!!

        $attributes->merge([

            'class' => 'border-gray-300 focus:border-indigo-500 focus:ring-indigo-500 rounded-md shadow-sm'

            ])

    !!}>

</textarea>

En el archivo resources/views/index.blade.php modificamos el formulario que teníamos creado añadiéndole un campo textarea.

  <form action="{{route('posts.store')}}" method="POST" >

                    @csrf

                    <div class="mt-4">

                        <x-input-label for="body" :value="\_\_('Body')" />

                        <x-textarea class="block mt-1 w-full" name="body" required />

                    </div>

                   </form>

Procedemos a crear un botón Save para el formulario

 <div class="flex justify-end mt-4">

                        <x-primary-button>

                             {{ \_\_('Save') }}

                        </x-primary-button>

                    </div>

Middleware Auth

En esta sección vamos a eliminar ´código innecesario para esta aplicación y a agregar seguridad mediante Middleware Auth

Para ello modificamos el archivo de rutas routes\auth.php y eliminamos las que no se van a usar. Quedará de la siguente manera.

use App\Http\Controllers\Auth\AuthenticatedSessionController;

use App\Http\Controllers\Auth\RegisteredUserController;

use Illuminate\Support\Facades\Route;

Route::middleware('guest')->group(function () {

    Route::get('register', [RegisteredUserController::class, 'create'])

                ->name('register');

    Route::post('register', [RegisteredUserController::class, 'store']);

    Route::get('login', [AuthenticatedSessionController::class, 'create'])

                ->name('login');

    Route::post('login', [AuthenticatedSessionController::class, 'store']);

});

Route::middleware('auth')->group(function () {

    Route::post('logout', [AuthenticatedSessionController::class, 'destroy'])

                ->name('logout');

});

Eliminamos la página de bienvenida resources/views/welcome.blade.php

Como ya no existe la vista welcome vamos a establecer una redirección en el archivo routes\web.php

Route::redirect('/', 'dashboard'); redirige todo lo que vaya a la raíz ‘/’ a la ruta dashboard.

Route::get('dashboard',[PostController::class, 'index'])->middleware(['auth', 'verified'])->name('dashboard'); indica que dashboard es la ruta para index. Además queremos que el dashboard sea directamente la vista de publicaciones. Para ello modificamos el archivo routes\web.php

Route::redirect('/', 'dashboard');

Route::middleware('auth')->group(function () {

    Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');

    Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');

    Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');

});

Route::get('dashboard',[PostController::class, 'index'])->middleware(['auth', 'verified'])->name('dashboard');

Route::post('posts',[PostController::class, 'store'])->middleware(['auth', 'verified'])->name('posts.store');

Route::delete('posts/{post}', [PostController::class, 'destroy'])->name('posts.index.destroy');

require \_\_DIR\_\_.'/auth.php';

En las rutas hemos añadido el método Middleware Auth para protegerlas con aurenticación, es decir que solo permite el acceso a usuarios logueados. Si no lo está se redirige a la página de login.

Ahora la ruta que no funciona es posts.index a la que hacíamos referencia en resources\views\layouts\navigation.blade.php. En ésta habíamos incorporado una entrada en la barra de navegación y otra en el botón hamburguesa.

Las eliminamos y ahora en la página de dashboard aparece directamente la página index.blade.php que habíamos creado con el formulario.