Implementación del Proyecto

Una vez creado nuestro modelo de datos, y creados los datos necesarios para poder probar el desarrollo de nuestra aplicación con información suficiente, lo siguiente será generar rutas, controlador y vistas para darle forma.

Rutas de mi aplicación: Controlador de páginas

Vamos a crear un controlador que gestionará las páginas de mi aplicación. Lo llamaré PageController.

```
$ php artisan make:controller PageController
```

Este controlador retornará las vistas según las rutas indicadas, por tanto tendremos que importarlo en web.php

Las rutas que definiremos en web.php serán:

Web.php

```
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\PageController;


// Ruta raiz, alias 'home'
Route::get('/', [PageController::class, 'home'])->name('home');

// Ruta category/{category}, alias 'page.category'
Route::get('/categoria/{category}', [PageController::class, 'category']
)->name('page.category');

// Ruta etiqueta/{tag}, alias 'page.tag'
Route::get('/etiqueta/{tag}', [PageController::class, 'tag']
)->name('page.tag');

// Ruta hilo/{thread}, alias 'page.thread'
Route::get('/pregunta/{thread}', [PageController::class, 'thread']
)->name('page.thread');
```

Nuestro controlador deberá implementar las funciones home, category, tag y thread, que son las indicadas en las rutas.

PageController.php

```
class PageController extends Controller
```

```
{
    public function home() {
        return 'home';
    }

    public function category($category) {
        return 'category ' . $category;
    }

    public function tag($tag) {
        return 'tag ' . $tag;
    }

    public function thread($thread) {
        return 'thread ' . $thread;
    }
}
```

De momento hemos creado unas funciones básicas que sólo muestran texto, para poder probar todas las rutas de nuestra aplicación. Recuerda, si pruebas desde un navegador, tendrás que escribir la ruta completa, no podrás utilizar el alias.

Estructura de la web

Una vez probadas las rutas, vamos a desarrollar la aplicación.

Modifico el controlador para que enlace con las vistas

Para empezar, queremos que los métodos del controlador generen una vista, no un texto.

PageController.php

```
class PageController extends Controller
{
    public function home() {
        return view('home');
    }

    public function category($category) {
        return view('category', compact('category'));
    }

    public function tag($tag) {
        return view('tag', compact('tag'));
    }

    public function thread($thread) {
        return view('thread', compact('thread'));
```

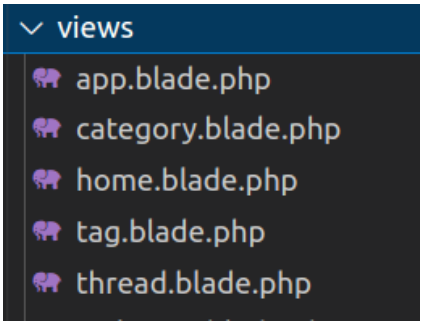
```
    }
}
```

Crearemos una vista para cada ruta, y recibirá como parámetro la variable de la ruta, mediante la función compact.

Creamos las vistas

Además de las cuatro vistas, una por ruta, crearemos una vista plantilla. La vista que hará de plantilla será app.blade.php.

Para ello, creo archivos nuevos en el directorio views, con la nomenclatura de Laravel:



Código de la plantilla app.blade.php

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>@yield('title')</title>
</head>
<body>
    <div>
        <h2>@yield('title')</h2>
        <p>@yield('description')</p>
    </div>

    <div>
        @yield('content')
    </div>
</body>
</html>
```

La información del título la generaremos de forma dinámica, por eso el @yield en el head.

La plantilla constará de dos apartados, el primero con un título y una descripción, el segundo con el contenido de la página.

Vista home: home.blade.php

Esta vista importará la plantilla, y luego sustituirá los apartados por diversos valores:

home.blade.php

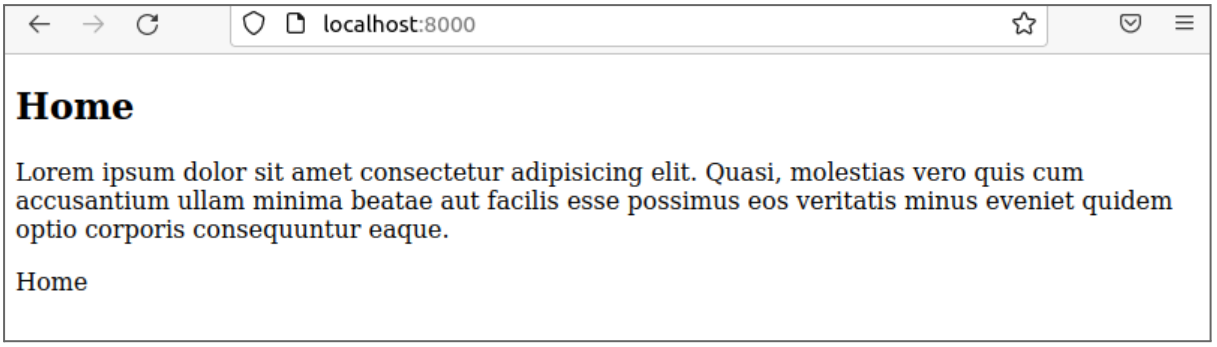
```
@extends('app')

@section ('title', 'Home')

@section ('description', 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Quasi, molestias vero quis cum accusantium ullam minima beatae aut facilis esse possimus eos veritatis minus eveniet quidem optio corporis consequuntur eaque.')

@section('content')

@endsection
```

Como puede verse, hay dos formas de referenciar una sección:

1. @section(nombreSección, textoContenidoEnLaSección). Es el caso de title y description, como son sólo un texto, no hace falta abrir y cerrar sección, podemos poner el contenido como parámetro.
2. @section(nombreSeccion) …… @endsection. En este caso, en el interior de la sección va a haber mucho código, por lo que no es visual ponerlo como parámetro

Nota: Hemos utilizado la utilidad lorem de Visual Studio Code para generer el texto de la descripción. Puede hacerse escribiendo lorem y luego pulsando enter.

Si ponemos la ruta raíz en nuestro navegador, ahora se mostrará lo siguiente:

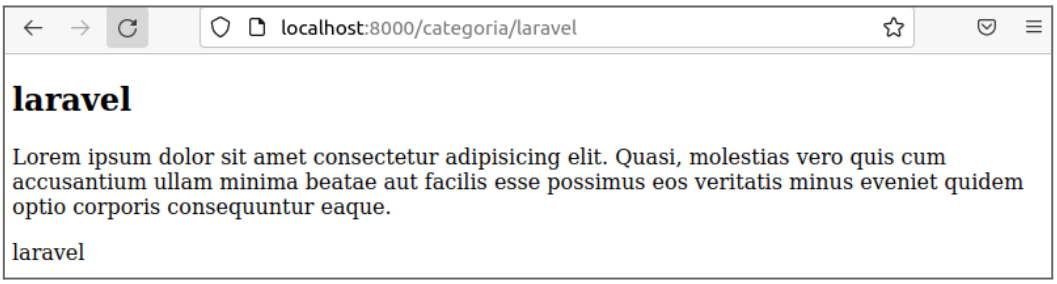


Vista category.blade.php

```
@extends('app')
```

```
@section ('title', $category)

@section ('description', 'Lorem ipsum dolor sit amet consectetur
adipisicing elit. Quasi, molestias vero quis cum accusantium ullam minima
beatae aut facilis esse possimus eos veritatis minus eveniet quidem optio
corporis consequuntur eaque.')

@section('content')
    {{$category}}
@endsection
```

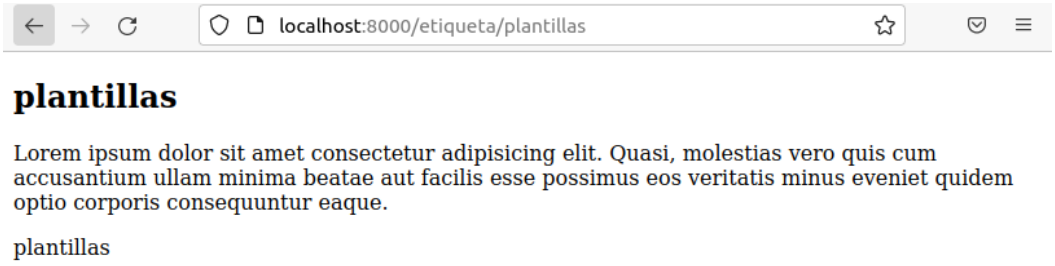
En el navegador tendrá el aspecto:



Vista tag.blade.php

```
@extends('app')

@section ('title', $tag)

@section ('description', 'Lorem ipsum dolor sit amet consectetur
adipisicing elit. Quasi, molestias vero quis cum accusantium ullam minima
beatae aut facilis esse possimus eos veritatis minus eveniet quidem optio
corporis consequuntur eaque.')

@section('content')
    {{$tag}}
@endsection
```

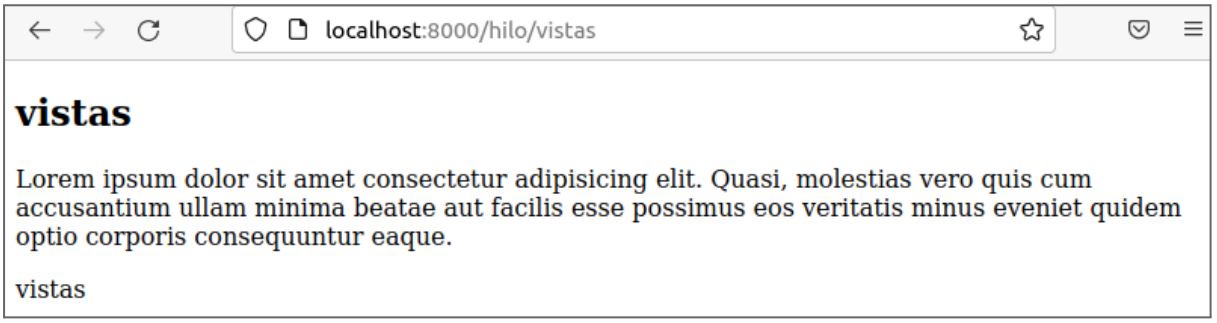
En el navegador se verá como:



Vista thread.blade.php

```
@extends('app')

@section ('title', $thread)

@section ('description', 'Lorem ipsum dolor sit amet consectetur
adipisicing elit. Quasi, molestias vero quis cum accusantium ullam minima
beatae aut facilis esse possimus eos veritatis minus eveniet quidem optio
corporis consequuntur eaque.')

@section('content')
    {{$thread}}
@endsection
```

En el navegador se verá como:



Diseño de plantilla

La ventaja de trabajar con una plantilla, es que podemos darle estilo, que será heredado por el resto de vistas de la aplicación.

Para dar estilo, podemos utilizar:

- Iconos obtenidos de alguna página web, por ejemplo heroicons.dev
- CSS
- CSS con un preprocesador. Recuerda procesarlo antes de ponerlo en el public
- Un framework para diseño responsivo, como por ejemplo bootstrap

Todo estilo que le des a la plantilla, será heredado por todas las páginas de tu aplicación.

PÁGINA HOME

En este apartado, vamos a recuperar información de la base de datos para mostrar en nuestra página home. La idea es mostrar todos los hilos.

Como sabemos, en Laravel es el controlador el intermediario entre modelo y vista. Por tanto, necesitaremos:

1. Modificar el método home de PostController.php para recuperar la información.
2. Pasar de parámetro esa información a la vista
3. Recuperar los datos en la vista con el formato que me interese.

Controlador

PostController.php

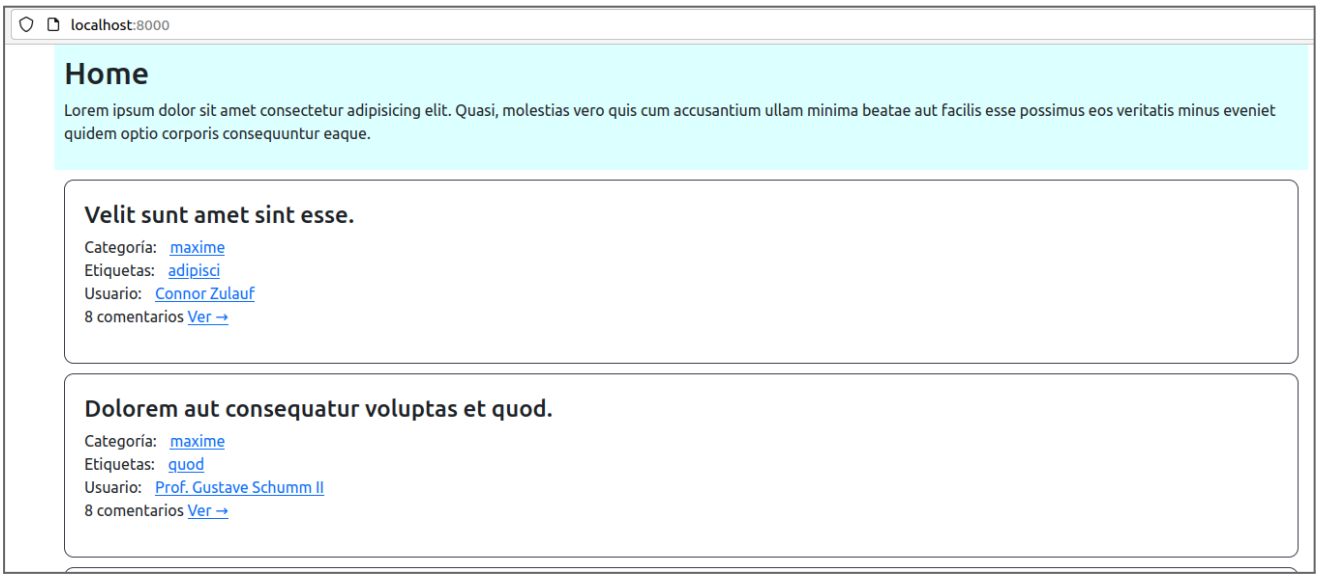
```
use App\Models\Thread;
...
    public function home() {
        $threads = Thread::orderBy('id','DESC')->paginate();
        return view('home', compact('threads'));
    }
```

Obtenemos todas las preguntas de la tabla threads, ordenada por id, de forma descendente (las últimas se visualizarán primero), y luego utilizaremos paginate para paginarlas.

Vista y modificación del modelo

En la vista, utilizaré los datos pasados por el controlador de parámetro en la variable threads.

El objetivo es que muestre la siguiente información:



Nota: El estilo es orientativo, lo ideal sería utilizar CSS para dar un aspecto profesional, utilizando diseño responsivo, iconos, etc.

Lo que vamos a mostrar es los siguiente:

1. Todas las preguntas, con una barra de navegación en la parte inferior para que no haya demasiadas en la misma página. Las más recientes aparecerán primero
2. De cada pregunta, mostraremos:

a. Su categoría: El nombre de la categoría
b. Sus etiquetas: El nombre de la etiqueta
c. El usuario que la creó: El nombre del usuario que la creó
d. Los comentarios: Indicando el número, y un enlace a los mismos

Obtención de datos para la vista

Como se ve en el ejemplo, vamos a mostrar, de cada pregunta, el título, la categoría, las etiquetas y el usuario.

Necesitaremos:

1. Implementar en los modelos los métodos que obtienen los datos de las relaciones
2. Hacer que el controlador pase los datos a la vista
3. Implementar la visualización de los datos en la vista

En el modelo de la tabla threads, ya habíamos incluido funciones para obtener los comentarios y las etiquetas. Sin embargo, no habíamos incluido funciones para obtener el usuario o la categoría, esa relación había quedado sin implementar.

Por tanto, tendré que modificar el modelo Thread.php para permitir mostrar esa información:

Tread.php

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

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

La función belongsTo obtiene el padre de una relación 1 a n. En este caso, obtendrá la categoría de la pregunta, y el usuario de la pregunta.

Una vez implementadas estas relaciónes, la vista podrá recuperar la información.

home.blade.php

```
// Importo la plantilla básica
@extends('app')

// Incluyo un título, la sección title contendrá “Home”,
@section ('title', 'Home')

// Añadimos la descripción, la sección description contendrá un texto largo
@section ('description', 'Lorem ipsum dolor sit amet consectetur adipisicing
elit. Quasi, molestias vero quis cum accusantium ullam minima beatae aut
facilis esse possimus eos veritatis minus eveniet quidem optio corporis
consequuntur eaque.')
```

```
// La sección content tiene muchas cosas, se obtendrá mediante código blade
// El código blade de content se muestra en morado. El resto es HTML
// Muestra todas las preguntas, mediante un foreach
@section('content')
    @foreach($threads as $thread)
        <div class="borde">
            <h2 class='tgrande'>{{ $thread->titulo }}</h2>
            <div>
                Categoría:
                <a href="#" class="separado">
                    {{ $thread->category->nombre }}
                </a>
                <br>
                Etiquetas:
                <span>
                    @foreach($thread->tags as $tag)
                        <a class="separado" href="#">
                            {{ $tag->nombre}}
                        </a>
                    @endforeach
                </span>
                <br>
                Usuario:
                <span>
                    <a class="separado" href="#">
                        {{ $thread->user->name }}
                    </a>
                </span>
                <p>
                    {{$thread->comments->count()}} comentarios
                    <span><a href="#">Ver &rarr;</a></span>
                </p>
            </div>
        </div>
    @endforeach

    {{ $threads->links('pagination::bootstrap-5') }}
@endsection
```

Si observamos, las dos primeras secciones utilizan una sintaxis

```
@section(nombreSeccion, contenidoSeccion)
```

En cambio la sección del contenido, como requiere mucho código, utiliza la sintaxis:

```
@section(nombreSeccion)
      // Aquí el código blade para mostrar el contenido
 @endsection
```

Utilizamos las clases de nuestro modelo para obtener la información.

El @foreach se utiliza para recorrer los arrays. El código php se incrusta mediante llaves.

Al final de la sección contenido, hemos añadido la paginación:
`{{ $threads->links('pagination::bootstrap-5') }}`
Se recorre el array de preguntas, mostrando varios por página, conforme al css de bootstrap 5. Para ello, es necesario que en nuestra plantilla incluyamos un link a bootstrap 5:

app.blade.php

```
<head>
...
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous"
>
...
```

Nueva plantilla _item.blade.php

La plantilla home.blade.php queda muy extensa, y por tanto, complicada de leer.

Vamos a crear una plantilla adicional, para organizar mejor el código de la plantilla home. Esta nueva plantilla incorporará todo el código correspondiente a un único elemento (item). La llamaré _item.blade.php

Juntaré en esta nueva plantilla todo el código correspondiente a una única pregunta.

Además, la plantilla _item.blade.php podrá utilizarse en otras vistas cuando quiera mostrar una pregunta con toda su información, siempre que la pregunta se haya obtenido en una variable $thread.

El código quedaría como sigue:

home.blade.php

```
@extends('app')

@section ('title', 'Home')

@section ('description', 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Quasi, molestias vero quis cum accusantium ullam minima beatae aut facilis esse possimus eos veritatis minus eveniet quidem optio corporis consequuntur eaque.')

@section('content')
    @foreach($threads as $thread)
```

```
        <div class="borde">
            @include('_item')
        </div>
    @endforeach

    {{ $threads->links('pagination::bootstrap-5') }}
@endsection
```

He cortado el código correspondiente a la visualización de cada una de las preguntas, y me lo he llevado al archivo _item.blade.php. En su lugar, pongo el **include**, que incluirá todo lo que hay en la plantilla _item en ese sitio.

La plantilla _item quedará:

_item.blade.php

```
<h2 class='tgrande'>{{ $thread->titulo }}</h2>
<div>
    Categoría:
    <a href="#" class="separado">
        {{ $thread->category->nombre }}
    </a>
    <br>
    Etiquetas:
    <span>
        @foreach($thread->tags as $tag)
            <a class="separado" href="#">
                {{ $tag->nombre}}
            </a>
        @endforeach
    </span>
    <br>
    Usuario:
    <span>
        <a class="separado" href="#">
            {{ $thread->user->name }}
        </a>
    </span>
    <p>
        {{$thread->comments->count()}} comentarios
        <span><a href="#">Ver &rarr;</a></span>
    </p>
</div>
```

Si ejecutamos esto en el navegador, el resultado será exactamente el mismo.

VISTA DE CATEGORÍAS

La vista de categorías mostrará las preguntas de una categoría concreta. Para conseguirlo, tendré que realizar las siguientes configuraciones:

1. Configurar el enlace a la categoría, en _item.blade.php, utilizando la ruta de categorías.
2. Modificar la ruta de categorías para indicar que la variable corresponde al slug de la categoría
3. Modificar el controlador para recuperar todas las preguntas de una categoría, y pasarlas como parámetro a la vista
4. Crear la vista de categorías, que será similar a la de home:
    a. Mostrará las preguntas recuperadas por el controlador, con el mismo formato
    b. Utilizará la plantilla
    c. Utilizará la vista auxiliar _item.blade.php

Configuración del enlace a la vista

Vamos a configurar el enlace a la página de categorías. Para ello, editamos el fichero _item.blade.php, y mediante la función route recuperamos la URL correspondiente a la ruta de la página de categorías.

_item.blade.php

```
Categoría:
    <a href="{{ route('page.category', $thread->category->slug)}}"
     class="separado">
        {{ $thread->category->nombre }}
    </a>
    <br>
```

Utilizo la función **route** para recuperar una URL correspondiente a una ruta de mi aplicación. Los parámetros de la función route son:

1. Nombre de la ruta. Aquí podremos utilizar el alias de la ruta, en nuestro caso page.category, ya definido en web.php
2. Si la ruta tiene variables, se pasan en forma de array en el segundo parámetro. Como en esta ruta sólo tenemos una variable, podemos ponerla directamente.
3. He puesto como parámetro el campo slug de la categoría, así que tendré que indicar en la ruta que ese es el campo por el que identificar la categoría

Configuración de la ruta para que utilice el slug

Para que la ruta tenga definida como variable el campo slug, tendré que modificarla de la forma:

1. Modifico el web.php para indicar que la variable es el slug de la categoría
2. Modifico el controlador

definirla de esta forma:

web.php

```
// Ruta category/{category}, alias 'page.category'
```

```
Route::get('/categoria/{category:slug}',
        [PageController::class, 'category'])->name('page.category');
```

De esta forma, queda definido el enlace.

Recuperación de las preguntas por el controlador

El controlador tendrá que recuperar las preguntas. Para ello:

PostController.php

```
use App\Models\Category;
...
public function category(Category $category) {
    $threads = $category->threads()->orderBy('id','DESC')->paginate();
    return view('category', compact('category', 'threads'));
}
```

Hemos modificado el parámetro, para que reciba un objeto con los datos de una categoría concreta, que se buscó mediante el slug (ver ruta).
En cuanto a la recuperación de las preguntas de una categoría, puede hacerse porque el modelo Category.php dispone de la función threads, que recupera las preguntas implementando la relación 1 a n.

Las preguntas, al igual que en home, se mostrarán paginadas, primero las más recientes.

Modificación de la vista de categoría

La vista es similar a la de home, esta vez recibe en threads las preguntas de una categoría, y también recibe el slug de la categoría.

category.blade.php

```
@extends('app')

@section ('title', $category->name)

@section ('description', 'Lorem ipsum dolor sit amet consectetur
adipisicing elit. Quasi, molestias vero quis cum accusantium ullam minima
beatae aut facilis esse possimus eos veritatis minus eveniet quidem optio
corporis consequuntur eaque.')

@section('content')
    @foreach($threads as $thread)
        <div class="borde">
            @include('_item')
        </div>
    @endforeach
```

```
    {{ $threads->links('pagination::bootstrap-5') }}
@endsection
```

VISTA DE ETIQUETAS

La vista de etiquetas es similar a la de home y a la de categorías. Lo que hacemos es mostrar las preguntas correspondientes a una etiqueta concreta.

Enlace a la página de etiquetas

_item.blade.php

```
        @foreach($thread->tags as $tag)
            <a href="{{route('page.tag', $tag->slug)}}" class="separado">
                {{ $tag->nombre}}
            </a>
        @endforeach
```

Adaptación de la ruta

web.php

```
// Ruta tag/{tag}, alias 'page.tag'
Route::get( '/etiqueta/{tag:slug}',
            [PageController::class, 'tag']
)->name('page.tag');
```

Adaptación del controlador

PostController.php

```
use App\Models\Tag;
...
    public function tag(Tag $tag) {
        $threads = $tag->threads()->orderBy('id','DESC')->paginate();
        return view('tag', compact('tag','threads'));
    }
```

En este controlador estamos recuperando las preguntas de una etiqueta. Para eso, necesitamos que nuestro modelo Tag.php disponga de la función threads, que implemente la relación, y no la tenemos.

Actualizamos el modelo de la forma:

Tag.php

```
    public function threads() {
        return $this->belongsToMany(Thread::class);
    }
```

En este caso, la relación era n a n, por lo tanto recupero las preguntas que pertenecen a esta etiqueta, sabiendo que pueden pertenecer a más etiquetas, mediante la función belongsToMany.

Configuramos la vista de etiquetas

tag.blade.php

```
@section('content')
    @foreach($threads as $thread)
        <div class="borde">
            @include('_item')
        </div>
    @endforeach
    {{ $threads->links('pagination::bootstrap-5') }}
@endsection
```

VISTA DE PREGUNTA

En esta vista se mostrará una pregunta concreta, con toda su información. Se verán completos todos los comentarios.

Enlace a la página de Preguntas

_item.blade.php

```
        {{$thread->comments->count()}} comentarios
        <span><a href="{{route('page.thread', $thread->slug)}}"></span>
```

Adaptación de la ruta

Web.php

```
// Ruta hilo/{thread}, alias 'page.thread'
Route::get('/hilo/{thread:slug}',[PageController::class, 'thread']
)->name('page.thread');
```

Adaptación del controlador

PageController.php

```
    public function thread(Thread $thread) {
        return view('thread', compact('thread'));
    }
```

Desarrollo de la vista de pregunta

Esta vista mostrará toda la información de una única pregunta.

Thread.blade.php

```
@extends('app')

@section ('title', $thread->titulo)

@section ('description', $thread->category->nombre)

@section('content')
    <div class="borde">
         {{ $thread->body }}
    </div>
    <div>
        Etiquetas:
        <span>
            @foreach($thread->tags as $tag)
                <a href="{{route('page.tag', $tag->slug)}}" class="separado">
                    {{ $tag->nombre}}
                </a>
            @endforeach
        </span>
    </div>
    <br>
    <div>
        <h2>
            {{$thread->comments->count()}} comentarios
            <span>
              <a href="{{route('page.thread', $thread->slug)}}">
                 Ver &rarr;
              </a>
            </span>
        </h2>
        @foreach($thread->comments as $comment)
            <!-- AQUI SE OBTIENE LA INFORMACIÓN DE UN COMENTARIO -->
        @endforeach
    </div>
@endsection
```

Querré mostrar un título, que coincidirá con el título de la pregunta.
Después, mostraré como descripción el nombre de la categoría de la pregunta.

Para mostrar el resto:

- Incluyo la información en la sección content
- Añado un div para mostrar el cuerpo de la pregunta
- Añado otro div para mostrar las etiquetas
- Añado otro div para mostrar cuántos comentarios hay

- Después, mostraremos toda la información de los comentarios, tanto su autor como el contenido, tal y como se explica en el siguiente apartado.

Obtención de los datos de los comentarios

Primero vamos a mostrar el usuario del comentario. Esta relación no está implementada en el modelo, por lo que tendremos que editar el modelo de la tabla de comentarios para añadir un método que recupere el usuario.

Comment.php

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

De esta forma, ya podremos referenciar el usuario desde un comentario con la expresión $comment->user, y una vez obtenido el usuario, podremos mostrar cualquiera de los campos del registro, como por ejemplo el nombre:

$comment->user->name

La parte de la vista que recupera los comentarios quedaría como sigue:

Thread.blade.php

```
    <div>
        <h2>
            {{$thread->comments->count()}} comentarios
        </h2>
        @foreach($thread->comments as $comment)
            <div class="borde">
                <div>
                    {{ $comment->user->name}}
                </div>
                <div>
                    {{ $comment->body }}
                </div>
            </div>
        @endforeach
    </div>
```

Así, si navegamos a la página de una pregunta concreta, obtendremos lo siguiente:


localhost:8000/hilo/deserunt-impedit-ipsam-nulla-dolor-harum
Deserunt impedit ipsam nulla dolor harum.
Praesentium vel eaque veniam cum accusamus eum recusandae. Quam facere laudantium quaerat ut officia. Quod deserunt enim possimus voluptatibus placeat et facere. Pariatur est non vero totam cumque tempore sint asperiores. Dolor aut corporis expedita in minima cum ratione nemo. Labore et pariatur quo minus et excepturi neque. Dolor dolorem quibusdam voluptatem expedita doloribus incidunt. Molestiae laborum quaerat vel. Culpa omnis perspiciatis sit sequi ea consequatur sit. Quaerat omnis expedita quia. Voluptas facilis ab nisi voluptatibus laudantium quis provident. Facilis ea repudiandae expedita consequuntur. Veritatis excepturi sed quis nisi esse. Recusandae omnis a pariatur ut et distinctio quas. Ad similique beatae voluptate. Cum quis culpa fugiat nostrum inventore nihil. Minus magni suscipit quaerat accusantium debitis minima ut dolorem. Facilis rerum modi et in velit rerum. Nemo et pariatur in praesentium nihil. Rerum voluptate consequatur laudantium quis similique autem fuga expedita. Dolor sunt assumenda tempore enim. Aut enim ut blanditiis sequi omnis corporis.
Etiquetas: aperiam adipisci quisquam ut iste quod
8 comentarios
Mr. Jess Ruecker
Deserunt nam voluptas nostrum est velit sit. Voluptatem nisi omnis tempora ratione. Aut eaque est quia aut sequi vero aliquam. Ea odio ea quam tempora. Reprehenderit explicabo amet cum voluptates sit occaecati similique. Perferendis dolores vel iure aliquam ducimus et quas. Neque totam quisquam assumenda doloribus natus sit nostrum sint. Non doloremque et et et explicabo. Corrupti aspernatur sunt consequatur non debitis. Sunt dolore inventore et est. Adipisci veniam nemo iure sunt.
Dortha Grady
Eos pariatur sit eos. Expedita repellat deleniti natus quas. Dolorum adipisci et laboriosam commodi cum ut voluptatem. Aspernatur illo consectetur commodi. Aut qui consequatur delectus quia. Voluptatem sed et consequatur fugit numquam nobis odit id. Voluptates quibusdam aut earum dolores quo molestias. Ea qui quia molestias.