Práctica Blog Paso a Paso

Desarrollo guiado de la práctica de Blog Personal

Descripción del proyecto

Se trata de un blog personal en el que los usuarios registrados podrán escribir y administrar sus publicaciones.

Las publicaciones tienen un título, resumen, cuerpo, fecha de publicación y el usuario que la creó.

Todos los usuarios y visitantes (no registrados) podrán ver las publicaciones en la página inicial. En esta, solo se verán las publicaciones que ya cumplieron su fecha de publicación.

Las publicaciones se mostrarán en orden descendente de fecha de publicación. Estas publicaciones se verán en dos formatos distintos según sean más recientes o no (en fecha de publicación).

- Para las 5 primeras publicaciones:
    - Fecha de publicación
    - Título
    - Resumen
    - Usuario
    - Botón “leer más”
- Para las 20 siguientes publicaciones:
    - Fecha de publicación
    - Título (es enlace a leer más)
    - Usuario

Una vez pulsado leer más se podrá ver la publicación al completo con todos sus datos.

Los usuarios registrados podrán votar las publicaciones dando un “me gusta”, los que no han iniciado sesión no podrán hacerlo.

En la página principal y en la lectura de cada publicación se mostrará su puntuación.

En la página de lectura es donde debe aparecer el botón que permite votar la publicación.

Planteamiento del proyecto

Se va a usar *Laravel* junto con *Breeze* para la gestión de autenticación del proyecto.

A continuación se describen cada una de los pasos a seguir para realizar la práctica.

Hay que tener en cuenta que al usar *Tailwind*, el código parece ser mucho más complicado de lo que realmente es.

Aquí tienes el repositorio por si se me ha escapado algún código:

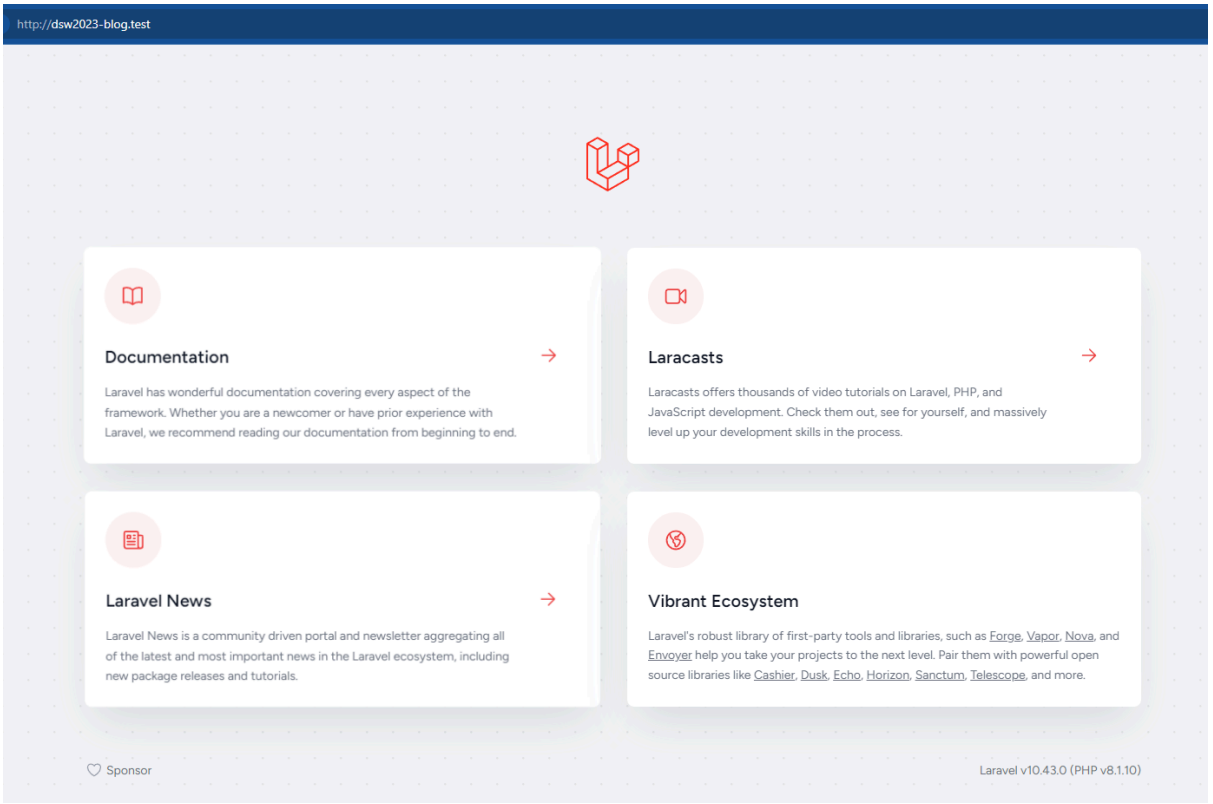
https://github.com/acrerosj/DSW2023-blog.git

1. Iniciar el proyecto.

1.1. Crear el sitio en Laravel.

Para ello usamos la herramienta de creación de sitios rápidos de Laragon.

Comprobamos que el sitio se ha creado y funciona la URL.



1.2. Variables de entorno

Cambiamos en el archivo *.env*: DB_DATABASE=**blog**

2. Autenticación de Usuarios

Para toda la parte de registro, inicio de sesión y gestión del perfíl vamos a usar el ecosistema de Laravel, porque funciona y no tenemos que reinventar la rueda.

2.1. Instalación de Breeze-Blade

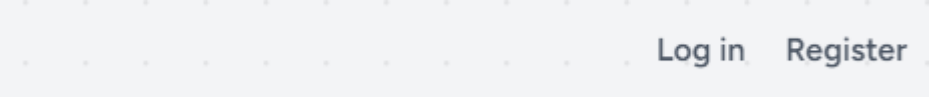
En concreto vamos a usar el paquete Breeze.

https://laravel.com/docs/10.x/starter-kits#laravel-breeze-installation

Resumiendo:

```
➢ composer require laravel/breeze --dev
➢ php artisan breeze:install blade
➢ php artisan migrate
➢ npm install
➢ npm run dev
```

Con esto ya tendremos instalada toda la funcionalidad para el registro y la autenticación de usuarios. Para comprobarlo tenemos que fijarnos que aparece en la parte superior a la derecha las opciones de login y registro.



Puedes registrarte y crear algunos usuarios, pero lo vamos a hacer más rápido mediante una semilla de usuarios de prueba.

2.2. Crear usuarios de prueba

https://laravel.com/docs/10.x/seeding

Para poblar la tabla de usuarios con datos de prueba vamos a crear un archivo de semillas (seeder) para la tabla User.

```
➢ php artisan make:seeder UserSeeder
```

Con esto se crea el archivo *database\seeders\UserSeeder.php*.

Como los usuarios solo necesitan obligatoriamente el *name*, *email* y *password* para crearse, vamos a crear un array con estos datos y luego lo insertamos en el modelo.

Hay que tener en cuenta que Breeze guarda la contraseña encriptada, como debe ser por seguridad, por lo que tenemos que encriptarla antes de insertarla. Para ello usaremos use *Illuminate\Support\Facades\Hash* como se indica en el código.

```php
namespace Database\Seeders;

use App\Models\User;
use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\Hash;

class UserSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        $users = [
            ['name' => 'pepe', 'email' => 'pepe@pepe.es', 'password' => Hash::make('123')],
            ['name' => 'marta', 'email' => 'marta@marta.es', 'password' => Hash::make('123')],
```

```
            ['name' => 'ana', 'email' => 'ana@ana.es', 'password' => Hash::make('123')],
            ['name' => 'juan', 'email' => 'juan@juan.es', 'password' => Hash::make('123')],
            ['name' => 'laura', 'email' => 'laura@laura.es', 'password' => Hash::make('123')],
            ['name' => 'carlos', 'email' => 'carlos@carlos.es', 'password' =>
Hash::make('123')],
            ['name' => 'maria', 'email' => 'maria@maria.es', 'password' => Hash::make('123')],
            ['name' => 'pedro', 'email' => 'pedro@pedro.es', 'password' => Hash::make('123')],
            ['name' => 'luisa', 'email' => 'luisa@luisa.es', 'password' => Hash::make('123')],
            ['name' => 'javier', 'email' => 'javier@javier.es', 'password' =>
Hash::make('123')],
        ];

        User::insert($users);
    }
}
```

Para lanzar la semilla y que se creen los usuarios:

```
➢ php artisan db:seed UserSeeder
```

2.2.1. UserSeeder en DatabaseSeeder

También es recomendable indicar en el archivo *DatabaseSeeder.php* que se debe lanzar *UserSeeder*. Esto se hace con el siguiente código.

```
<?php

namespace Database\Seeders;

// use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     */
    public function run(): void
    {
        // \App\Models\User::factory(10)->create();

        // \App\Models\User::factory()->create([
        //     'name' => 'Test User',
        //     'email' => 'test@example.com',
        // ]);
        $this->call([
```

```
            UserSeeder::class,
            PostSeeder::class,
        ]);
    }
}
```

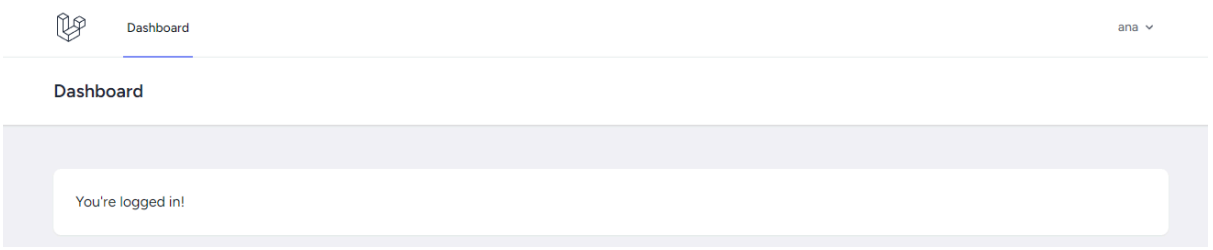
De esta forma, si tenemos que volver a lanzar toda la migración, por ejemplo si importamos el proyecto en otro equipo, podemos poblar al mismo tiempo que creamos las tablas.

```
➢ php artisan migrate:fresh --seed
```

Este comando elimina y vuelve a crear todas las tablas y además lanza *DatabaseSeeder* que a su vez lanza los archivos seeders que hayamos declarado.

2.2.2. Probar usuarios

Ya se puede comprobar que podemos entrar con los usuarios creados.



También podemos crear algún otro usuario mediante el registro para probar su uso.

3. CRUD de Publicaciones

3.1. Crear archivos necesarios

Vamos a crear el CRUD completo para publicaciones. Para ello vamos a tener que crear los siguientes archivos:

- **Modelo** para indicar los campos editables y la relación con los usuarios.
- **Controlador** para crear todos los métodos que hacen el CRUD.
- **Migración** para crear la tabla y los campos y clave foránea con el usuario.
- **Semilla** para poblar con algunos datos iniciales.

Lo mejor es hacer todos los archivos usando la herramienta Artisan en un solo comando.

```
➢ php artisan make:model Post -cmsr
```

Donde -c es controlador, -m migración, -s seeder y -r es que añade todos los métodos al controlador.

Se deben haber creado los archivos:

- app\Models\Post.php
- database\migrations/????_??_??_???????_create_posts_table.php donde las interrogaciones son año_mes_dia_horaminutossegundos.
- database\seeders\PostSeeder.php
- app\Http\Controllers\PostController.php

También vamos a crear las vistas:

- resources\views\posts\index.blade.php

Vamos a copiar el mismo contenido que dashboard.blade.php en nuestro archivo index.blade.php y luego iremos cambiando el contenido.

3.2. Ruta a publicaciones

Vamos al archivo routes\web.php y añadimos al final la siguiente ruta:

Route::resource('posts', 'App\Http\Controllers\PostController')->middleware('auth');

Con esto estamos añadiendo todas las rutas para las publicaciones.

Van a estar todas a partir de la ruta /posts y solo van a ser accesible a los usuarios autenticados.

Luego haremos otra para los visitantes no autenticados.

Podemos comprobarlo poniendo en consola:

```
➢ php artisan route:list -path=post
```

Debe devolvernos todas las rutas

3.3. Migración

https://laravel.com/docs/10.x/migrations

Analizando las necesidades de la Publicación, las descripción de las columnas es:

- **id**: clave primaria autoincremental.

- **user_id**: usuario propietario de la publicación. Es clave foránea. Vamos a poner la restricción de que toda publicación tiene que tener un usuario y que si se borra el usuario, se borran sus publicaciones.
- **title**: Título de la publicación. Un string de 255 caracteres será suficiente.
- **summary**: Resumen de la publicación. Puede tener un párrafo, por lo que lo dejaremos como tipo *text* que puede tener una longitud indeterminada pero luego se puede limitar por código. Es opcional y, por tanto, puede ser nulo.
- **body**: Contenido de la publicación. Un tipo text para que puedan publicar todo lo que quieran.
- **published_at**: Fecha de publicación.No debe llevar tiempo, solo la fecha.
- Timestamps: No son obligatorios pero recomendables.
    - **created_at**: fecha de creación del registro.
    - **updated_at**: fecha de modificación del registro.

En el archivo database\migrations/????_??_??_???????_create_posts_table.php vamos a indicar las columnas que queremos crear.

```php
Schema::create('posts', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->constrained()->onDelete('cascade');
    $table->string('title');
    $table->text('summary')->nullable();
    $table->text('body');
    $table->date('published_at');
    $table->timestamps();
});
```

Una vez creado, podemos realizar la migración.

```
➢ php artisan migrate
```

3.4. Publicaciones de prueba

Vamos a crear algunas publicaciones de prueba.

Primero comprobamos que tenemos creados usuarios y sus identificadores. Se supone que anteriormente creamos 10 usuarios con id del 1 al 10.

Para crear las publicaciones, voy a ayudarme de un Factory para generar contenido falso.

3.4.1. Definir un modelo de contenido falso: Factory

https://laravel.com/docs/10.x/eloquent-factories#sequences

```
➢ php artisan make:factory PostFactory
```

También podíamos haberlo creado en el crear el modelo añadiendo el modificador -f.

Editamos el archivo database\factories\PostFactory.

Para ayudarnos a la hora de crear el contenido falso, Laravel usa Faker. Puedes consultar la documentación con los formatos que genera en: https://fakerphp.github.io/

Voy a crearlo de la siguiente manera:

```
public function definition(): array
{
    return [
        'user_id' => fake()->numberBetween(1, 9),
        'title' => fake()->unique()->sentence(5,4),
        'summary' => fake()->paragraph(3),
        'body' => fake()->realTextBetween(200,1000,1),
        'published_at' => fake()->dateTimeBetween('-1 year', 'now'),
    ];
}
```

3.4.1. Modificar el Seeder para que use el Factory

Editamos database\seeder\PostSeeder.php y añadimos datos de prueba mediante el factory creado.

```
public function run(): void
{
    Post::factory(20)->create();
}
```

Vamos a crear 20 publicaciones para los usuarios del 1 al 9, por tanto el 10 no tendrá publicaciones.

```
➢ php artisan db:Seed PostSeeder
```

No olvidar poner el PostSeeder también en el DatabaseSeeder pero debajo del UserSeeder.

3.5. El Modelo de Publicaciones (Post)

Vamos a modificar algunos detalles del modelo que luego nos ayudaran.

Vamos a indicar qué campos son rellenables (title, summary, body, published_at). El id, user_id, created_at y updated_at no son rellenables ya que va a ser el propio sistema el que asigne el valor.

También vamos a indicar la relación que tiene con los usuarios para luego poder recuperar el usuario de la publicación.

El archivo app\Models\Post.php debe quedar así:

```php
<?php

namespace App\Models;

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

class Post extends Model
{
    use HasFactory;

    protected $fillable = [
        'title',
        'summary',
        'body',
        'published_at',
    ];

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

Podemos comprobar el correcto funcionamiento mediante tinker

- ➢ `php artisan tinker`
    - `App\Models\Post::find(1)->user->name`

      Estoy buscando el post con id 1 y de él obtener el nombre del usuario.

      Debe devolver un usuario. En mi caso es “juan”, pero al ser aleatorio la generación de datos de prueba, puede que sea distinto en tu caso.

      Para salir de tinker escribe “quit”.

3.5.1. Relación user-post

De igual manera, a los usuarios hay que añadir la función que permita obtener todos los posts que ha creado.

Por tanto, al archivo app\Models\User.php se le añade el siguiente método:

```php
public function posts() : HasMany
{
    return $this->hasMany(Post::class);
}
```

Lo comprobamos en tinker.

Ojo: Si has cambiado algún modelo, sal de tinker y vuelve a entrar o no se verán los cambios.

- `php artisan tinker`
    - `App\Models\User::find(2)->posts`

      Esto debe devolver todas las publicaciones del usuario con id 2. Quizás no tenga ninguna. Prueba con varios usuarios.

3.6. Enlace a publicaciones

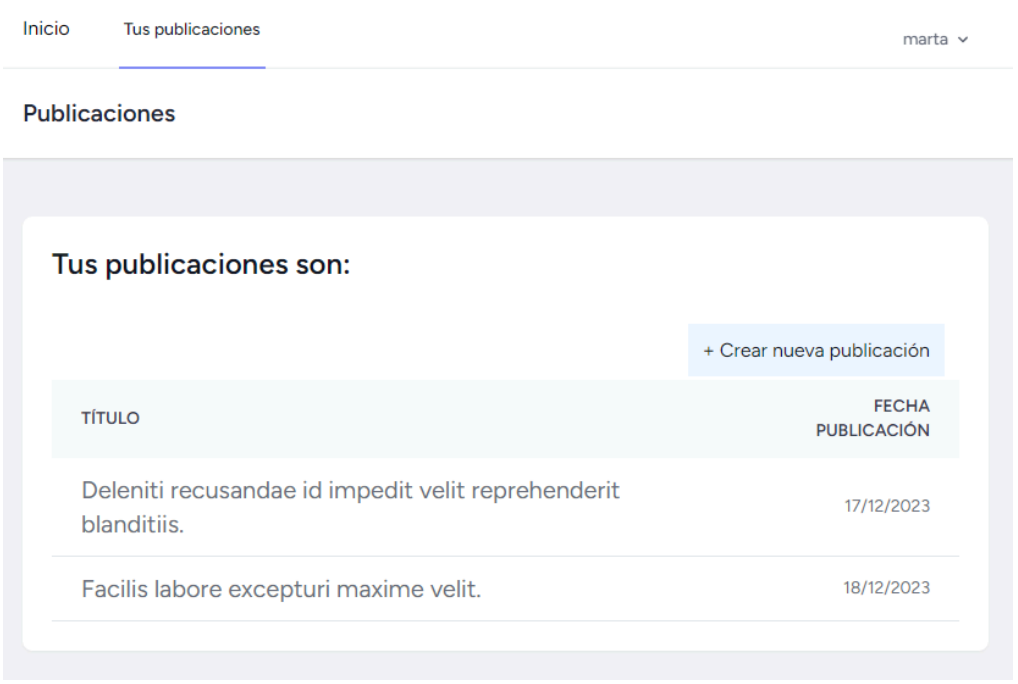
Añadimos el enlace a publicaciones sustituyendo los enlaces a dashboard por publicaciones.

Vamos a la vista layouts/navigation.blade.php y modificamos la línea 15 y 16.

```blade
<!-- Navigation Links -->
<div class="hidden space-x-8 sm:-my-px sm:ms-10 sm:flex">
    <x-nav-link :href="route('posts.index')" :active="request()->routeIs('posts.index')">
        Tus publicaciones
    </x-nav-link>
</div>
```

3.7. Listado de publicaciones.

Vamos a mostrar al usuario sus publicaciones. No todas, solo las que ha creado él.



Vamos a ir al controlador app\Http\Controllers\PostController.php, y en el método index, vamos a buscar y mostrar las publicaciones del usuario.

3.7.1. Usuario Autenticado

¿Cómo podemos obtener el usuario autenticado?

Existen varias forma que puedes consultar en: https://laravel.com/docs/10.x/authentication

Vamos a usar: `$user = Auth::user();`

Par la cual tendrás que usar: `use Illuminate\Support\Facades\Auth;`

Podemos hacer una prueba rápida con el siguiente código:

```
public function index()
{
    $user = Auth::user();
    echo $user;
    echo "<hr>";
    echo $user->posts;
}
```

Se verá el usuario y a continuación sus posts. Aunque no muy bonito.

Ahora vamos a llamar a la vista y pasarle los posts.

```
public function index()
{
```

```
        $posts = Auth::user()->posts;
        return view('posts.index', compact('posts'));
    }
```

Y modificamos la vista posts.index añadiendo los textos, el enlace para crear una nueva publicación, la tabla con las publicaciones (solo el título y la fecha de publicación) y el título que sea enlazable para mostrar toda la publicación.

Pongo en negrita lo más destacado porque el resto es la plantilla dashboard que copiamos y mucho formato de Tailwind.

```
<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 leading-tight">
            {{ __('Publicaciones') }}
        </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">
                    <h2 class="font-semibold text-2xl text-black-900 leading-tight">
                        Tus publicaciones son:
                    </h2>
                    <div class="relative overflow-x-auto mt-8">
                        <div class="text-right m-3">
                            <a class="bg-blue-50 p-3"
                                href="{{ route('posts.create')}}">
                                + Crear nueva publicación
                            </a>
                        </div>
                        <table class="w-full text-sm text-left rtl:text-right text-gray-500
dark:text-gray-400">
                            <thead class="text-gray-700 uppercase bg-gray-50 dark:bg-gray-700
dark:text-gray-400">
                                <tr>
                                    <th scope="col" class="px-6 py-3">Título</th>
                                    <th scope="col"
                                       class="px-6 py-3 text-right">Fecha Publicación</th>
                                </tr>
                            </thead>
                            <tbody>
                                @foreach ($posts as $post)
                                <tr class="bg-white border-b dark:bg-gray-800
dark:border-gray-700">
                                    <td class="px-6 py-3 text-xl">
                                        <a href="{{ route('posts.show', $post) }}">
                                            {{ $post->title }}
                                        </a>
                                    </td>
                                    <td class="px-6 py-3 text-right">
```

```
                                        {{
\Carbon\Carbon::parse($post->published_at)->format('d/m/Y') }}
                                    </td>
                                </tr>
                                @endforeach
                            </tbody>
                        </table>
                    </div>
                </div>
            </div>
        </div>
    </div>
</x-app-layout>
```

Ahora ya está implementado que cada usuario vea solo sus publicaciones.

Puedes probarlo con el path /***posts***.

Luego añadiremos que todos, tanto usuarios registrados como los visitantes puedan ver las publicaciones en la página de inicio.

Usamos ***Carbon*** para dar formato a la fecha de publicación.

3.8. Mostrar publicación

Vamos a mostrar en detalle una publicación. A esta página se llega pulsando el enlace del título de cada publicación.

La url para acceder a mostrar el post es /posts/?, siendo ? el identificador del post.

En el controlador se pone el siguiente código en el método show:

```
public function show(Post $post)
{
    return view('posts.show', compact('post'));
}
```

Creamos la vista posts/show.blade.php como una copia de posts/index.blade.php cambiando el contenido. Vamos a mostrar la fecha de publicación, el título, el resumen y el body. El usuario no hay que mostrarlo porque siempre debería ser del propio usuario que inició la sesión. Aunque si se cambia la url a mano y se cambia el post_id en la URL, un usuario puede ver el post de otro usuario.

También añadimos los botones de “volver” al listado, “editar” la publicación y “borrar”.

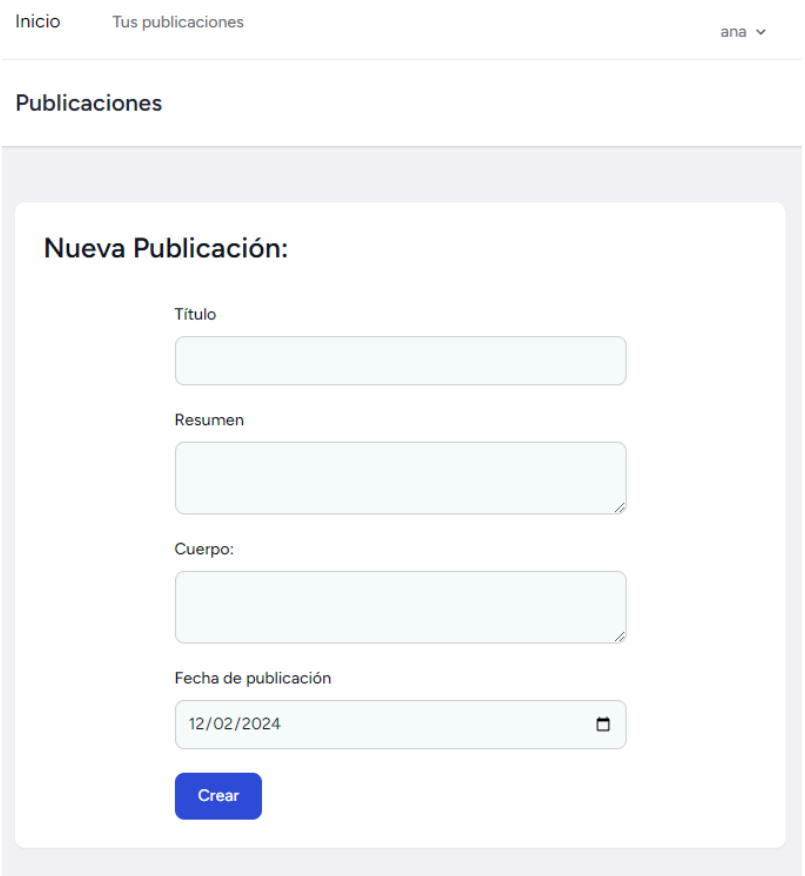
```
<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 leading-tight">
            {{ __('Publicaciones') }}
        </h2>
    </x-slot>

    <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">
                    <p class="text-sm text-right">
                        {{ \Carbon\Carbon::create($post->published_at)->format('d/m/Y') }}
                    </p>
                    <h2 class="font-semibold text-2xl text-black-900 leading-tight m-3">
                        {{ $post->title}}
                    </h2>
                    <p class="italic m-3 text-gray-800 font-semibold">
                        {{ $post->summary }}
                    </p>
                    <p class="m-3 text-sm ">
                        {{ $post->body }}
                    </p>
                    <div class="grid grid-cols-6">
                        <div class="col-start-1 col-span-1 m-3">
                            <a href="{{ route('posts.index') }}"
                            class="p-3 bg-blue-200 hover:bg-blue-500">Volver</a>
                        </div>
                        @if (auth()->user()->id == $post->user_id )
                            <div class="col-start-5 col-span-1 text-right m-3">
                                <a href="{{ route('posts.edit', $post) }}"
                                class="p-3 bg-green-200 hover:bg-green-500">Editar</a>
                            </div>
                            <div class="col-start-6 col-span-1 text-right m-0 p-0">
                                <form action="{{ route('posts.destroy', $post) }}"
                                    method="POST" class="m-0 p-0">
                                    @csrf
```

```
                                @method('DELETE')
                                <input type="submit" value="Borrar"
                                class="bg-red-200 hover:bg-red-500 p-3" />
                            </form>
                        </div>
                    @endif
                </div>
            </div>
        </div>
    </div>
</x-app-layout>
```

Está marcado los elementos destacados.

Fijate en el condicional **if** marcado que es para evitar que se muestren los botones editar y borrar para los usuarios que no son los que han creado el post.

3.9. Crear publicación



Lo primero es el método create del controlador.

```
    public function create()
    {
        return view('posts.create');
```

```
}
```

Creamos una copia de la plantilla anterior y la guardamos como create.blade.php.

Creamos el formulario y añadimos los campos. El campo de usuario no es necesario porque se obtiene de la sesión.

También se añaden los textos de error para cada campo.

```
<x-app-layout>
  <x-slot name="header">
    <h2 class="font-semibold text-xl text-gray-800 leading-tight">
      {{ __('Publicaciones') }}
    </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">
          <h2 class="font-semibold text-2xl text-black-900 leading-tight">
            Nueva Publicación:
          </h2>
          <div class="relative overflow-x-auto mt-8">
            <form class="max-w-sm mx-auto" method="post" action="{{ route('posts.store') }}">
              @csrf
              <div class="mb-5">
                <label for="title" class="block mb-2 text-sm font-medium text-gray-900
                  dark:text-white">Título</label>
                <input type="text" name="title" id="title" value="{{ old('title') }}"
                  class="bg-gray-50 border border-gray-300 text-gray-900 text-sm rounded-lg
                         focus:ring-blue-500 focus:border-blue-500 block w-full p-2.5
                         dark:bg-gray-700 dark:border-gray-600 dark:placeholder-gray-400
                         dark:text-white dark:focus:ring-blue-500 dark:focus:border-blue-500
                      @error('title') border-red-600 @enderror" required>
                @error('title')
                  <div class="text-sm bg-red-200 px-3 py-1 rounded-lg">
                    {{ $message }}
                  </div>
                @enderror
              </div>
              <div class="mb-5">
                <label for="summary"
                  class="block mb-2 text-sm font-medium text-gray-900 dark:text-white">
                    Resumen</label>
                <textarea name="summary" id="summary" value="{{ old('summary') }}"
                  class="bg-gray-50 border border-gray-300 text-gray-900 text-sm rounded-lg
                      focus:ring-blue-500 focus:border-blue-500 block w-full p-2.5
                      dark:bg-gray-700 dark:border-gray-600 dark:placeholder-gray-400
                      dark:text-white dark:focus:ring-blue-500 dark:focus:border-blue-500
                      @error('summary') border-red-600 @enderror"></textarea>
```

```
                @error('summary')
                  <div class="text-sm bg-red-200 px-3 py-1 rounded-lg">
                    {{ $message }}
                  </div>
                @enderror
              </div>
              <div class="mb-5">
                <label for="body"
                  class="block mb-2 text-sm font-medium text-gray-900 dark:text-white">
                    Cuerpo:</label>
                <textarea name="body" id="body" value="{{ old('body') }}"
                  class="bg-gray-50 border border-gray-300 text-gray-900 text-sm rounded-lg
                      focus:ring-blue-500 focus:border-blue-500 block w-full p-2.5
                      dark:bg-gray-700 dark:border-gray-600 dark:placeholder-gray-400
                      dark:text-white dark:focus:ring-blue-500 dark:focus:border-blue-500
                      @error('body') border-red-600 @enderror" required></textarea>
                @error('body')
                  <div class="text-sm bg-red-200 px-3 py-1 rounded-lg">
                    {{ $message }}
                  </div>
                @enderror
              </div>
              <div class="mb-5">
                <label for="published_at"
                  class="block mb-2 text-sm font-medium text-gray-900 dark:text-white"
                  >Fecha de publicación</label>
                <input type="date" name="published_at" id="published_at"
                   value="{{ old('published_at', date('Y-m-d')) }}"
                  class="bg-gray-50 border border-gray-300 text-gray-900 text-sm rounded-lg
                      focus:ring-blue-500 focus:border-blue-500 block w-full p-2.5
                       dark:bg-gray-700 dark:border-gray-600 dark:placeholder-gray-400
                       dark:text-white dark:focus:ring-blue-500 dark:focus:border-blue-500
                      @error('published_at') border-red-600 @enderror" required>
                @error('published_at')
                  <div class="text-sm bg-red-200 px-3 py-1 rounded-lg">
                    {{ $message }}
                  </div>
                @enderror
              </div>
              <button type="submit"
                class="text-white bg-blue-700 hover:bg-blue-800 focus:ring-4 focus:outline-none
                    focus:ring-blue-300 font-medium rounded-lg text-sm w-full sm:w-auto px-5
                     py-2.5 text-center dark:bg-blue-600 dark:hover:bg-blue-700
                     dark:focus:ring-blue-800"
                >Crear</button>
            </form>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

```
</x-app-layout>
```

El método store queda de la forma:

```
  public function store(Request $request)
  {
    $validated = $request->validate([
      'title' => 'required|unique:posts|min:3|max:255',
      'summary' => 'max:2000',
      'body' => 'required',
      'published_at' => 'required|date',
    ]);

    $post = new Post();
    $post->user_id = Auth::id();
    $post->title = $request->title;
    $post->summary = $request->summary;
    $post->body = $request->body;
    $post->published_at = $request->published_at;
    $post->save();

    return redirect()->route('posts.index')
                ->with('success', 'Publicación creada correctamente');
  }
```

Cabe destacar la validación que hacemos de los campos. Por otro lado, el user_id lo tomamos de la autenticación del usuario.

Una vez creado, volvemos a la lista de publicaciones del usuario mostrando el mensaje de confirmación.

Para mostrar el mensaje de confirmación debemos añadir una división que lo muestre en posts/index.blade.php.

```
<div class="p-6 text-gray-900">
    @if (session('success'))
        <div class="p-4 mb-4 text-sm text-green-800 rounded-lg bg-green-50
            dark:bg-gray-800 dark:text-green-400" role="alert">
            <span class="font-medium">¡Ok! </span>{{ session('success') }}
        </div>
    @endif
    @if (session('error'))
        <div class="p-4 mb-4 text-sm text-red-800 rounded-lg bg-red-50
            dark:bg-gray-800 dark:text-red-400" role="alert">
            <span class="font-medium">¡Error! </span>{{ session('error') }}
        </div>
    @endif
    <h2 class="font-semibold text-2xl text-black-900 leading-tight">
        Tus publicaciones son:
    </h2>
```

3.10. Borrar publicación

Para borrar un publicación, es bastante fácil porque ya lo tenemos el formulario en la vista posts/show.blade.php con el siguiente formulario:

```
<form action="{{ route('posts.destroy', $post) }}"
      method="POST" class="m-0 p-0">
      @csrf
      @method('DELETE')
      <input type="submit" value="Borrar"
             class="bg-red-200 hover:bg-red-500 p-3" />
</form>
```

El método en el controlador debe quedar así:

```
public function destroy(Post $post)
{
  $post->delete();
  return redirect()->route('posts.index')
          ->with('success', 'Publicación eliminada correctamente');
}
```

Pero, ¿has pensado que si un usuario malicioso edita el HTML y cambia el id de la URL y trata de borrar una publicación que no es suya?

Para evitar que el método acepte que un usuario elimine una publicación que no le pertenece, podemos añadir un condicional en el propio método.

```
public function destroy(Post $post)
{
  if ($post->user_id == Auth::id()) {
    $post->delete();
    return redirect()->route('posts.index')
            ->with('success', 'Publicación eliminada correctamente.');
  } else {
    return redirect()->route('posts.index')
            ->with('error', 'No puedes eliminar una publicación de la que no eres el autor.');
  }
}
```

De esta forma se añade un extra de seguridad,

3.11. Actualizar publicación

Para la creación de la publicación debemos usar los métodos ***edit*** y ***update***. Pero también le vamos a añadir, por lo menos al método ***update***, la misma comprobación de usuario que al método ***destroy***.

La vista de *posts/edit.blade.php* es muy parecida a la de *create.blade.php*.

No la explicaremos.

4. Listado Principal

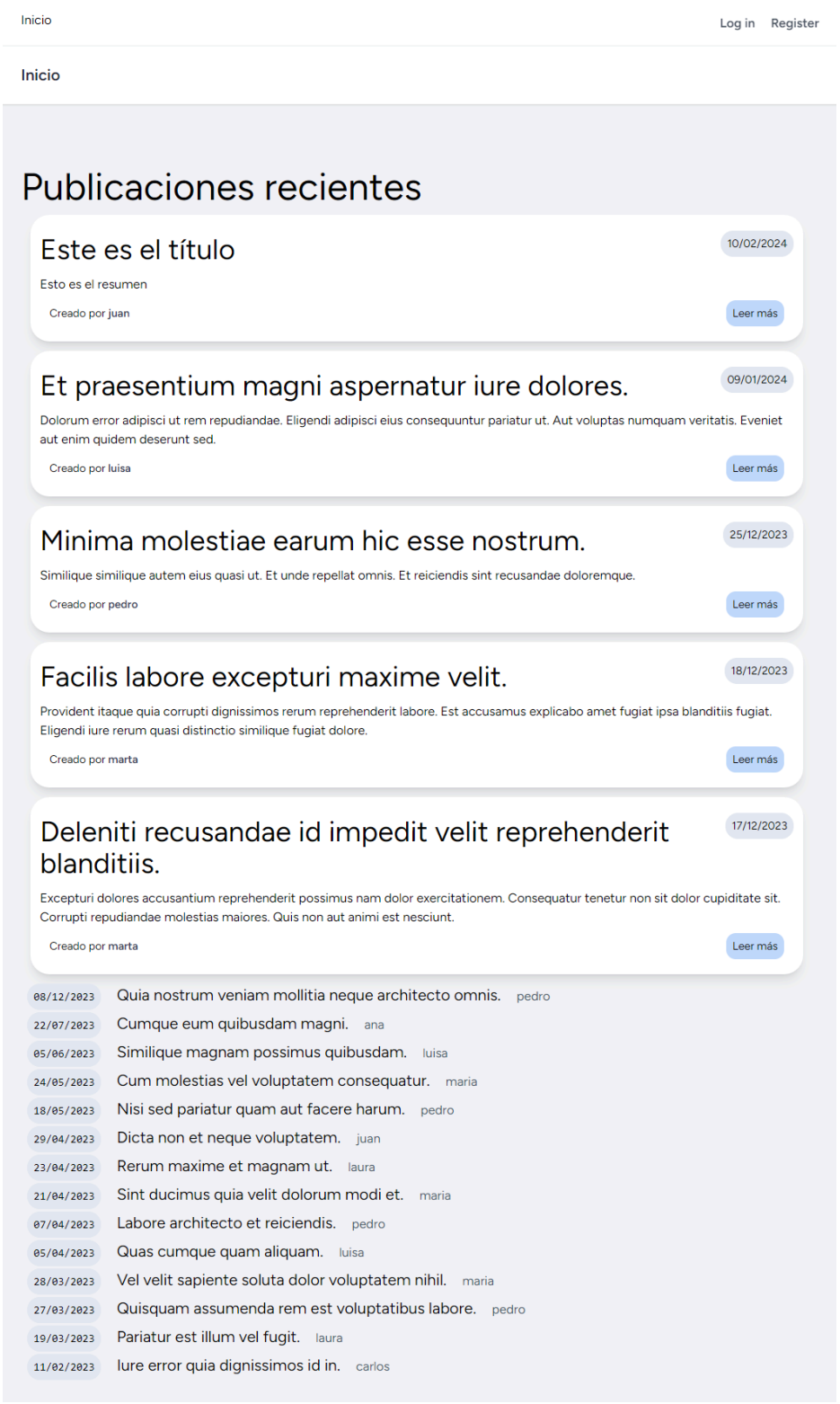
Modificamos la página principal para que se muestre un listado de publicaciones tanto si estás registrado como no. Las publicaciones se van a mostrar ordenadas por orden de publicación de forma descendente y sin visualizar las que tienen una publicación posterior a la de hoy.

Las 5 primeras publicaciones se mostrarán con título y resumen.

Luego las 20 siguientes publicaciones se mostrarán sólo con título.

Y todas con el botón leer más para acceder a la publicación completa.

Más o menos la página de inicio deberá quedar así:



4.1. Menú de navegación

Queremos que toda la aplicación tenga una sola plantilla y con el mismo menú de navegación. Pero por otro lado, algunos enlaces dependerán de si el usuario se ha

autenticado o no. Por tanto, vamos a modificar el menú añadiendo las directivas @auth / @endauth a aquellos enlaces que son solo para usuarios logueados. Y por otro lado, vamos a añadir los enlaces para login y registrarse que encontramos en la plantilla de welcome.

Abrimos el archivo layouts/navigation.blade.php y lo modificamos de la siguiente manera.

```
<nav x-data="{ open: false }" class="bg-white border-b border-gray-100">
    <!-- Primary Navigation Menu -->
    <div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
        <div class="flex justify-between h-16">
            <div class="flex">
                <!-- Logo -->
                <div class="shrink-0 flex items-center">
                    <a href="{{ route('home') }}">Inicio</a>
                </div>
                @auth
                <!-- Navigation Links -->
                <div class="hidden space-x-8 sm:-my-px sm:ms-10 sm:flex">
                     <x-nav-link :href="route('posts.index')"
                        :active="request()->routeIs('posts.index')">
                         {{ __('Tus publicaciones') }}
                    </x-nav-link>
                </div>
                @endauth
            </div>
            @if (Route::has('login'))
            <div class="sm:fixed sm:top-0 sm:right-0 p-6 text-right z-10">
                @auth
                <!-- Settings Dropdown -->
                <div class="hidden sm:flex sm:items-center sm:ms-6">
                    <x-dropdown align="right" width="48">
                        <x-slot name="trigger">
                            <button class="inline-flex items-center px-3 py-2 border
border-transparent text-sm leading-4 font-medium rounded-md text-gray-500 bg-white
hover:text-gray-700 focus:outline-none transition ease-in-out duration-150">
                                <div>{{ Auth::user()->name }}</div>

                                <div class="ms-1">
                                    <svg class="fill-current h-4 w-4"
xmlns="http://www.w3.org/2000/svg" viewBox="0 0 20 20">
                                        <path fill-rule="evenodd" d="M5.293 7.293a1 1 0 011.414
0L10 10.586l3.293-3.293a1 1 0 111.414 1.414l-4 4a1 1 0 01-1.414 0l-4-4a1 1 0 010-1.414z"
clip-rule="evenodd" />
                                    </svg>
                                </div>
                            </button>
                        </x-slot>

                        <x-slot name="content">
                            <x-dropdown-link :href="route('profile.edit')">
                                {{ __('Profile') }}
                            </x-dropdown-link>
```

```
                            <!-- Authentication -->
                            <form method="POST" action="{{ route('logout') }}">
                                @csrf

                                <x-dropdown-link :href="route('logout')"
onclick="event.preventDefault();
                                                this.closest('form').submit();">
                                    {{ __('Log Out') }}
                                </x-dropdown-link>
                            </form>
                        </x-slot>
                    </x-dropdown>
                </div>
                @else
                <a href="{{ route('login') }}" class="font-semibold text-gray-600
hover:text-gray-900 dark:text-gray-400 dark:hover:text-white focus:outline focus:outline-2
focus:rounded-sm focus:outline-red-500">Log in</a>

                @if (Route::has('register'))
                <a href="{{ route('register') }}" class="ml-4 font-semibold text-gray-600
hover:text-gray-900 dark:text-gray-400 dark:hover:text-white focus:outline focus:outline-2
focus:rounded-sm focus:outline-red-500">Register</a>
                @endif
                @endauth
            </div>
            @endif

            <!-- Hamburger -->
            <div class="-me-2 flex items-center sm:hidden">
                <button @click="open = ! open" class="inline-flex items-center justify-center
p-2 rounded-md text-gray-400 hover:text-gray-500 hover:bg-gray-100 focus:outline-none
focus:bg-gray-100 focus:text-gray-500 transition duration-150 ease-in-out">
                    <svg class="h-6 w-6" stroke="currentColor" fill="none" viewBox="0 0 24 24">
                        <path :class="{'hidden': open, 'inline-flex': ! open }"
class="inline-flex" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M4 6h16M4
12h16M4 18h16" />
                        <path :class="{'hidden': ! open, 'inline-flex': open }" class="hidden"
stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M6 18L18 6M6 6l12 12" />
                    </svg>
                </button>
            </div>
        </div>
    </div>

    <!-- Responsive Navigation Menu -->
    <div :class="{'block': open, 'hidden': ! open}" class="hidden sm:hidden">
        <div class="pt-2 pb-3 space-y-1">
            <x-responsive-nav-link :href="route('dashboard')"
:active="request()->routeIs('dashboard')">
                {{ __('Dashboard') }}
```

```
            </x-responsive-nav-link>
        </div>
        @auth
        <!-- Responsive Settings Options -->
        <div class="pt-4 pb-1 border-t border-gray-200">
            <div class="px-4">
                <div class="font-medium text-base text-gray-800">{{ Auth::user()->name }}</div>
                <div class="font-medium text-sm text-gray-500">{{ Auth::user()->email }}</div>
            </div>

            <div class="mt-3 space-y-1">
                <x-responsive-nav-link :href="route('profile.edit')">
                    {{ __('Profile') }}
                </x-responsive-nav-link>

                <!-- Authentication -->
                <form method="POST" action="{{ route('logout') }}">
                    @csrf

                    <x-responsive-nav-link :href="route('logout')"
onclick="event.preventDefault();
                                        this.closest('form').submit();">
                        {{ __('Log Out') }}
                    </x-responsive-nav-link>
                </form>
            </div>
        </div>
        @endauth
    </div>
</nav>
```

Ahora el menú nos servirá tanto para usuarios registrados o no.

4.2. Ruta de navegación

En *routes/web.php* comentamos la ruta '*/*' que va a welcome y añadimos la ruta a *home*.

```
// Route::get('/', function () {
//     return view('welcome');
// });
Route::get('/', [PostController::class, 'home'])->name('home');
```

Vamos a usar el mismo controlador *PostController* pero añadiendo el método *home*.

4.3. Método home

En el controlador *PostController* añadimos el método *home* para mostrar la vista home.

Hay que destacar que para optimizar, no vamos a recuperar todos los datos de todas las publicaciones, vamos a hacer la selección de columnas (***select***) y un filtrado (***where***). Además ordenamos por fecha de publicación (***orderByDesc***) y obtenemos sólo los elementos que queremos (***take***) y saltandonos los primeros (***skip***).

```
  public function home()
  {
    $firstPosts = Post::select('id', 'title', 'summary', 'published_at', 'user_id')
      ->where('published_at', '<=', \Carbon\Carbon::today())
      ->orderByDesc('published_at')
      ->take(5)
      ->get();

    $otherPosts = Post::select('id', 'title', 'published_at', 'user_id')
      ->where('published_at', '<=', \Carbon\Carbon::today())
      ->orderByDesc('published_at')
      ->skip(5)
      ->take(20)
      ->get();

      return view('welcome', compact('firstPosts', 'otherPosts'));
  }
```

En *firstPosts* obtenemos los 5 primeros resultados en los que se verá el título, usuario, fecha y resumen. Y en *otherPosts*, los 20 siguientes en los que no necesitamos el resumen.

4.4. Vista home

Ahora creamos el archivo home.blade.php como una copia de dashboard.blade.php y modificamos para que se muestren las publicaciones.

```
<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 leading-tight">
            {{ __('Inicio') }}
        </h2>
    </x-slot>

    <div class="py-12">
    <div class="max-w-7xl mx-auto p-6 lg:p-8">
      <h1 class="text-5xl">Publicaciones recientes</h1>
      <div class="">
        @foreach ($firstPosts as $post)
        <div class="rounded-3xl shadow-lg bg-white m-3 p-3 border-gray-500">
          <div class="grid grid-flow-col justify-items-stretch">
            <span class="text-4xl mt-3 col-auto">{{$post->title}}</span>
            <div class="text-right pt-3">
              <span class="text-sm bg-slate-200 p-2 rounded-2xl mt-5">
```

```
                {{ \Carbon\Carbon::create($post->published_at)->format('d/m/Y')}}
              </span>
            </div>
          </div>
          <p class="mt-3">{{$post->summary}}</p>
          <div class="grid grid-cols-2">
            <div class="m-3">
              <span class="text-sm">
                Creado por
                <span class=" font-bold text-slate-700">
                  {{ $post->user->name }}
                </span>
              </span>
            </div>
            <div class="m-3 text-right">
              <a class="text-sm bg-blue-200 p-2 rounded-xl"
                 href="{{ Route('posts.read', $post->id)}}">
                Leer más
              </a>
              </span>
            </div>
          </div>
        </div>
        @endforeach
      </div>
      <ul class="list-outside">
        @foreach ($otherPosts as $post)
        <li class="m-2">
          <span class="text-sm font-mono bg-slate-200 p-2 rounded-2xl">
            {{ \Carbon\Carbon::create($post->published_at)->format('d/m/Y')}}</span>
          <a href="{{ Route('posts.read', $post->id)}}"
            class="text-xl mt-3 col-auto mx-4 hover:text-blue-500">{{$post->title}}</a>
          <span class="mt-4 text-slate-600 col-auto">{{$post->user->name}}</span>
        </li>
        @endforeach
      </ul>
    </div>
    </div>
</x-app-layout>
```

Cabe destacar que tenemos dos @foreach, uno para las 5 primeras publicaciones y otro para las 20 siguientes.

También se añade un enlace para leer más, en las 5 primeras en un botón y en las siguientes con un enlace en el título. Dicho enlace lleva a leer la publicación que se muestra a continuación.

Añade la ruta en routes/web.php

```
Route::get('/posts/{id}/read', [PostController::class, 'read'])->name('posts.read');
```

5. Leer publicación

Ahora creamos una copia de s*how.blade.php* y la llamamos *read.blade.php*. Cambiamos el estilo a nuestro gusto, añadimos el usuario y eliminamos los botones de editar y borrar.

También modificamos el volver para que vaya a *home*.

```
<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 leading-tight">
            {{ __('Leyendo Publicación') }}
        </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">
                    <p class="text-sm text-right">
                        {{ \Carbon\Carbon::create($post->published_at)->format('d/m/Y') }}
                    </p>
                    <h2 class="font-semibold text-4xl text-black-900 leading-tight m-3">
                        {{ $post->title}}
                    </h2>
                    <p class="italic m-3 text-xl text-gray-800 font-semibold">
                        {{ $post->summary }}
                    </p>
                    <p class="m-3 text-lg">
                        {{ $post->body }}
                    </p>
                    <div class="text-right text-xl text-slate-500">
                        - {{ $post->user->name }} -
                    </div>
                    <div class="text-center">
                        <a href="{{ route('home') }}"
                        class="p-3 bg-blue-200 hover:bg-blue-500">Volver</a>
                    </div>
                </div>
            </div>
        </div>
    </div>
</x-app-layout>
```

Podíamos haber puesto show y read en una misma página, con algunos cambios, pero es mejor aparte porque seguramente posteriormente personalicemos más la página de lectura de un visitante que la de visualizar tu propia publicación.

El método read en PostController queda de la siguiente manera:

```
  public function read(int $id)
  {
```

```
    $post = Post::find($id);
    return view('posts.read', compact('post'));
}
```

6. Valorar publicación

La valoración de la publicación es muy sencilla. Cuando un usuario registrado está leyendo un artículo, le aparece un botón de “me gusta”. Si pulsa el botón se añade un punto a la publicación. Si el usuario vuelve a visitar la página, no puede volver a pulsar el botón “me gusta”.

Por tanto tenemos una relación de usuario vota publicación que es muchos a muchos ya que un usuario puede votar muchas publicaciones y una publicación puede ser votada por muchos usuarios.

6.1. Crear la tabla relación

Laravel tiene una forma rápida de trabajar con estas tablas muchos a muchos y sugiere llamarla *‘post_user’*, pero no me parece buena práctica porque no se sabe qué es. Por lo que, aunque escribamos algo más de código, vamos a llamarla *‘vote’* en el modelo y *‘votes’* en la tabla de la base de datos.

Creamos la migración para esta tabla.

```
➢ php artisan make:migration create_votes_table
```

Ahora vamos a crear los dos campos necesarios que son los identificadores de usuario y publicación. Si existe un registro, es que ese usuario dio un voto a dicha publicación.

```
Schema::create('votes', function (Blueprint $table) {
    $table->id();
    $table->unsignedBigInteger('user_id');
    $table->unsignedBigInteger('post_id');
    $table->timestamps();
    $table->foreign('user_id')->references('id')->on('users');
    $table->foreign('post_id')->references('id')->on('posts');
});
```

No te olvides de hacer la migración

```
➢ php artisan migrate
```

6.2. Funciones en los modelos

Ahora añadimos la funcionalidad de la relación en los modelos.

En el modelo Post ponemos la función que nos devuelve todas los usuarios que la han votado.

```
public function votedUsers() : BelongsToMany
{
    return $this->belongsToMany(User::class, 'votes');
}
```

Hay que añadirle como segundo parámetro el nombre de la tabla porque no estamos usando ‘post_user’.

Añade algunos registros que muestren la relación entre usuarios y publicaciones. Puedes crear un seeder si quieres.

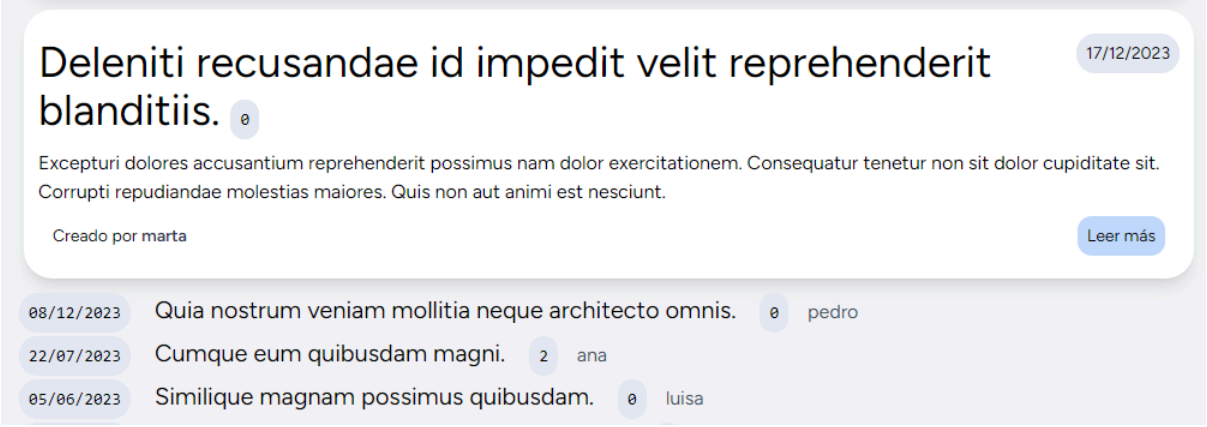
6.3. Visualizar las votaciones

Añadimos en la página home los votos para cada publicación.

Junto a cada título, tanto en el primer foreach como en el segundo, añadimos el siguiente código:

```
<span class="text-4xl mt-3 col-auto">{{$post->title}}
  @if ($post->votedUsers)
  <span class="text-sm font-mono bg-slate-200 p-2 rounded-xl">
    {{ $post->votedUsers->count() }}</span>
  @endif
</span>
```

De esta forma se ven los votos de cada artículo.



6.4. Realizar votación

En el archivo posts/read.blade.php añadimos el botón para votar si es un usuario registrado.

Lo añadimos junto al título de la siguiente forma:

```
<h2 class="font-semibold text-4xl text-black-900 leading-tight m-3">
```

```
        {{ $post->title }}
      @if ($post->votedUsers)
          <span class="text-sm font-mono bg-slate-200 p-2 rounded-xl">
            {{ $post->votedUsers->count() }}</span>
      @endif
      @auth
        <form action="{{ route('posts.vote', $post) }}" method="post"
            class="text-sm inline-block">
            @csrf
            <input type="submit" value="me gusta"
                class="bg-cyan-300 hover:bg-green-400 p-2
                      rounded-xl cursor-pointer">
        </form>
      @endauth
    </h2>
```

Puedes ver que además de añadir el número de votos hay un formulario que va a lanzar la ruta posts/?/vote donde la ? es el id del posts. El usuario no hace falta pasarlo porque está en la sesión.

Creamos la ruta:

```
Route::post('/posts/{post}/vote', [PostController::class, 'vote'])->name('posts.vote')
    ->middleware('auth');
```

No hay que olvidar el middleware para que no permita acceder a usuarios no autenticados.

Ahora el controlador debe quedar de la forma siguiente:

```
  public function vote(Post $post) {
    // Comprobamos que no haya votado ya.
    $vote = $post->votedUsers()->find(Auth::id());
    // Si no ha votado, lo añadimos.
    if (!$vote) {
      $post->votedUsers()->attach(Auth::id());
    }
    return redirect()->back();
  }
```

Es interesante ver que primero comprobamos si ya había votado. Si no hay voto, podemos usar el modelo Vote (que tendríamos que crearlo porque no lo hicimos) y decirle que creara uno nuevo con post_id y user_id según lo que tenemos en $post->id y Auth:id(). Pero podemos usar la función attach que nos simplifica crear la relación.

Por otro lado, Laravel también tiene una función ***sync*** que nos permite ahorrarnos comprobar si hay había votado.

Por último, hay una maravilla más de Laravel `redirect()->back()` nos permite volver a la página que teníamos. Y se verá ya actualizado.

Ya estaría listo y solo se puede votar una vez.

Pero no indica que ya se ha votado ni se puede eliminar el voto.

6.5. Anular votación

Para anular la votación vamos a modificar el controlador para dar la opción de eliminar la relación si ya hay un voto dado.

```
  public function vote(Post $post) {
    // Comprobamos que no haya votado ya.
    $vote = $post->votedUsers()->find(Auth::id());
    if (!$vote) {
      // Si no ha votado, lo añadimos.
      $post->votedUsers()->attach(Auth::id());
    } else {
      // Si ha votado, lo eliminamos.
      $post->votedUsers()->detach(Auth::id());
    }
    return redirect()->back();
  }
```

Para ello usamos ***detach*** que es lo contrario a *attach*.

Y modificamos la vista para que el botón se vea diferente según haya votado o no.

```
<h2 class="font-semibold text-4xl text-black-900 leading-tight m-3">
    {{ $post->title }}
    @if ($post->votedUsers)
        <span class="text-sm font-mono bg-slate-200 p-2 rounded-xl">
        {{ $post->votedUsers->count() }}</span>
    @endif
    @auth
    <form action="{{ route('posts.vote', $post) }}" method="post"
    class="text-sm inline-block">
    @csrf
    @if ($post->votedUsers->contains(auth()->user()))
```

```
        <input type="submit" value="ya votaste"
            class="bg-green-300 hover:bg-red-400 p-2 rounded-xl cursor-pointer">
    @else
        @csrf
        <input type="submit" value="me gusta"
            class="bg-cyan-300 hover:bg-green-400 p-2 rounded-xl cursor-pointer">
    @endif
    </form>
    @endauth
</h2>
```