

PRÁCTICA MF 0967_3

Para la realización de esta práctica vamos a seguir el siguiente tutorial para llevar a cabo un proyecto con Laravel y Tailwind.

<https://cosasdedevs.com/posts/vamos-a-crear-un-blog-con-laravel-8-y-tailwind-css/>

Creamos un nuevo proyecto de Laravel, a través de Laragon.

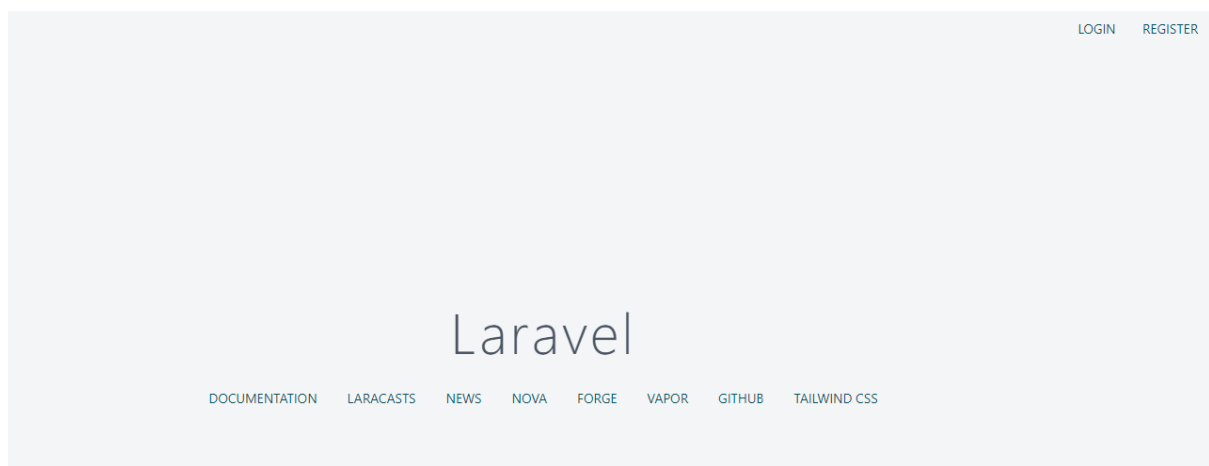
Una vez que tenemos nuestro proyecto en marcha ejecutamos el siguiente comando: “`composer require laravel/ui --dev`” con el vamos a crear un sistema de autenticación.

Después ejecutamos el comando “`composer require laravel-frontent-presets/tailwindcss --dev`” con el que conseguiremos añadir estilos de css.

Una vez hecho esto ejecutamos el comando “`php artisan ui tailwindcss --auth`” de esta forma especificamos que queremos utilizar tailwind.

A continuación ejecutamos los comandos “`npm install && npm run dev`” el último da error aún ejecutando los comandos por separado. Lo solucionamos ejecutando el comando `npm audit fix --force`

De esta forma la compilación se lleva a cabo. Seguido de esto ejecutamos el comando `php artisan serve` y ya podemos ver en nuestro navegador nuestro proyecto con Laravel y Tailwind.



Una vez hecha esta primera parte, vamos a trabajar con la base de datos:

En el archivo `.env` establecemos la conexión con la bbdd.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel-tailwind
DB_USERNAME=root
DB_PASSWORD=ukJbhaVJ
```

Después creamos los modelos y las migraciones con los siguientes comandos:

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

```
php artisan make:model Comment -m
```

Una vez que tenemos la estructura, vamos a modificar los siguientes archivos:

En primer lugar en el archivo `create_users_table.php` modificamos el siguiente código:

```
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->rememberToken();
        $table->timestamps();
        $table->boolean('is_staff')->default(false);
        $table->boolean('is_admin')->default(false);
    });
}
```

En segundo lugar en el archivo `app/Models/User.php` añadimos al final de la clase el siguiente código:

```
public function isAdmin()
{
    return $this->is_admin;
}

public function isStaff()
{
    return $this->is_staff;
}
```

```

    }

    public function authorizeRoles($roles)
    {
        abort_unless($this->hasAnyRole($roles), 404);
        return true;
    }

    public function hasAnyRole($roles)
    {
        if (is_array($roles)) {
            foreach ($roles as $role) {
                if ($this->$role) {
                    return true;
                }
            }
        } else {
            if ($this->$roles) {
                return true;
            }
        }
        return false;
    }
}

```

A continuación en el archivo *create_comments_table.php* modificamos la function up con el siguiente código:

```

    public function up()
    {
        Schema::create('comments', function (Blueprint $table) {
            $table->id();

            $table->foreignId('user_id')->constrained();
            $table->foreignId('post_id')->constrained();

            $table->text('comment');

            $table->timestamps();
        });
    }
}

```

Después vamos al modelo *comment.php* y lo sustituimos por el siguiente código:

```
<?php
```

```
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Comment extends Model
{
    use HasFactory;

    protected $fillable = [
        'comment', 'user_id', 'post_id'
    ];

    public function user()
    {
        return $this->belongsTo(User::class);
    }

    public function post()
    {
        return $this->belongsTo(Post::class);
    }
}
```

Después en el archivo *create_posts_table.php* modificamos la función up:

```
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();

        $table->foreignId('user_id')->constrained();

        $table->string('title')->unique();
        $table->string('slug')->unique();
        $table->mediumText('body');
        $table->boolean('is_draft')->default(false);

        $table->timestamps();
    });
}
```

Una vez hemos hecho esto configuramos el modelo Posts con el siguiente comando:

```
composer require cviebrock/eloquent-sluggable
```

Ahora vamos al archivo *app/Models/Post.php* y sustituimos el código por el siguiente:

`<?php`

```
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Cviebrock\EloquentSluggable\Sluggable;

class Post extends Model
{
    use HasFactory;
    use Sluggable;

    /**
     * Get the user record associated with the post.
     */
    public function user()
    {
        return $this->belongsTo(User::class);
    }

    /**
     * Get the comments for the blog post.
     */
    public function comments()
    {
        return $this->hasMany(Comment::class);
    }

    /**
     * Return the sluggable configuration array for this model.
     *
     * @return array
     */
    public function sluggable()
    {
        return [
            'slug' => [
                'source' => 'title',
                'onUpdate' => true
            ]
        ];
    }

    public function getGetLimitBodyAttribute()
    {

```

```

        return substr($this->body, 0, 140) . '...';
    }
}

```

A continuación creamos las tablas en la base de datos con el siguiente comando en la terminal: `php artisan migrate`

Después creamos datos fake con el comando `php artisan make:factory PostFactory -m Post`

Seguido esto modificamos el código del archivo `PostFactory.php`:

`<?php`

```

namespace Database\Factories;

use App\Models\Post;
use Illuminate\Database\Eloquent\Factories\Factory;

class PostFactory extends Factory
{
    /**
     * The name of the factory's corresponding model.
     *
     * @var string
     */
    protected $model = Post::class;

    /**
     * Define the model's default state.
     *
     * @return array
     */
    public function definition()
    {
        return [
            'user_id' => 1,
            'title' => $this->faker->sentence,
            'body' => $this->faker->text(3000),
        ];
    }
}

```

A continuación vamos al archivo `DatabaseSeeder.php` y sustituimos el código por:

`<?php`

```

namespace Database\Seeders;

```

```

use Illuminate\Database\Seeder;
use App\Models\User;
use App\Models\Post;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        User::create([
            'name' => 'Alber',
            'email' => 'alber@cosasdedevs.com',
            'password' => bcrypt('admin123'),
            'is_admin' => true,
            'is_staff' => true,
        ]);

        Post::factory()->count(50)->create();
    }
}

```

Creamos un usuario sustituyendo el código por el usuario que queramos.

```

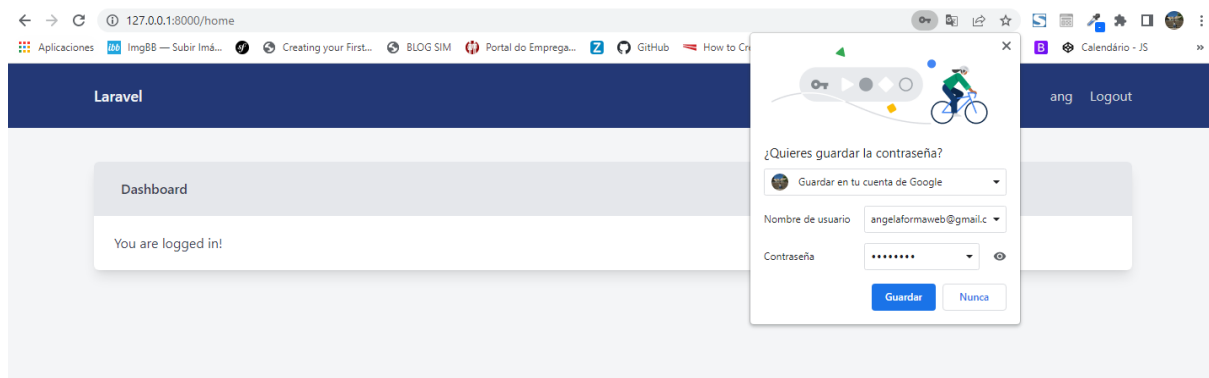
public function run()
{
    User::create([
        'name' => 'ang',
        'email' => 'angelaformaweb@gmail.com',
        'password' => bcrypt('admin123'),
        'is_admin' => true,
        'is_staff' => true,
    ]);

    Post::factory()->count(50)->create();
}

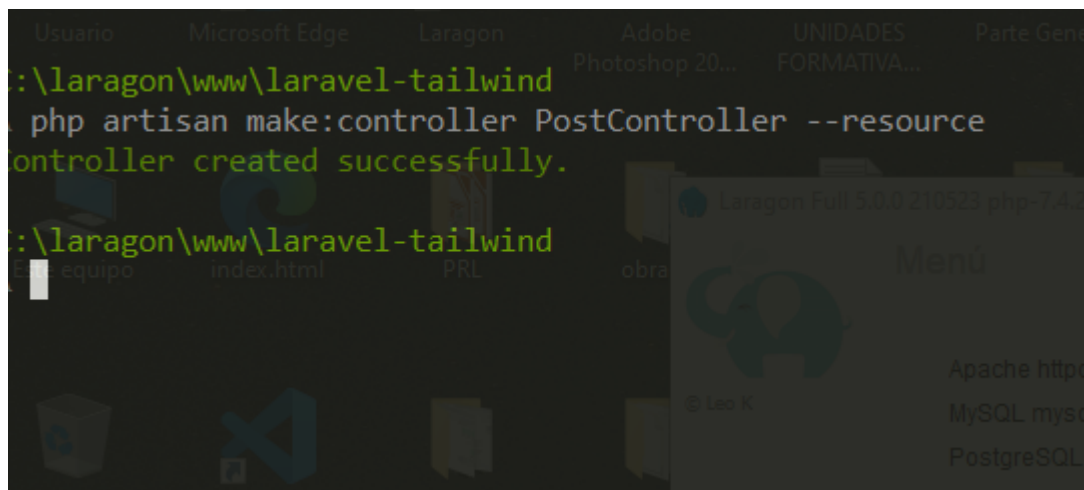
```

Después ejecutamos el comando: `php artisan db:seed`

Ya nos podemos loguear:



Una vez que nos podemos loguear vamos a continuar con la práctica, vamos a ejecutar el comando `php artisan make:controller PostController --resource` para crear el controlador que administra nuestros posts.



Una vez hecho esto en el archivo `app/Http/Controllers/PostController.php` al principio de la clase añadimos dos nuevas funciones:

```
public function home()
{
    return view('posts/home', [
        'posts' => Post::where('is_draft', 0)->orderBy('created_at',
'desc')->get()->take(6)
    ]);
}

/**
 * Display the specified resource.
 *
 * @param string $slug
 * @return \Illuminate\Http\Response
 */
```



```

    public function detail($slug)
    {
        $post = Post::where('slug', $slug)->where('is_draft',
false)->first();
        abort_unless($post, 404);
        return view('posts/post', [
            'post' => $post
        ]);
    }
}

```

Crearemos una nueva carpeta *resources/views/posts* dentro de la vistas y creamos el archivo *home.blade.php*

A continuación modificamos el archivo *resources/views/layouts/app.blade.php*:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>My Laravel Blog</title>
    <!-- Styles -->
    <link href="{{ mix('css/app.css') }}" rel="stylesheet">
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.14.0/css/all.m
in.css"
integrity="sha512-1PK0gIY59xJ8Co8+NE6FZ+LOAZKjy+KY8iq0G4B3CyeY6wYHN3yt9PW0
XpSriVlkMXe40PTKnXrLnZ9+fkDaog==" crossorigin="anonymous" />

</head>
<body>
    <header class="w-full">
        <nav class="w-full bg-orange-300 p-1 text-white flex
justify-center">
            <div class="w-full flex justify-between px-4">
                @guest
                <ul class="flex justify-between" style="width:130px">
                    <li>
                        <a class="hover:text-blue-600" href="{{ {
route('home') }}">
                            HOME
                        </a>
                    </li>
                    <li>
                        <a class="hover:text-blue-600" href="{{ {
route('login') }}">

```

```

                <i class="fas fa-sign-in-alt"></i>
            </a>
        </li>
        <li>
            <a class="hover:text-blue-600" href="{
route('register') }}">
                <i class="fas fa-user-plus"></i>
            </a>
        </li>
    </ul>
    @else
        <ul class="flex justify-between" style="width:140px">
            <li>
                <a class="hover:text-blue-600" href="{
route('home') }}">
                    HOME
                </a>
            </li>
            <li>{{ Auth::user()->name }}</li>
            @if( Auth::user()->isAdmin() or
Auth::user()->isStaff() )
                <li>
                    <a class="hover:text-blue-600" href="{
route('posts.store') }}" title="Admin">
                        <i class="fas fa-user-shield"></i>
                    </a>
                </li>
            @endif
            <li>
                <a class="hover:text-blue-600" href="{
route('logout') }}" title="logout" class="no-underline hover:underline"
onclick="event.preventDefault();
document.getElementById('logout-form').submit();"><i class="fas
fa-sign-out-alt"></i></a>
                <form id="logout-form" action="{ route('logout')
}" method="POST" class="hidden">
                    {{ csrf_field() }}
                </form>
            </li>
        </ul>
    @endguest
    <ul class="flex justify-between" style="width:99px">
        <li>
            <a class="hover:text-blue-600" href="http://">
                <i class="fab fa-twitter"></i>
            </a>
        </li>
    </ul>

```

```

        <li>
            <a class="hover:text-blue-600" href="http://">
                <i class="fab fa-facebook-f"></i>
            </a>
        </li>
        <li>
            <a class="hover:text-blue-600" href="http://">
                <i class="fas fa-rss"></i>
            </a>
        </li>
    </ul>
</div>
</nav>
<div class="text-center py-8 text-4xl font-bold">
    <h1>My Laravel Blog</h1>
</div>
</header>
@yield('content')
<footer class="mt-12">
    <div class="max-w-full bg-orange-300 p-4"></div>
    <div class="max-w-full text-center bg-gray-700 text-white p-4">
        <div class="text-lg font-bold">@MyLaravelBlog By <a
class="hover:underline" href="https://cosasdedevs.com/"
target="_blank">Alberto Ramírez</a></div>
    </div>
</footer>
</body>
</html>

```

A continuación añadimos el siguiente código al archivo home.blade.php de la carpeta posts:

```

@extends('..layouts.app')

@section('content')
<section class="w-full bg-gray-200 py-4 flex-row justify-center
text-center">
    <h2 class="py-4 text-3xl">About me</h2>
    <div class="flex text-justify justify-center">
        <div class="max-w-5xl px-2">
            Lorem ipsum dolor sit, amet consectetur adipisicing elit.
            Voluptate necessitatibus ullam commodi perferendis accusamus sint error
            sequi, dolorem nam, vel praesentium dignissimos nostrum quod fuga corporis
            asperiores laudantium, possimus veniam!
            Lorem ipsum dolor sit amet, consectetur adipisicing elit.
            Consectetur iure cumque qui impedit quod earum dolores nisi nemo totam
            vero natus aperiam, libero consequuntur nesciunt atque officia
            exercitationem rerum. Veritatis!
        </div>
    </div>
</section>

```

Lorem ipsum dolor sit amet consectetur adipisicing elit.
 Voluptas in hic ratione recusandae nostrum, saepe aliquam alias ipsum?
 Asperiores rerum numquam officia harum atque, impedit perspiciatis facilis
 nobis tempora est!

```
    </div>
  </div>
</section>
<section class="w-full">
  <div class="flex justify-center">
    <div class="max-w-6xl text-center">
      <h2 class="py-4 text-3xl border-solid border-gray-300
border-b-2">Lasts posts</h2>
      <div class="flex flex-wrap justify-between">
        @foreach($posts as $post)
          <article style="width:300px" class="text-left p-2">
            <h3 class="py-4 text-xl">{{$post->title}}</h3>
            <p>{{$post->get_limit_body}} <a class="font-bold
text-blue-600 no-underline hover:underline" href="{{ route('posts.detail',
$post->slug) }}">Read more</a></p>
          </article>
        @endforeach
      </div>
    </div>
  </div>
</section>

@endsection
```

Después creamos el archivo post.blade.php con el siguiente código:

```
@extends('..layouts.app')
```

```
@section('content')
<section class="w-full bg-gray-200 py-4 flex-row justify-center
text-center">
  <div class="flex justify-center">
    <div class="max-w-4xl">
      <h1 class="px-4 text-6xl break-words">{{$post->title}}</h1>
    </div>
  </div>
</section>
<article class="w-full py-8">
  <div class="flex justify-center">
    <div class="max-w-4xl text-justify">
      {{$post->body}}
    </div>
  </div>
</article>
<section class="w-full py-8">
```

```

        <div class="max-w-4xl flex-row justify-start p-3 text-left ml-auto
mr-auto border rounded shadow-sm bg-gray-50">
            <h3 class="py-4 text-2xl">Comments</h3>
            <div>
                @foreach($post->comments as $comment)
                    <div class="w-full bg-white p-2 my-2 border">
                        <div class="header flex justify-between mb-4 text-sm
text-gray-500">
                            <div>
                                By {{$comment->user->name}}
                            </div>
                            <div>
                                {{$comment->created_at->format('j F, Y')}}
                            </div>
                        </div>
                        <div class="text-lg">{{$comment->comment}}</div>
                    </div>
                @endforeach
            </div>
        </section>
    @endsection

```

Modificamos el archivo *web.php* añadiendo nuevas rutas:

```

Route::get('/', [\App\Http\Controllers\PostController::class,
'home'])->name('home');

Route::get('/posts/{slug}', [\App\Http\Controllers\PostController::class,
'detail'])->name('posts.detail');

```

Este es el resultado que se ve en el navegador:



Vamos a continuar creando formularios:

En primer lugar modificamos la constante home en el archivo `app/Providers/RouteServiceProvider.php` de la siguiente forma:

```
public const HOME = '/';
```

Para crear comentarios, lo primero que necesitaremos será crear un custom request. Ejecutamos el siguiente comando:

```
php artisan make:request CommentRequest
```

```
C:\laragon\www\laravel-tailwind
λ php artisan make:request CommentRequest
Request created successfully.
    ↳ Controller.php
C:\laragon\www\laravel-tailwind
λ
    ↳ PostController.php
```

A continuación sustituimos el código en el archivo `CommentRequest.php` de la siguiente forma:

```
<?php
```

```
namespace App\Http\Requests;
```

```
use Illuminate\Auth\Access\AuthorizationException;
```

```

use Illuminate\Foundation\Http\FormRequest;
use Illuminate\Support\Facades\Auth;

class CommentRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return Auth::check();
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'comment' => 'required|max:1000',
            'post_id' => 'exists:App\Models\Post,id',
        ];
    }

    /**
     * Get the error messages for the defined validation rules.
     *
     * @return array
     */
    public function messages()
    {
        return [
            'comment.required' => 'A comment is required',
            'comment.max' => 'A comment cannot exceed 1000 characters',
            'post_id.exists' => 'You must sent a valid post'
        ];
    }

    protected function failedAuthorization()
    {
        throw new AuthorizationException('You must be logged in to write
comments');
    }
}

```

```
}
```

Después vamos a crear un controlador para administrar los comentarios con el siguiente comando:

```
php artisan make:controller CommentController
```

Después vamos al archivo *app/Http/Controllers/CommentController.php* y sustituimos el código por el siguiente:

```
<?php

namespace App\Http\Controllers;

use App\Http\Requests\CommentRequest;
use App\Models\Comment;
use App\Models\Post;
use Illuminate\Support\Facades\Auth;

class CommentController extends Controller
{
    /**
     * Store a newly created resource in storage.
     *
     * @param App\Http\Requests\CommentRequest $request
     * @return \Illuminate\Http\Response
     */
    public function store(CommentRequest $request)
    {
        $request->validated();

        $user = Auth::user();
        $post = Post::find($request->input('post_id'));

        $comment = new Comment;
        $comment->comment = $request->input('comment');
        $comment->user()->associate($user);
        $comment->post()->associate($post);

        $res = $comment->save();

        if ($res) {
```



```

        return back()->with('status', 'Comment has been created
sucessfully');
    }

    return back()->withErrors(['msg', 'There was an error saving the
comment, please try again later']);
}
}

```

Después en el archivo *resources/views/posts/post.blade.php*, archivo que se encarga de mostrar el contenido del post y añadimos el siguiente código en la línea 20 (se muestra el valor del status)

```

@guest
    <p>You must be logged in to write comments</p>
@else
    <form action="{{ route('comments.store') }}" method="post">
        @csrf
        <textarea class="w-full h-28 resize-none border rounded
focus:outline-none focus:shadow-outline p-2" name="comment"
placeholder="Write your comment here" required>{{ old('comment')
}}</textarea>
        <input type="hidden" name="post_id" value="{{ $post->id }}">
        <input type="submit" value="SEND" class="px-4 py-2
bg-orange-300 cursor-pointer hover:bg-orange-500 font-bold w-full border
rounded border-orange-300 hover:border-orange-500 text-white">
        @if (session('status'))
            <div class="w-full bg-green-500 p-2 text-center my-2
text-white">
                {{ session('status') }}
            </div>
        @endif
        @if($errors->any())
            <div class="w-full bg-red-500 p-2 text-center my-2
text-white">
                {{$errors->first()}}
            </div>
        @endif
    </form>
@endguest

```

A continuación en el archivo *web.php* añadimos la ruta:

```
Route::post('/comment', [\App\Http\Controllers\CommentController::class, 'store'])->name('comments.store');
```

De esta forma cuando la url */comment* reciba una petición de tipo *post*, la enviará a *CommentController* y al método *store* que se encargará de realizar el proceso de guardado.

Para gestionar los *post* vamos a necesitar una nueva ruta en el archivo *web.php*:

A continuación ejecutamos el comando `php artisan route:list` para ver las rutas que tenemos en el proyecto

```
C:\laragon\www\laravel-tailwind
> php artisan route:list
```

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/	home	App\Http\Controllers\PostController@home	web
	POST	admin/posts	posts.store	App\Http\Controllers\PostController@store	web
	GET HEAD	admin/posts	posts.index	App\Http\Controllers\PostController@index	web
	GET HEAD	admin/posts/create	posts.create	App\Http\Controllers\PostController@create	web
	DELETE	admin/posts/{post}	posts.destroy	App\Http\Controllers\PostController@destroy	web
	PUT PATCH	admin/posts/{post}	posts.update	App\Http\Controllers\PostController@update	web
	GET HEAD	admin/posts/{post}	posts.show	App\Http\Controllers\PostController@show	web
	GET HEAD	admin/posts/{post}/edit	posts.edit	App\Http\Controllers\PostController@edit	web
	GET HEAD	api/user		Closure	api

Ahora vamos al archivo *PostController.php* y crearemos la primera acción que será la de mostrar el listado de *posts*. Modificamos el siguiente código:

```
public function index(Request $request)
{
    abort_unless(Auth::check(), 404);

    $user = $request->user();

    if ($user->isAdmin()) {
        $posts = Post::orderBy('created_at', 'desc')->get();
    } elseif ($user->isStaff()) {
        $posts = Post::where('user_id',
$user->id)->orderBy('created_at', 'desc')->get();
    } else {
        abort_unless(Auth::check(), 404);
    }

    return view('posts/list', [
        'posts' => $posts
    ]);
}
```

```
]);
```

```
}
```

Como ya tenemos el controlador vamos a crear la plantilla, creamos un nuevo archivo *list.blade.php* en la carpeta posts, en las vistas.

Ahora vamos al navegador <http://127.0.0.1:8000/admin/posts>

Title	Creation	Author	Status	Actions
Quia veritatis quisquam veritatis officia.	24 May, 2022	ang	Published	Edit Delete
Ut tenetur delectus nihil facere.	24 May, 2022	ang	Published	Edit Delete
Est doloremque consequatur similique numquam sunt.	24 May, 2022	ang	Published	Edit Delete
Repudiandae minus neque porro et excepturi qui et.	24 May, 2022	ang	Published	Edit Delete
Voluptatem dolor voluptatem voluptas ratione nihil perferendis perspiciatis officia.	24 May, 2022	ang	Published	Edit Delete
Et quas autem porro ad beatae.	24 May, 2022	ang	Published	Edit Delete
Aut quia et ea non dolorem vitae nesciunt.	24 May, 2022	ang	Published	Edit Delete
Temporibus quia voluptatem itaque fuga aut.	24 May, 2022	ang	Published	Edit Delete
Omnis adipisci qui dolorum amet veniam.	24 May, 2022	ang	Published	Edit Delete
Harum aut rerum asperiores aspernatur.	24 May, 2022	ang	Published	Edit Delete
Quisquam deleniti facere natus odio accusantium voluptatem eos.	24 May, 2022	ang	Published	Edit Delete
Expedita nam ipsa earum totam soluta.	24 May, 2022	ang	Published	Edit Delete
Facilis ut velit rerum ut aut blanditiis nihil.	24 May, 2022	ang	Published	Edit Delete
Dolor dolorum cumque quis deserunt.	24 May, 2022	ang	Published	Edit Delete
Dolorem suscipit quos dolore quisquam nobis facere.	24 May, 2022	ang	Published	Edit Delete
Iusto et voluptates corrupti minus eos quo.	24 May, 2022	ang	Published	Edit Delete
Omnis doloremque distinctio quo possimus possimus.	24 May, 2022	ang	Published	Edit Delete
Nisi ad beatae et tempore vero corporis.	24 May, 2022	ang	Published	Edit Delete
Ut blanditiis ex corporis vel dignissimos.	24 May, 2022	ang	Published	Edit Delete
Ad error est neque odio voluptatem et est.	24 May, 2022	ang	Published	Edit Delete
Qui vero nobis accusantium ullam blanditiis.	24 May, 2022	ang	Published	Edit Delete
Fugiat facere repudiandae dicta at.	24 May, 2022	ang	Published	Edit Delete
...

Ahora añadimos un nuevo enlace en la barra de navegación, modificamos el archivo *app.blade.php*

```
@if( Auth::user()->isAdmin() or Auth::user()->isStaff() )
    <li>
        <a class="hover:text-blue-600" href="{{
route('posts.store') }}" title="Admin">
            <i class="fas fa-user-shield"></i>
        </a>
    </li>
@endif
```

Podemos comprobar cómo se añade un nuevo botón.

A continuación vamos al archivo *PostController.php* y modificamos la función create:

```
public function create(Request $request)
{
    abort_unless(Auth::check(), 404);
    $request->user()->authorizeRoles(['is_staff', 'is_admin']);
    return view('posts/create');
}
```

Ahora creamos el archivo *resources/views/posts/create.blade.php* y añadimos el siguiente código:

```
@extends('..layouts.app')
```

```
@section('content')
```

```
<section class="w-full bg-gray-200 py-4 flex-row justify-center
text-center">
```

```
<div class="flex justify-center">
```

```
<div class="max-w-4xl">
```

```
<h1 class="px-4 text-6xl break-words">Create Post</h1>
```

```
</div>
```

```
</div>
```

```
</section>
```

```
<article class="w-full py-8">
```

```
<div class="flex justify-center">
```

```
<div class="max-w-7xl text-justify">
```

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

```
@csrf
```

```
<input class="w-full border rounded focus:outline-none
focus:shadow-outline p-2 mb-4" type="text" name="title" value="{{
old('title') }}" placeholder="Write the title of the post">
```

```
<textarea class="w-full h-72 resize-none border rounded
focus:outline-none focus:shadow-outline p-2 mb-4" name="body"
placeholder="Write your post here" required>{{ old('body') }}</textarea>
```

```
<div class="mb-4">
```

```
<input type="hidden" name="is_draft" value="0">
```

```
<input type="checkbox" name="is_draft" value="1"> Is
draft?
```

```
</div>
```

```

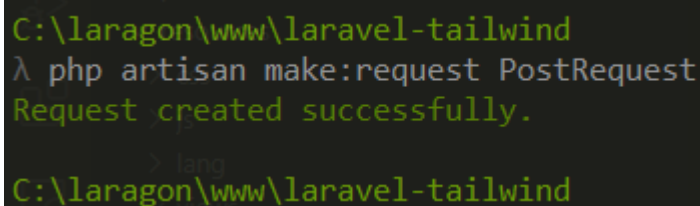
        <input type="submit" value="SEND" class="px-4 py-2
bg-orange-300 cursor-pointer hover:bg-orange-500 font-bold w-full border
rounded border-orange-300 hover:border-orange-500 text-white">
        @if (session('status'))
            <div class="w-full bg-green-500 p-2 text-center my-2
text-white">
                {{ session('status') }}
            </div>
        @endif
        @if($errors->any())
            <div class="w-full bg-red-500 p-2 text-center my-2
text-white">
                {{$errors->first()}}
            </div>
        @endif
    </form>
</div>
</div>
</article>
@endsection

```

Este último contiene un formulario con los datos que necesitaremos para crear un post.

Después necesitamos crear una request, lo haremos con el siguiente comando:

```
php artisan make:request PostRequest
```



```

C:\laragon\www\laravel-tailwind
λ php artisan make:request PostRequest
Request created successfully.
> lang
C:\laragon\www\laravel-tailwind

```

Ya creado añadimos el siguiente código:

```
<?php
```

```
namespace App\Http\Requests;
```

```
use Illuminate\Foundation\Http\FormRequest;
```

```
use Illuminate\Support\Facades\Auth;
```

```

class PostRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return Auth::check();
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'title' => 'required|max:255',
            'body' => 'required',
            'is_draft' => 'required',
        ];
    }

    /**
     * Get the error messages for the defined validation rules.
     *
     * @return array
     */
    public function messages()
    {
        return [
            'title.required' => 'A title is required',
            'title.max' => 'A title cannot exceed 255 characters',
        ];
    }
}

```

```

        'body.required' => 'You must sent a body',
        'is_draft.required' => 'You must sent if is draft or not',
    ];
}
}

```

Modificamos el siguiente código en *PostController.php*:

```

public function store(PostRequest $request)
{
    $request->validated();
    $user = Auth::user();

    $request->user()->authorizeRoles(['is_staff', 'is_admin']);

    $post = new Post;
    $post->title = $request->input('title');
    $post->body = $request->input('body');
    $post->is_draft = $request->input('is_draft');
    $post->user()->associate($user);

    $res = $post->save();

    if ($res) {
        return back()->with('status', 'Post has been created
sucessfully');
    }

    return back()->withErrors(['msg', 'There was an error saving the
post, please try again later']);
}

```

Añadimos también: `use App\Http\Requests\PostRequest;`

Ahora podremos crear nuevos posts, se ve de la siguiente forma en el navegador.

Mi Blog de Laravel

Create Post

Write the title of the post

Write your post here

☐ Is draft?

SEND

El siguiente paso es poder editar los posts. En postcontroller modificamos la función de editar:

```
public function edit(Request $request, $id)
{
    abort_unless(Auth::check(), 404);
    $request->user()->authorizeRoles(['is_staff', 'is_admin']);
    $post = Post::find($id);
    if (($post->user->id != $request->user()->id) &&
    !$request->user()->isAdmin()) {
        abort_unless(false, 401);
    }
    return view('posts/edit', [
        'post' => $post
    ]);
}
```

Ahora crearemos la plantilla para poder editar creando un nuevo archivo:
resources/views/posts/edit.blade.php :

```
@extends('..layouts.app')
```



```

@section('content')
<section class="w-full bg-gray-200 py-4 flex-row justify-center text-center">
  <div class="flex justify-center">
    <div class="max-w-4xl">
      <h1 class="px-4 text-6xl break-words">Edit Post</h1>
    </div>
  </div>
</section>
<article class="w-full py-8">
  <div class="flex justify-center">
    <div class="max-w-7xl text-justify">
      <form action="{{ route('posts.update', $post) }}" method="post">
        @csrf
        @method('PUT')
        <input class="w-full border rounded focus:outline-none
focus:shadow-outline p-2 mb-4" type="text" name="title" value="{{ $post->title
}}" placeholder="Write the title of the post">
        <textarea class="w-full h-72 resize-none border rounded
focus:outline-none focus:shadow-outline p-2 mb-4" name="body"
placeholder="Write your post here" required>{{ $post->body }}</textarea>
        <div class="mb-4">
          <input type="hidden" name="is_draft" value="0">
          @if (!$post->is_draft)
            <input type="checkbox" name="is_draft" value="1">
          @else
            <input type="checkbox" name="is_draft" value="1"
checked>
          @endif
          Is draft?
        </div>
        <input type="submit" value="SEND" class="px-4 py-2
bg-orange-300 cursor-pointer hover:bg-orange-500 font-bold w-full border
rounded border-orange-300 hover:border-orange-500 text-white">
        @if (session('status'))
          <div class="w-full bg-green-500 p-2 text-center my-2
text-white">
            {{ session('status') }}
          </div>
        @endif
        @if($errors->any())
          <div class="w-full bg-red-500 p-2 text-center my-2
text-white">

```

```

        {{$errors->first()}}
    </div>
    @endif
</form>
</div>
</div>
</article>
@endsection

```

Ahora necesitamos crear la acción update, volvemos a *postcontroller* y modificamos la función update:

```

public function update(PostRequest $request, $id)
{
    $request->validated();
    $request->user()->authorizeRoles(['is_staff', 'is_admin']);
    $post = Post::find($id);
    if (($post->user->id != $request->user()->id) &&
    !$request->user()->isAdmin()) {
        abort_unless(false, 401);
    }

    $post->title = $request->input('title');
    $post->body = $request->input('body');
    $post->is_draft = $request->input('is_draft');

    $res = $post->save();

    if ($res) {
        return back()->with('status', 'Post has been updated
sucessfully');
    }

    return back()->withErrors(['msg', 'There was an error updating the
post, please try again later']);
}

```

Ahora ya podemos editar los posts:

Mi Blog de Laravel

Edit Post

Write the title of the post

Write your post here

☐ Is draft?

SEND

Por último necesitamos la función de borrar posts. En el archivo *postcontroller* de nuevo modificamos el método *destroy*:

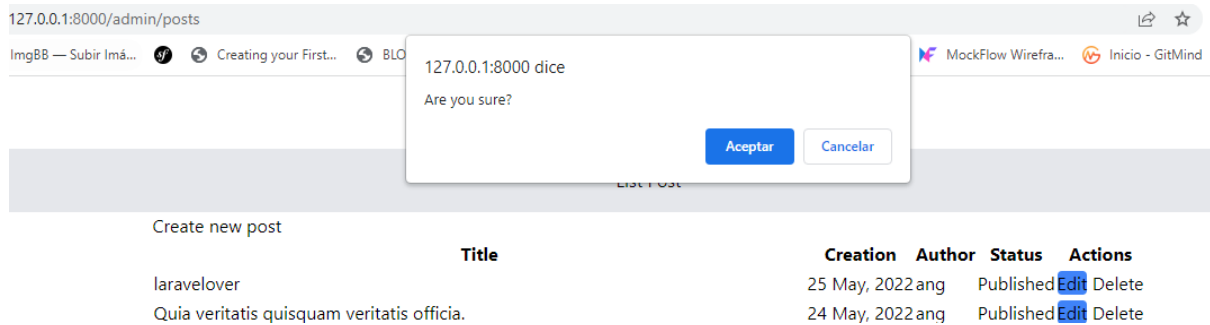
```
public function destroy(Request $request, $id)
{
    abort_unless(Auth::check(), 404);
    $request->user()->authorizeRoles(['is_staff', 'is_admin']);
    $post = Post::where('id', $id)->first();
    if (($post->user->id != $request->user()->id) &&
    !$request->user()->isAdmin()) {
        abort_unless(false, 401);
    }

    $post->delete();

    return back()->with('status', 'Post has been deleted
    sucessfully');
}
```

De esta forma ya podemos borrar

Create new post					
	Title	Creation	Author	Status	Actions
	laravelover	25 May, 2022 ang	Published	Edit	Delete
	Quia veritatis quisquam veritatis officia.	24 May, 2022 ang	Published	Edit	Delete



Por último vamos a implementar los tests unitarios:

Son scripts que generamos para verificar el correcto funcionamiento de un fragmento de nuestro código.

Para poder trabajar con ellos vamos a crear una nueva base de datos que la vamos a llamar *laravel-tailwind-2* creamos a continuación el archivo *.env.testing*

A continuación modificamos el archivo *database/factories/UserFactory.php*. modificamos el método *definition()* para que ahora retorne este array:

```
public function definition()
{
    return [
        'name' => $this->faker->name,
        'email' => $this->faker->unique()->safeEmail,
        'email_verified_at' => now(),
        'password' =>
'$2y$10$92IXUNpkj00r0Q5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', //
password
        'remember_token' => Str::random(10),
        'is_admin' => true,
        'is_staff' => true,
    ];
}
```

A continuación vamos a crear el test de los comentarios con el siguiente comando:

```
php artisan make:test CommentTest
```

```
C:\laragon\www\laravel-tailwind
λ php artisan make:test CommentTest
Test created successfully.
```

Modificamos el archivo *CommentTest.php*:

```
<?php

namespace Tests\Feature;

use Illuminate\Foundation\Testing\RefreshDatabase;
use Illuminate\Foundation\Testing\WithFaker;
use Tests\TestCase;
use App\Models\User;
use App\Models\Post;
use App\Models\Comment;

class CommentTest extends TestCase
{
    /**
     * A basic feature test example.
     *
     * @return void
     */
    public function testCreateComment()
    {
        $user = User::factory()->create();
        $test_post = [
            'title' => 'My test post with comment',
            'body' => 'This is a test functional post',
            'is_draft' => false
        ];
        $response = $this->actingAs($user)->post('/admin/posts', $test_post);
        $response->assertSessionHas('status', 'Post has been created
sucessfully');

        $post = Post::where('title', $test_post['title'])->first();

        $comment = 'This is a test comment';
        $test_comment = [
            'post_id' => $post->id,
            'comment' => $comment
        ];
        $response = $this->actingAs($user)->post('/comment', $test_comment);
        $response->assertSessionHas('status', 'Comment has been created
sucessfully');
```

```

        $comment = Comment::where('user_id', $user->id)
        ->where('post_id', $post->id)
        ->where('comment', $comment)->first();

        $this->assertNotNull($comment);

        $this->post('/logout');

        // Ahora probamos con un usuario sin loguear
        $response = $this->post('/comment', $test_comment);
        $response->assertStatus(403);
    }
}

```

Ejecutamos ahora el comando *php artisan test*. Además creamos el archivo `PostTest` con el siguiente comando: `php artisan make:test PostTest` y le damos forma con el siguiente código:

```
<?php
```

```

namespace Tests\Feature;

use Illuminate\Foundation\Testing\RefreshDatabase;
use Illuminate\Foundation\Testing\WithFaker;
use Tests\TestCase;
use App\Models\User;
use App\Models\Post;
use Cviebrock\EloquentSluggable\Services\SlugService;

class PostTest extends TestCase
{
    use RefreshDatabase;

    /**
     * A basic feature test example.
     *
     * @return void
     */
    public function testMainPage()
    {
        $response = $this->get('/');

        $response->assertStatus(200);
    }
}

```

```

/**
 * Testing create posts and checked if exists and can be visualized
 *
 * @return void
 */
public function testCreatePost()
{
    $user = User::factory()->create();
    $title = 'My test post';
    $response = $this->actingAs($user)->post('/admin/posts', [
        'title' => $title,
        'body' => 'This is a test functional post',
        'is_draft' => false
    ]);
    $response->assertSessionHas('status', 'Post has been created
sucessfully');
    $response->assertRedirect();

    // Con unique == false no revisa en la base posts y entonces no se
raya con que sea un duplicado
    $slug_url = SlugService::createSlug(Post::class, 'slug', $title,
['unique' => false]);

    $response = $this->get('/posts/' . $slug_url);
    $response->assertStatus(200);

    $title .= '2';
    $response = $this->actingAs($user)->post('/admin/posts', [
        'title' => $title,
        'body' => 'This is a test functional post',
        'is_draft' => true
    ]);
    $response->assertSessionHas('status', 'Post has been created
sucessfully');
    $response->assertRedirect();

    // Con unique == false no revisa en la base posts y entonces no se
raya con que sea un duplicado
    $slug_url = SlugService::createSlug(Post::class, 'slug', $title,
['unique' => false]);

    $response = $this->get('/posts/' . $slug_url);
    $response->assertStatus(404);

    $response = $this->actingAs($user)->post('/admin/posts', []);
    $response->assertSessionHasErrors([

```

```

        'title' => 'A title is required',
        'body' => 'You must sent a body',
        'is_draft' => 'You must sent if is draft or not'
    ]);
}

/**
 * Testing create posts and checked if exists and can be visualed
 *
 * @return void
 */
public function testEditPost()
{
    $user = User::factory()->create();
    $test_post = [
        'title' => 'test edit post',
        'body' => 'This is a test functional post',
        'is_draft' => false
    ];
    $response = $this->actingAs($user)->post('/admin/posts',
    $test_post);
    $response->assertSessionHas('status', 'Post has been created
    sucessfully');

    $test_post_update = [
        'title' => 'test post update',
        'body' => 'This is a test functional post update',
        'is_draft' => true
    ];
    // Como usamos el RefreshDatabase elimina los datos en cada test
    así que es el 1 si o si
    $response = $this->actingAs($user)->put('admin/posts/1',
    $test_post_update);
    $response->assertSessionHas('status', 'Post has been updated
    sucessfully');

    $ddbb_post = Post::find(1);

    $this->assertEquals($ddbb_post['title'],
    $test_post_update['title']);
    $this->assertEquals($ddbb_post['body'],
    $test_post_update['body']);
    $this->assertEquals($ddbb_post['is_draft'],
    $test_post_update['is_draft']);
}

/**

```



```

    * Testing create posts and checked if exists and can be visualized
    *
    * @return void
    */
    public function testDeletePost()
    {
        $user = User::factory()->create();
        $test_post = [
            'title' => 'test post delete',
            'body' => 'This is a test functional post',
            'is_draft' => false
        ];
        $response = $this->actingAs($user)->post('/admin/posts',
        $test_post);
        $response->assertSessionHas('status', 'Post has been created
        sucessfully');

        $post = Post::where('title', $test_post['title'])->first();

        $response = $this->actingAs($user)->delete('admin/posts/' .
        $post->id);
        $response->assertSessionHas('status', 'Post has been deleted
        sucessfully');

        $ddbb_post = Post::find($post->id);

        $this->assertNull($ddbb_post);
    }

    public function testEditPostStaffMember()
    {
        $user_admin = User::factory()->create();
        $test_post = [
            'title' => 'test post admin',
            'body' => 'This is a test functional post',
            'is_draft' => false
        ];
        $response = $this->actingAs($user_admin)->post('/admin/posts',
        $test_post);
        $response->assertSessionHas('status', 'Post has been created
        sucessfully');

        $post = Post::where('title', $test_post['title'])->first();

        $user_staff = $this->createStaffUser();

        $test_post_update = [

```

```

        'title' => 'test post update',
        'body' => 'This is a test functional post update',
        'is_draft' => true
    ];
    $response = $this->actingAs($user_staff)->put('admin/posts/' .
$post->id, $test_post_update);
    $response->assertStatus(401);

    $ddbb_post = Post::find($post->id);

    $this->assertEquals($ddbb_post['title'], $test_post['title']);
    $this->assertEquals($ddbb_post['body'], $test_post['body']);
    $this->assertEquals($ddbb_post['is_draft'],
$test_post['is_draft']);
    }

    public function testDeletePostStaffMember()
    {
        $user_admin = User::factory()->create();
        $test_post = [
            'title' => 'test post delete admin',
            'body' => 'This is a test functional post',
            'is_draft' => false
        ];
        $response = $this->actingAs($user_admin)->post('/admin/posts',
$test_post);
        $response->assertSessionHas('status', 'Post has been created
sucessfully');

        $post = Post::where('title', $test_post['title'])->first();

        $user_staff = $this->createStaffUser();

        $post = Post::where('title', $test_post['title'])->first();

        $response = $this->actingAs($user_staff)->delete('admin/posts/' .
$post->id);

        $ddbb_post = Post::find($post->id);

        $this->assertNotNull($ddbb_post);
        $response->assertStatus(401);
    }

    private function createStaffUser()
    {
        $user_staff = new User;
    }

```

```

        $user_staff->name = rand(1, 99) . 'Staff user';
        $user_staff->email = rand(1, 99) . 'staff@cosasdedevs.com';
        $user_staff->password = bcrypt('123456');
        $user_staff->is_staff = true;
        $user_staff->save();
        return $user_staff;
    }
}

```

Por último damos estilo en el archivo *app.css* el resultado es el siguiente:



←→↻127.0.0.1:8000

Aplicaciones Subir Imá...Creating your First...BLOG SIMPortal do Emprega...GitHubHow to Create a M...MockFlow Wirefra...Inicio - GitMindCalendário - JS»

HOMEang👤🔗

Mi Blog de Laravel

About me

Lorem ipsum dolor sit, amet consectetur adipisicing elit. Voluptate necessitatibus ullam commodi perferendis accusamus sint error sequi, dolorem nam, vel praesentium dignissimos nostrum quod fuga corporis asperiores laudantium, possimus veniam! Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consectetur iure cumque qui impedit quod earum dolores nisi nemo totam vero natus aperiam, libero consequuntur nesciunt atque officia exercitationem rerum. Veritatis! Lorem ipsum dolor sit amet consectetur adipisicing elit. Voluptas in hic ratione recusandae nostrum, saepe aliquam alias ipsum? Asperiores rerum numquam officia harum atque, impedit perspiciatis facilis nobis tempora est!

Lasts posts

laravelover

inicios laravel... **Read more**

Quia veritatis quisquam veritatis officia.

Hic odio modi officiis accusamus non itaque atque cupiditate. Rem velit sit dolor quo sunt asperiores et. Consequatur praesentium officiis q... **Read more**

Ut tenetur delectus nihil facere.

Enim hic consequatur ex aut laudantium cumque. Harum quam voluptatem ut laudantium. Dolor fugiat aut incidunt. Dolorem et et quasi eligendi ... **Read more**

Est doloreque consequatur similique numquam sunt.

Et voluptas distinctio est temporibus. Nam minima cupiditate rerum molestias vero delectus dolor eum. Aut sed officiis repellendus. Labore p... **Read more**

Repudiandae minus neque porro et excepturi qui et.

Ea nobis perspiciatis dolor eum dicta. Quia reiciendis voluptas quam officiis recusandae quis dolores. Nisi quod consequatur et cumque minus... **Read more**

Voluptatem dolor voluptatem voluptas ratione nihil perferendis perspiciatis officia.

Totam in in consequatur inventore quos et rem doloribus. Quasi laboriosam deleniti ab. Repellat facilis facilis numquam non inventore. Qui n... **Read more**

@MyLaravelBlog By Ang

←→↻127.0.0.1:8000/admin/posts

Aplicaciones Subir Imá...Creating your First...BLOG SIMPortal do Emprega...GitHubHow to Create a M...MockFlow Wirefra...Inicio - GitMindCalendário - JS»

HOMEang👤🔗

Mi Blog de Laravel

List Post

Create new post

Title	Creation	Author	Status	Actions
laravelover	25 May, 2022	ang	Published	Edit Delete
Quia veritatis quisquam veritatis officia.	24 May, 2022	ang	Published	Edit Delete
Ut tenetur delectus nihil facere.	24 May, 2022	ang	Published	Edit Delete
Est doloreque consequatur similique numquam sunt.	24 May, 2022	ang	Published	Edit Delete
Repudiandae minus neque porro et excepturi qui et.	24 May, 2022	ang	Published	Edit Delete
Voluptatem dolor voluptatem voluptas ratione nihil perferendis perspiciatis officia.	24 May, 2022	ang	Published	Edit Delete
Et quas autem porro ad beatae.	24 May, 2022	ang	Published	Edit Delete
Aut quia et ea non dolorem vitae nesciunt.	24 May, 2022	ang	Published	Edit Delete
Temporibus quia voluptatem itaque fuga aut.	24 May, 2022	ang	Published	Edit Delete
Omnis adipisci qui dolorum amet veniam.	24 May, 2022	ang	Published	Edit Delete
Harum aut rerum asperiores aspernatur.	24 May, 2022	ang	Published	Edit Delete
Quisquam deleniti facere natus odio accusantium voluptatem eos.	24 May, 2022	ang	Published	Edit Delete
Expedita nam ipsa earum totam soluta.	24 May, 2022	ang	Published	Edit Delete
Facilis ut velit rerum ut aut blanditiis nihil.	24 May, 2022	ang	Published	Edit Delete
Dolor dolorum cumque quis deserunt.	24 May, 2022	ang	Published	Edit Delete
Dolorem suscipit quos dolore quisquam nobis facere.	24 May, 2022	ang	Published	Edit Delete
Iusto et voluptates corrupti minus eos quo.	24 May, 2022	ang	Published	Edit Delete
Omnis doloreque distinctio quo possimus possimus.	24 May, 2022	ang	Published	Edit Delete
Nisi ad beatae et tempore vero corporis.	24 May, 2022	ang	Published	Edit Delete
Ut blanditiis ex corporis vel dignissimos.	24 May, 2022	ang	Published	Edit Delete
Ad error est neque odio voluptatem et est.	24 May, 2022	ang	Published	Edit Delete
Qui vero nobis accusantium ullam blanditiis.	24 May, 2022	ang	Published	Edit Delete
Existe facere repudiandae dicta at	24 May, 2022	ang	Published	Edit Delete

← → ↻ 127.0.0.1:8000/posts/laravelover

Aplicaciones ImgBB — Subir Imá... Creating your First... BLOG SIM Portal do Emprega... GitHub How to Create a M... MockFlow Wirefra... Inicio - GitMind Calendário - JS »

El último leído

HOME ang 👤 ➔

Mi Blog de Laravel

laravelover

inicios laravel

Comments

Write your comment here

SEND

Comment has been created sucessfully

By ang

25 May, 2022

comentario

@MyLaravelBlog By Ang