

[<--- Volver](#)

Working with databases

Archivos de entorno y conexiones de bases de datos / Environment Files and Database Connections

Los archivos de entorno nos ayudan a tener privatizados algunos datos sobre nuestro proyecto que no queremos que salgan a la luz, por ejemplo las conexiones a nuestra base de datos, y para eso se utilizan las variables .env, dentro del archivo .env se establecen variables para lo que quieras mantener en secreto por ejemplo API Keys o en nuestro caso la conexión a la base de datos.

Además de hacer los cambios para poder conectarnos al abse de datos vamos a utilizar un software como MySQL Workbench para tener una mejor interfaz de la base de datos.

Migraciones: conceptos básicos absolutos / Migrations: The Absolute Basics

Para crear migraciones en la base de datos utilizamos el comando

```
php artisan migrate
```

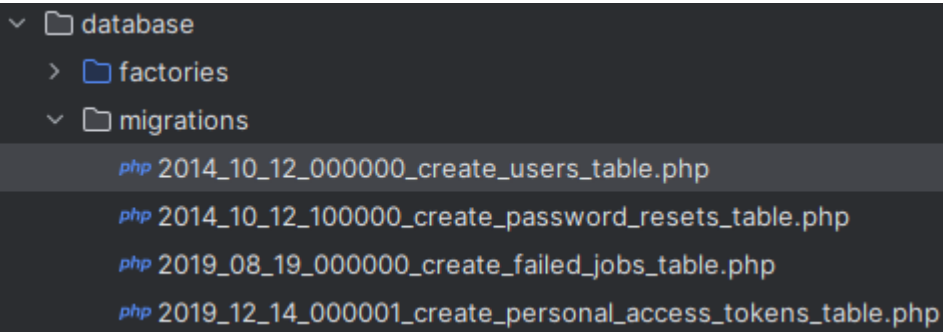
Y para volver atrás de una migración utilizamos el comando

```
php artisan migrate:rollback
```

Aca añadiré los demás comandos que se pueden utilizar con **migrate**

migrate	
migrate:fresh	Drop all tables and re-run all migrations
migrate:install	Create the migration repository
migrate:refresh	Reset and re-run all migrations
migrate:reset	Rollback all database migrations
migrate:rollback	Rollback the last database migration
migrate:status	Show the status of each migration

Para que los cambios que realizamos en el esquema de la base de datos en los archivos de migracion, tenemos que utilizar los comandos



```
php artisan migrate:rollback
php artisan migrate

// O por el contrario podemos utilizar

php artisan migrate:fresh
```

Elocuente y el patrón de registro activo / Eloquent and the Active Record Pattern

Ahora vamos a crear un usuario directamente desde nuestra terminal de comandos, por lo que vamos a ingresar a la VM webserver y vamos a correr los siguientes comandos

```
php artisan tinker

$user = new App\Models\User;

$user = new User;

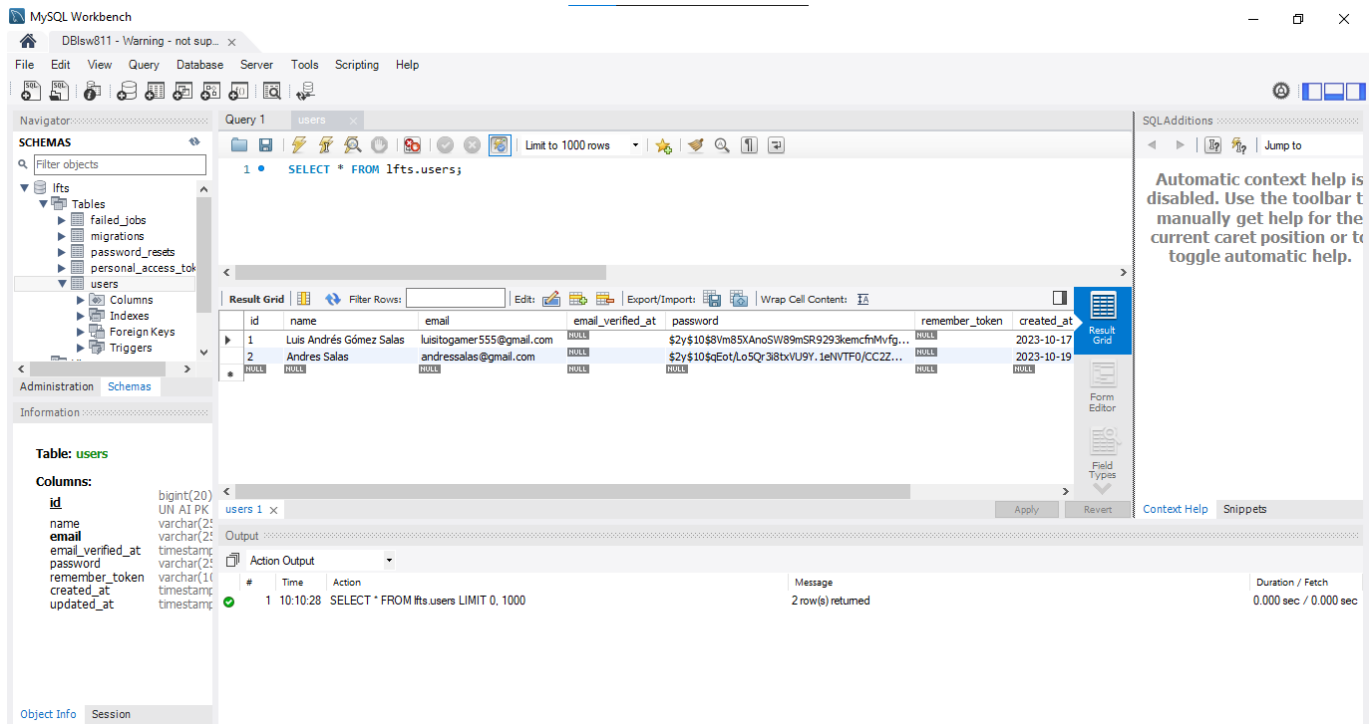
$user->name = 'Andres Salas'

$user->email = 'andressalas@gmail.com'

$user->password = bcrypt('!password')

$user->save();
```

Ahora vamos a Workbench a verificar si nuestro nuevo usuario fue creado



Como podemos visualizar nuestro usuario fue creado perfectamente.

Ahora si dentro de la terminal de comandos utilizamos la variable ``\$user` podremos observar la información del usuario recientemente creado

```
> $user
= App\Models\User {#6118
  name: "Andres Salas",
  email: "andressalas@gmail.com",
  #password: "$2y$10$qEot/Lo5Qr3i8txVU9Y.1eNVTF0/CC2ZwXZVoh0QbG8RSYxXfkFQi",
  updated_at: "2023-10-19 06:35:27",
  created_at: "2023-10-19 06:35:27",
  id: 2,
}
```

También podemos realizar cambios sobre ese mismo objeto, de la siguiente manera

```
$user->name = 'John Doe';

$user->save();
```

Podemos visualizar como los cambios son mostrados la proxima vez que llamemos la variable

```
> $user
= App\Models\User {#6118
  name: "John Doe",
  email: "andressalas@gmail.com",
  #password: "$2y$10$qEot/Lo5Qr3i8txVU9Y.1eNVTF0/CC2ZwXZVoh0QbG8RSXyxfkFQi",
  updated_at: "2023-10-19 06:41:35",
  created_at: "2023-10-19 06:35:27",
  id: 2,
}
```

También podemos buscar usuarios por medio del ID de la siguiente manera

```
> User::find(1);
[!] Aliasing 'User' to 'App\Models\User' for this Tinker session.
= App\Models\User {#6454
  id: 1,
  name: "Luis Andrés Gómez Salas",
  email: "luisitogamer555@gmail.com",
  email_verified_at: null,
  #password: "$2y$10$8Vm85XAnoSW89mSR9293kemcfmMvfg.u/acgCOw7b61wFR9wOSYm",
  #remember_token: null,
  created_at: "2023-10-17 21:03:37",
  updated_at: "2023-10-17 21:03:37",
}
```

En mi caso yo ya tenía un usuario guardado en la base de datos, pero para buscar el recién creado utilizamos el mismo comando con el id 2

```
> User::find(2);
= App\Models\User {#7076
  id: 2,
  name: "John Doe",
  email: "andressalas@gmail.com",
  email_verified_at: null,
  #password: "$2y$10$qEot/Lo5Qr3i8txVU9Y.1eNVTF0/CC2ZwXZVoh0QbG8RSXyxfkFQi",
  #remember_token: null,
  created_at: "2023-10-19 06:35:27",
  updated_at: "2023-10-19 06:41:35",
}
```

También podemos llamar todos los usuarios de esta manera

```
> User::all();
= Illuminate\Database\Eloquent\Collection {#7075
  all: [
    App\Models\User {#7073
      id: 1,
      name: "Luis Andrés Gómez Salas",
      email: "luisitogamer555@gmail.com",
      email_verified_at: null,
      #password: "$2y$10$8Vm85XAnoSW89mSR9293kemcfnMvfg.u/acgCOnW7b61wFR9wOSYm",
      #remember_token: null,
      created_at: "2023-10-17 21:03:37",
      updated_at: "2023-10-17 21:03:37",
    },
    App\Models\User {#6816
      id: 2,
      name: "John Doe",
      email: "andressalas@gmail.com",
      email_verified_at: null,
      #password: "$2y$10$qEot/Lo5Qr3i8txVU9Y.1eNVTF0/CC2ZwXZVoh0QbG8RSXyxfkFQi",
      #remember_token: null,
      created_at: "2023-10-19 06:35:27",
      updated_at: "2023-10-19 06:41:35",
    },
  ],
}
```

Podemos guardar el resultado de esta funcion dentro de una variable

```
> $users = User::all();
= Illuminate\Database\Eloquent\Collection {#6116
  all: [
    App\Models\User {#6121
      id: 1,
      name: "Luis Andrés Gómez Salas",
      email: "luisitogamer555@gmail.com",
      email_verified_at: null,
      #password: "$2y$10$8Vm85XAnoSW89mSR9293kemcfnMvfg.u/acgCOnW7b61wFR9wOSYm",
      #remember_token: null,
      created_at: "2023-10-17 21:03:37",
      updated_at: "2023-10-17 21:03:37",
    },
    App\Models\User {#7079
      id: 2,
      name: "John Doe",
      email: "andressalas@gmail.com",
      email_verified_at: null,
      #password: "$2y$10$qEot/Lo5Qr3i8txVU9Y.1eNVTF0/CC2ZwXZVoh0QbG8RSXyxfkFQi",
      #remember_token: null,
      created_at: "2023-10-19 06:35:27",
      updated_at: "2023-10-19 06:41:35",
    },
  ],
}
```

Y retornar lo que queramos de esta variable

```
> $users->pluck('name');
= Illuminate\Support\Collection {#7075
  all: [
    "Luis Andrés Gómez Salas",
    "John Doe",
  ],
}
```

Tambien podemos retornar el primer dato de esa variable

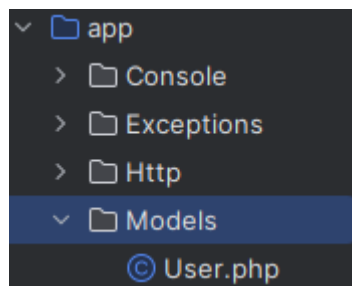
```
> $users->first()
= App\Models\User {#6121
  id: 1,
  name: "Luis Andrés Gómez Salas",
  email: "luisitogamer555@gmail.com",
  email_verified_at: null,
  #password: "$2y$10$8Vm85XAnoSW89mSR9293kemcfmMvfg.u/acgCOnW7b61wFR9wOSYm",
  #remember_token: null,
  created_at: "2023-10-17 21:03:37",
  updated_at: "2023-10-17 21:03:37",
}
```

O tambien podemos hacerlo como un array

```
> $users[1]
= App\Models\User {#7079
  id: 2,
  name: "John Doe",
  email: "andressalas@gmail.com",
  email_verified_at: null,
  #password: "$2y$10$qEot/Lo5Qr3i8txVU9Y.1eNVTF0/CC2ZwXZVoh0QbG8RSXyxfkFQi",
  #remember_token: null,
  created_at: "2023-10-19 06:35:27",
  updated_at: "2023-10-19 06:41:35",
}
```

Crear un modelo de Post y migrarlo / Make a Post Model adn Migration

Vamos a eliminar la clase Post que creamos para crar una nueva con un modelo elocuente.



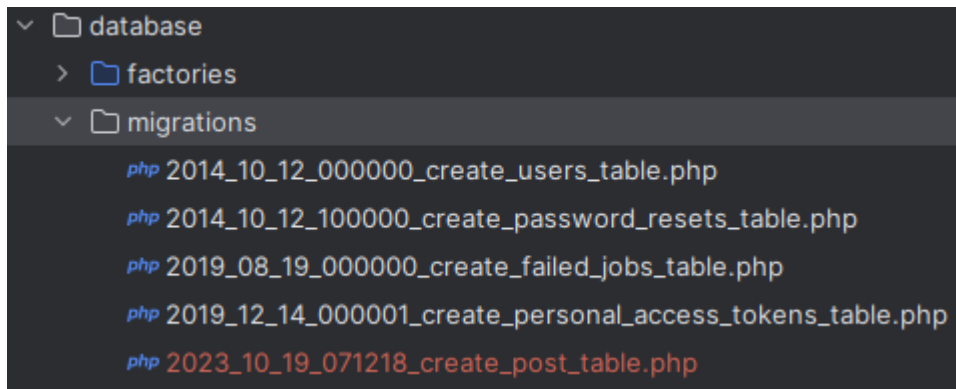
Ahora nos dirigimos a la terminal de comandos y junto a *php artisan* vamos a utilizar una extension llamada `make` la cual funciona para crear archivos.

Primero utilizamos este comando para crear una tabla de post

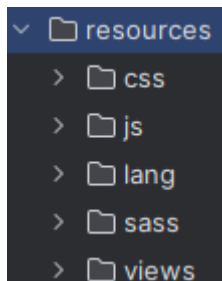
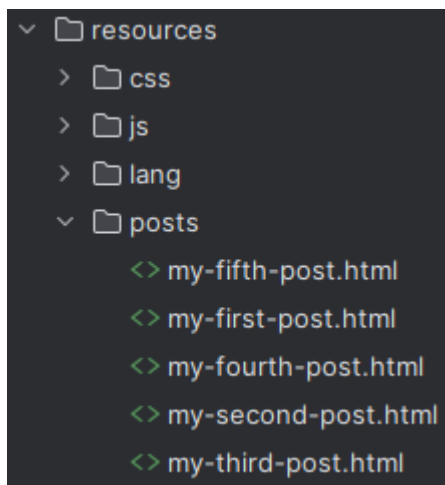
```
php artisan make:migration create_post_table
```

```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ php artisan make:migration create_post_table
Created Migration: 2023_10_19_071218_create_post_table
```

Y como podemos observar, en la carpeta de migrations en el folder de nuestro proyecto se creó una nueva migration



Ahora eliminamos la carpeta posts que habiamos creado en el pasado ya que ahora los posts se cargaran desde la base de datos



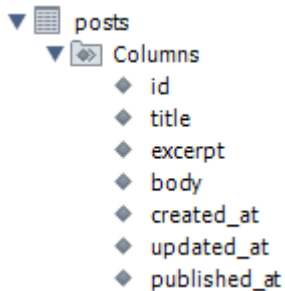
Ahora vamos a la migracion de posts y editamos el codigo

```
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->text('excerpt');
        $table->text('body');
        $table->timestamps();
        $table->timestamp('published_at')->nullable();
    });
}
```

Ahora nos movemos a la terminal y migramos la tabla a la base de datos

```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ php artisan migrate
Migrating: 2023_10_19_071218_create_posts_table
Migrated: 2023_10_19_071218_create_posts_table (125.49ms)
```

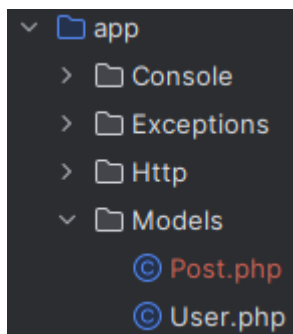
Y ahora checamos en Workbench que nuestra nueva tabla haya sido creada correctamente



Ahora vamos a crear el respectivo modelo elocuente, recordar que el nombre del modelo siempre debe ser el nombre de la tabla preo en singular, en este caso la tabla se llama **posts** y el modelo se llama **Post**

```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ php artisan make:model Post
Model created successfully.
```

Y ahora podemos observar que tenemos una clase Post en nuestra carpeta de modelos



Ahora vamos a utilizar tinker y vamos a intentar traer todos los posts que tengamos en la base de datos, aunque de momento no tenemos ninguno.

```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ php artisan tinker
Psy Shell v0.11.21 (PHP 8.2.7 - cli) by Justin Hileman
> App\Models\Post::all();
= Illuminate\Database\Eloquent\Collection {#6860
  all: [],
}
```

Tambien podemos utilizar count() para verificar cuantos posts tenemos en la base de datos

```
> App\Models\Post::count();
= 0
```

Ahora vamos a crear un nuevo post desde la terminal


```
> $post = new App\Models\Post;
= App\Models\Post {#6816}

> $post->title = 'My First Post';
= "My First Post"

> $post->excerpt = 'Lorem ipsum dolor sit amet.';
= "Lorem ipsum dolor sit amet."

> $post->body = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.';
= "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

> $post->save();

Illuminate\Database\QueryException SQLSTATE[42S22]: Column not found: 1054 Unknown column 'excerpt' in 'field list' (SQL: insert into 'posts' ('title', 'excerpt', 'body', 'updated_at', 'created_at') values ('My First Post', 'Lorem ipsum dolor sit amet.', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.', '2023-10-19 07:40:51', '2023-10-19 07:40:51')).

> $post->save();
= true
```

Revisamos Workbench para visualziar nuestro nuevo posts

	id	title	excerpt	body	created_at	updated_at	published_at
▶	1	My First Post	Lorem ipsum dolor sit amet.	Lorem ipsum dolor sit amet, consectetur adipisci...	2023-10-19 07:40:51	2023-10-19 07:40:51	NULL
✱	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Y desde nuestra terminal volvemos a utilizar las funciones de count() y all()

```
> use App\Models\Post;
> Post::count();
= 1

> Post::all();
= Illuminate\Database\Eloquent\Collection {#6454
  all: [
    App\Models\Post {#6114
      id: 1,
      title: "My First Post",
      excerpt: "Lorem ipsum dolor sit amet.",
      body: "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.",
      created_at: "2023-10-19 07:40:51",
      updated_at: "2023-10-19 07:40:51",
      published_at: null,
    },
  ],
}
```

Ahora vamos a cambiar en el archivo `web.php` el siguiente codigo ya que no necesitamos pasar el slug, si no el id del post

```
Route::get('posts/{post}', function ($id) {

    return view('post', [
        'post' => Post::findOrFail($id)
    ]);
});
```

Y ahora visualizamos la pagina web para comprobar que todo este en orden.

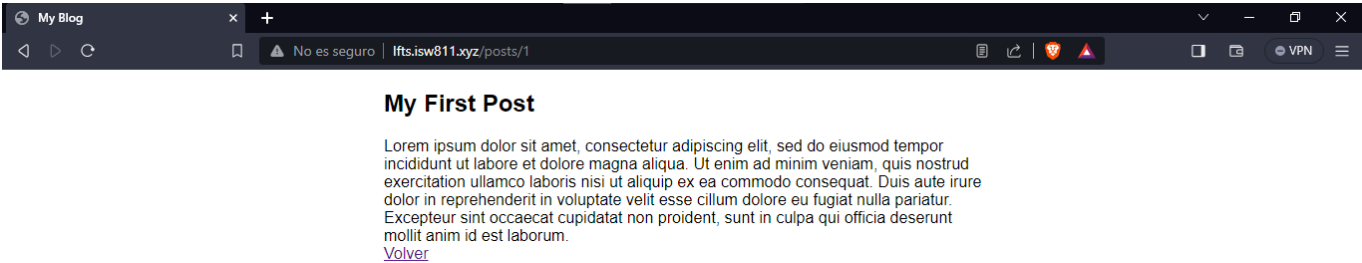


Y ahora necesitamos cambiar en el archivo `posts.blade.php` el *slug* por *id*

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

@section('content')
    @foreach ($posts as $post)
        <article>
            <h1><a href="/posts/{{ $post->id }}"> {{ $post->title }} </a></h1>
            <div>
                {{ $post->excerpt }}
            </div>
        </article>
    @endforeach
@endsection
```

Y como podemos ver ahora podemos ingresar al post y se carga la información.



Vamos a crear un post mas

```
> $post = new App\Models\Post;
= App\Models\Post {#6117}

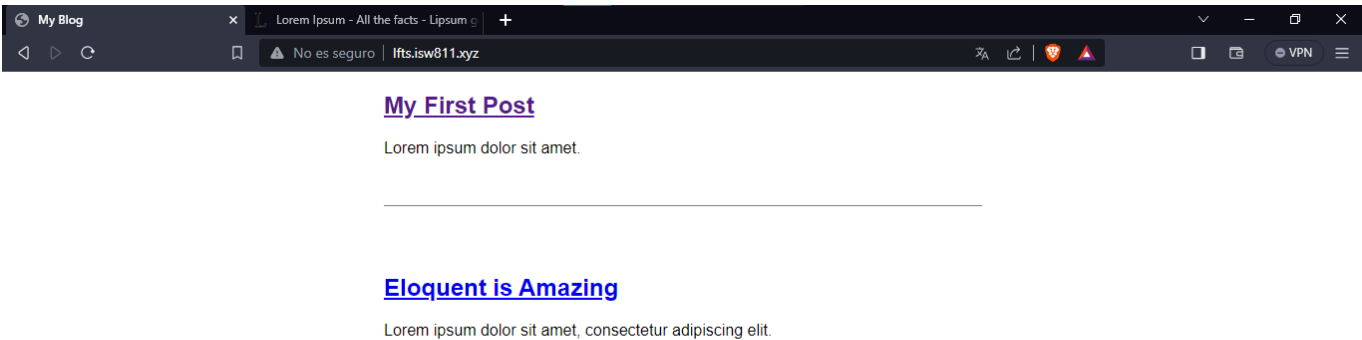
> $post->title = 'Eloquent is Amazing';
= "Eloquent is Amazing"

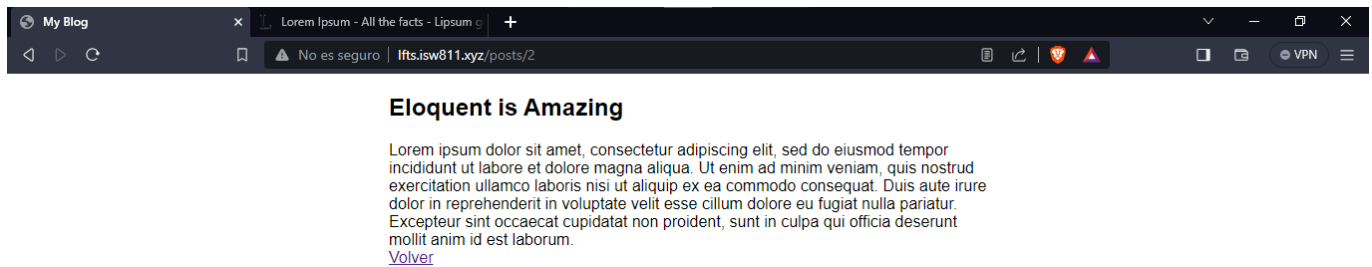
> $post->excerpt = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit.';
= "Lorem ipsum dolor sit amet, consectetur adipiscing elit."

> $post->body = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.';
= "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

> $post->save();
= true
```

Y visualizamos la web para visualizar nuestro nuevo post





Actualizaciones elocuentes y escaneo HTML / Eloquent Updates and HTML Escaping

Vamos a cambiar el body del posts para que al cargarse en el html este se cargue en una etiqueta `<p>` en vez de cargarse en texto plano.

```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ php artisan tinker
Psy Shell v0.11.21 (PHP 8.2.7 - cli) by Justin Hileman
> $post = App\Models\Post::first();
= App\Models\Post {#6816
  id: 1,
  title: "My First Post",
  excerpt: "Lorem ipsum dolor sit amet.",
  body: "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad min
im veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate vel
it esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est
laborum.",
  created_at: "2023-10-19 07:40:51",
  updated_at: "2023-10-19 07:40:51",
  published_at: null,
}
> $post->body;
= "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim venia
m, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum
."
> $post->body = '<p>' . $post->body . '<p>';
= "<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim ve
niam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit es
se cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est labo
rum.<p>"
> $post->save();
= true
```

Ahora vemos como nuestro posts esta en una etiqueta

en vez de en texto plano y se carga el css de la etiqueta.



My First Post

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

[Volver](#)

Ahora repetimos el mismo proceso con el otro post.

```
> $post = App\Models\Post::find(2);
= App\Models\Post {#6119
  id: 2,
  title: "Eloquent is Amazing",
  excerpt: "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
  body: "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.",
  created_at: "2023-10-19 08:08:13",
  updated_at: "2023-10-19 08:08:13",
  published_at: null,
}

> $post->body;
= "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

> $post->body = '<p>' . $post->body . '<p>';
= "<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.<p>"

> $post->save();
= true
```

Para poder cargar etiquetas desde la base de datos hay que tener la etique donde se cargue la informacion de esta manera. {!! \$post->title !!}, de lo contrario no funcionaria.

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

@section('content')
<article>

    <h1> {!! $post->title !!} </h1>

    <div>
        {!! $post->body !!}
    </div>

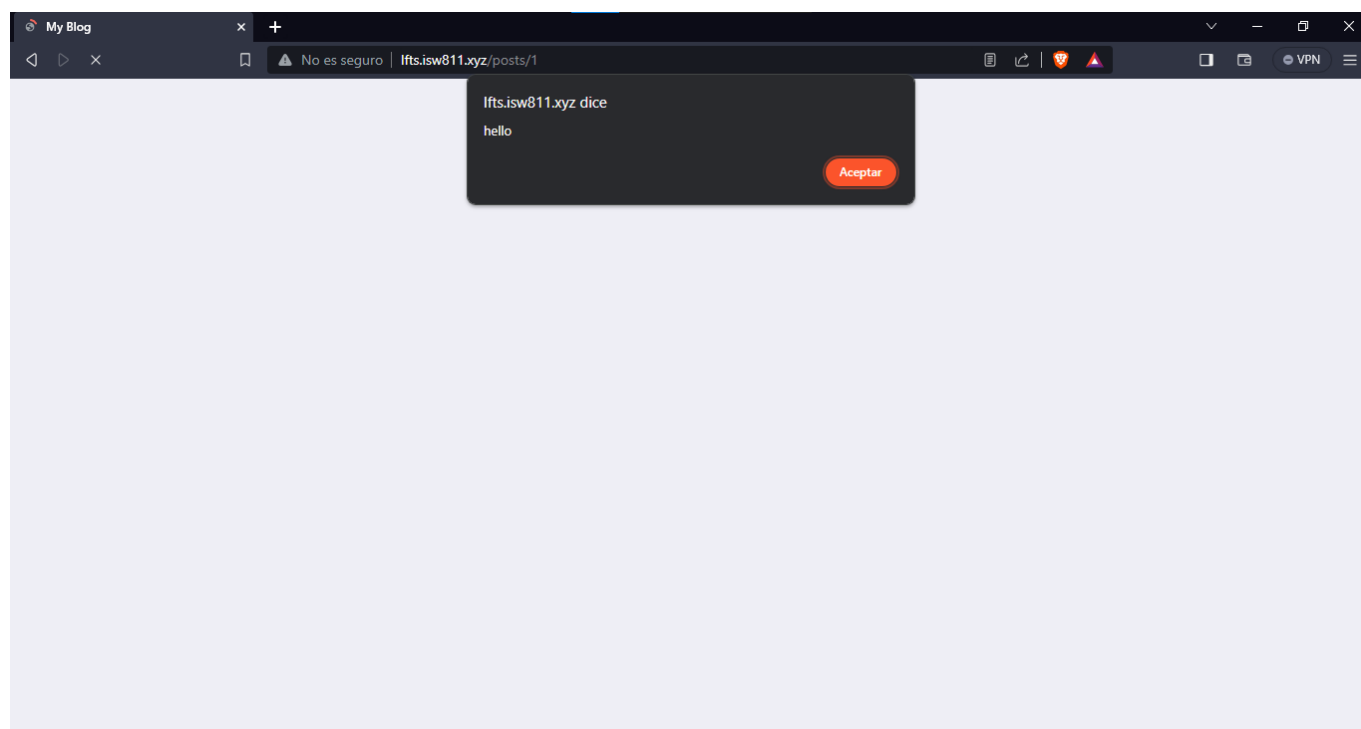
</article>
```

```
<a href="/">Volver</a>
@endsection
```

Ahora vamos a guardar una alerta en el titulo del primer post. Esto es normalmente utilizado para inyectar codigo malicioso dentro de nuestro codigo.

```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ php artisan tinker
Psy Shell v0.11.21 (PHP 8.2.7 - cli) by Justin Hileman
> $post = App\Models\Post::first();
= App\Models\Post {#6816
  id: 1,
  title: "My First Post",
  excerpt: "Lorem ipsum dolor sit amet.",
  body: "<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate
velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id
est laborum.<p>",
  created_at: "2023-10-19 07:40:51",
  updated_at: "2023-10-19 08:17:18",
  published_at: null,
}
> $post->title = 'My Post <script>alert("hello")</script>';
= "My Post <script>alert("hello")</script>"
> $post->save();
= true
```

Y vemos como al cargar la web podemos visualizar la alerta.



Cualquier persona que cree un post dentro de nuestra web y sepa un poco del tema podria inyectar javascript dentro de nuestro codigo por lo que lo conveniente sería mantener las etiquetas de esta manera

```
{{ $post->title }}
```

para evitar esos problemas, ahora vi visualizamos la web, la alerta ya no cargaría.



Tres formas de mitigar las vulnerabilidades de las asignaciones masivas / 3 Ways to Mitigate Mass Assignment Vulnerabilities

Vamos a crear un nuevo post

```
> use App\Models\Post;
> $post = new Post;
= App\Models\Post {#6454}
> $post->title = 'My Third Post';
= "My Third Post"
> $post->excerpt = 'Excerpt of post';
= "Excerpt of post"
> $post->body = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.';
= "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
> $post->save();
= true
```

Ahora vamos a agregar un parametro nuevo en la clase de Post para poder mitigar ciertas vulnerabilidades dentro de los Post, lo que no este dentro de este nuevo atributo será ignorado a la hora de crear el post, para esto utilizamos \$fillable.

```
protected $fillable = ['title', 'excerpt', 'body'];
```

Ahora creamos un nuevo post, pero ademas de title, excerpt y body, agregaremos id y veremos como este será ignorado

```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ php artisan tinker
Psy Shell v0.11.21 (PHP 8.2.7 - cli) by Justin Hileman
> Post::create(['title' => 'My Fourth Post', 'excerpt' => 'Excerpt of post', 'body' => 'Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.', 'id' => 1000000]);
[!] Aliasing 'Post' to 'App\Models\Post' for this Tinker session.
= App\Models\Post {#6120
  title: "My Fourth Post",
  excerpt: "Excerpt of post",
  body: "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.",
  updated_at: "2023-10-19 08:52:20",
  created_at: "2023-10-19 08:52:20",
  id: 4,
}
```

También está el atributo `$guarded` que lo que hace es rellenar todos los atributos a excepción de los guardados dentro de la variable

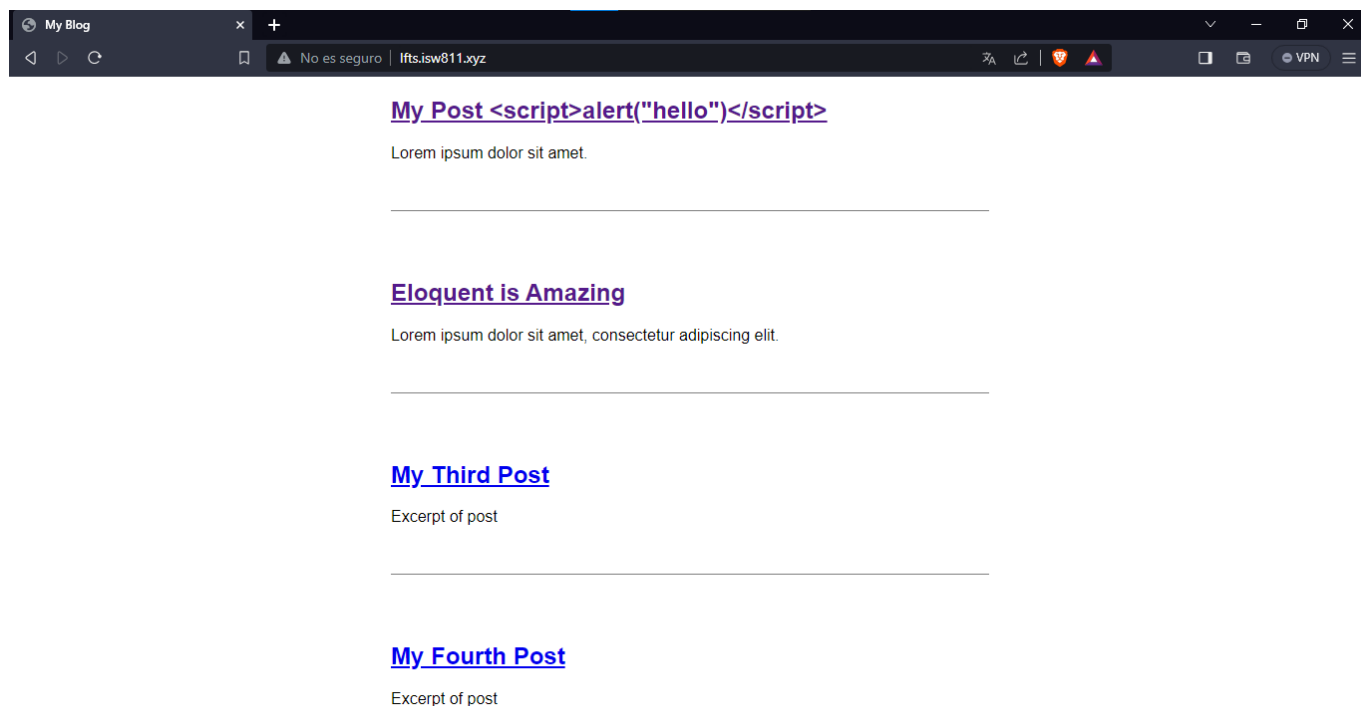
```
protected $guarded = ['id'];
```

Enlace del modelo de ruta / Route Model Binding

Vamos a modificar el código en el archivo `web.php` esta modificación funciona exactamente que la anterior cuando se le pasaba el id del post por parámetro, solo que como en el wildcard estamos pasando `{post}` en los parámetros al ponerlo de esta manera funcionaría igualmente.

```
Route::get('posts/{post}', function (Post $post) {

    return view('post', [
        'post' => Post::findOrFail($post)
    ]);
});
```



Ahora dentro del archivo de migracion `create_post` vamos a añadir un nuevo atributo el cual será el `slug` este llevará `unique()` ya que no queremos que el slug se repita.

```
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->string('slug')->unique();
        $table->string('title');
        $table->text('excerpt');
        $table->text('body');
        $table->timestamps();
        $table->timestamp('published_at')->nullable();
    });
}
```

Ahora vamos a refrescar la migración para que los cambios se vean reflejados en la base de datos

```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ php artisan migrate:fresh
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (100.08ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (46.04ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (46.67ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated: 2019_12_14_000001_create_personal_access_tokens_table (76.58ms)
Migrating: 2023_10_19_074650_create_posts_table
Migrated: 2023_10_19_074650_create_posts_table (43.63ms)
```

Ademas de hacer esto, tenemos que ir a la base de datos y hacer un insert de los posts ya creados, ya que al utilizar `fresh` este eliminará toda la información que se encontraba en la tabla

Una vez haber hecho el insert y añadido el slug dentro de ese mismo insert podemos ver como nuestros posts y la tabla tienen el slug

	id	slug	title	excerpt	body	created_at
▶	1	my-first.post	My First Post	Lorem ipsum dolor sit amet.	<p>Lorem ipsum dolor sit amet, consectetur adi...	2023-10-19 07:40:51
	2	my-second.post	Eloquent is Amazing	Lorem ipsum dolor sit amet, consectetur adipisci...	<p>Lorem ipsum dolor sit amet, consectetur adi...	2023-10-19 08:08:13
	3	my-third.post	My Third Post	Excerpt of post	Lorem ipsum dolor sit amet, consectetur adipisci...	2023-10-19 08:42:16
	4	my-fourth.post	My Fourth Post	Excerpt of post	Lorem ipsum dolor sit amet, consectetur adipisci...	2023-10-19 08:52:20
*	NULL	NULL	NULL	NULL	NULL	NULL

Ahora vamos a `web.php` y editamos el código para que no se filtre por id si no por slug

```
Route::get('posts/{post:slug}', function (Post $post) {

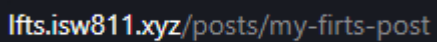
    return view('post', [
        'post' => $post
    ]);
});
```

Además de esto vamos al archivo `post.blade.php` a pasar por parametro el slug en vez del id

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

@section('content')
    @foreach ($posts as $post)
        <article>
            <h1><a href="/posts/{{ $post->slug }}"> {{ $post->title }} </a></h1>
            <div>
                {{ $post->excerpt }}
            </div>
        </article>
    @endforeach
@endsection
```

Ahora vamos a visualizar los cambios en la url de la nuestra web al momento de ingresar en un post



Ahora vamos a ver otro metodo para encontrar el identificador de un post, dentro de la clase Post, vamos a añadir este código.

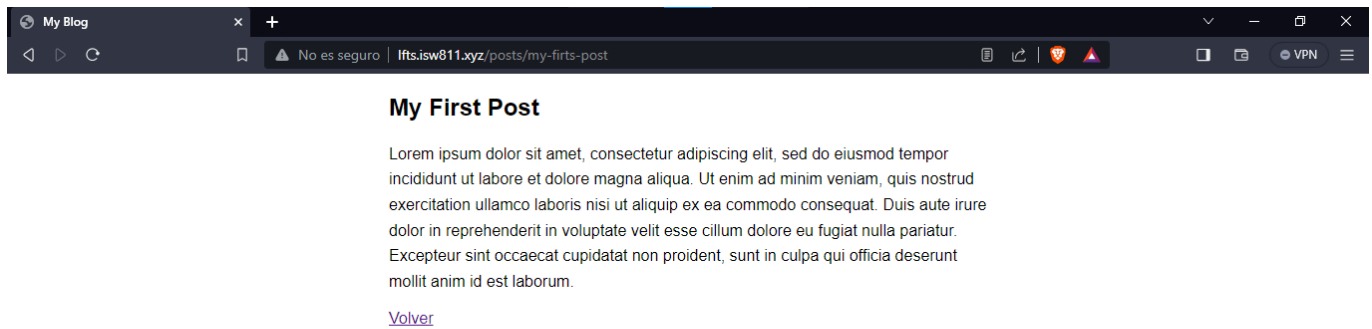
```
public function getRouteKeyName()
{
    return 'slug';
}
```

Ahora vamos a `web.php` y cambiamos los siguiente

```
Route::get('posts/{post}', function (Post $post) {

    return view('post', [
        'post' => $post
    ]);
});
```

Y ahora visualizamos la web para ver como los post se cargan por el slug aunque no este tipado en el wildcard



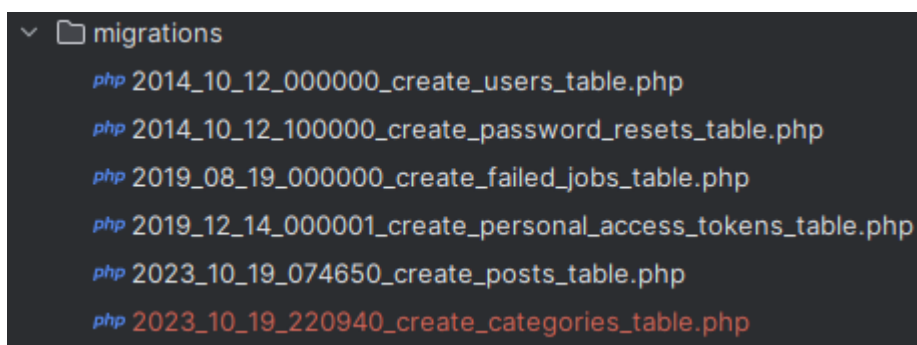
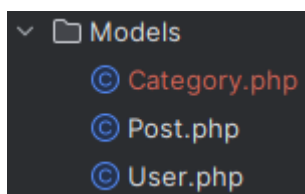
Tu primera relación elocuente / Your Firts Eloquent Relationship

Lo siguiente será figurar las categorías de cada post, para en el futuro poder filtrar por categoría.

Vamos a ver una nueva manera de crear una tabla, en este caso vamos a crear el Model de Category y vamos a agregar -m para crear la migración que posteriormente se convertirá en una tabla de la base de datos.

```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ php artisan make:model Category -m
Model created successfully.
Created Migration: 2023_10_19_220940_create_categories_table
```

Como vemos se creó tanto el Model como la migración.



En la migración modificaremos el código para agregar las columnas que necesitamos

```
public function up()
{
```

```
Schema::create('categories', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('slug');
    $table->timestamps();
});
}
```

Y ahora crearemos una foreign key para enlazar la tabla categories con la de posts

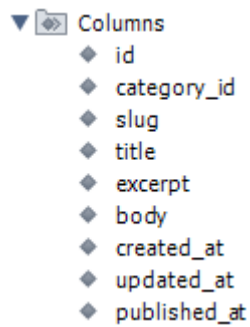
```
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->foreignId('category_id');
        $table->string('slug')->unique();
        $table->string('title');
        $table->text('excerpt');
        $table->text('body');
        $table->timestamps();
        $table->timestamp('published_at')->nullable();
    });
}
```

Y ahora refrescamos la base de datos para agregar la nueva tabla y la nueva columna

```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ php artisan migrate:fresh
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (157.49ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (48.56ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (70.49ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated: 2019_12_14_000001_create_personal_access_tokens_table (99.27ms)
Migrating: 2023_10_19_074650_create_posts_table
Migrated: 2023_10_19_074650_create_posts_table (213.22ms)
Migrating: 2023_10_19_220940_create_categories_table
Migrated: 2023_10_19_220940_create_categories_table (29.55ms)
```

Y como vemos ya estaría la nueva tabla creada y la nueva columna en la tabla posts

```
categories
├── Columns
│   ├── id
│   ├── name
│   ├── slug
│   ├── created_at
│   └── updated_at
```



Ahora vamos a crear una nueva categoría en la base de datos

```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ php artisan tinker
Psy Shell v0.11.21 (PHP 8.2.7 - cli) by Justin Hileman
> use App\Models\Category;
> $c = new Category;
= App\Models\Category {#6113}

> $c->name = 'Personal';
= "Personal"

> $c->slug = 'personal';
= "personal"

> $c->save();
= true
```

Vamos a crear dos más.

```
> $c = new Category;
= App\Models\Category {#6453}

> $c->name = 'Work';
= "Work"

> $c->slug = 'work';
= "work"

> $c->save();
= true

> $c = new Category;
= App\Models\Category {#6858}

> $c->name = 'Hobbies';
= "Hobbies"

> $c->slug = 'hobbies';
= "hobbies"

> $c->save();
= true
```

Ahora en nuestra base de datos tenemos 3 categorías.

id	name	slug	created_at	updated_at
1	Personal	personal	2023-10-19 22:32:59	2023-10-19 22:34:45
2	Work	work	2023-10-19 22:35:02	2023-10-19 22:35:02
3	Hobbies	hobbies	2023-10-19 22:35:35	2023-10-19 22:35:35

Creamos nuevos post

```
> Post::create([
  'title' => 'My Family Post',
  'excerpt' => 'Excerpt for my post',
  'body' => 'Lorem ipsum dolar sit amet.',
  'slug' => 'my-family-post',
  'category_id' => 1
]);
= App\Models\Post {#6122
  title: "My Family Post",
  excerpt: "Excerpt for my post",
  body: "Lorem ipsum dolar sit amet.",
  slug: "my-family-post",
  category_id: 1,
  updated_at: "2023-10-19 22:41:37",
  created_at: "2023-10-19 22:41:37",
  id: 1,
}
```

Ya que tenemos un post creado vamos a hacer un select en el workbench donde la categoría sea 1

1 • `SELECT * from posts where category_id =1;`

Result Grid

	id	category_id	slug	title	excerpt	body	created_at	updated_at
▶	1	1	my-family-post	My Family Post	Excerpt for my post	Lorem ipsum dolar sit amet.	2023-10-19 22:41:37	2023-10-19 22:41:37
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Result Grid

Ahora creamos otros dos post más

```
> Post::create([
  'title' => 'My Work Post',
  'excerpt' => 'Excerpt for my post',
  'body' => 'Lorem ipsum dolar sit amet.',
  'slug' => 'my-work-post',
  'category_id' => 2
]);
= App\Models\Post {#7081
  title: "My Work Post",
  excerpt: "Excerpt for my post",
  body: "Lorem ipsum dolar sit amet.",
  slug: "my-work-post",
  category_id: 2,
  updated_at: "2023-10-19 22:44:58",
  created_at: "2023-10-19 22:44:58",
  id: 2,
}

> Post::create([
  'title' => 'My Hobbies Post',
  'excerpt' => 'Excerpt for my post',
  'body' => 'Lorem ipsum dolar sit amet.',
  'slug' => 'my-hobbies-post',
  'category_id' => 3
]);
= App\Models\Post {#6135
  title: "My Hobbies Post",
  excerpt: "Excerpt for my post",
  body: "Lorem ipsum dolar sit amet.",
  slug: "my-hobbies-post",
  category_id: 3,
  updated_at: "2023-10-19 22:45:31",
  created_at: "2023-10-19 22:45:31",
  id: 3,
}
```

Y vemos en la base de datos los 3 posts

id	category_id	slug	title	excerpt	body	created_at	updated_at
1	1	my-family-post	My Family Post	Excerpt for my post	Lorem ipsum dolar sit amet.	2023-10-19 22:41:37	2023-10-19 22:41:37
2	2	my-work-post	My Work Post	Excerpt for my post	Lorem ipsum dolar sit amet.	2023-10-19 22:44:58	2023-10-19 22:44:58
3	3	my-hobbies-post	My Hobbies Post	Excerpt for my post	Lorem ipsum dolar sit amet.	2023-10-19 22:45:31	2023-10-19 22:45:31

Ahora vamos a la Post a crear la relación elocuente

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

Y ahora en nuestra terminal podemos visualizar los detalles del post y de su categoria

```

> $post = App\Models\Post::first();
= Illuminate\Database\Eloquent\Relations\BelongsTo {#6814}
  id: 1,
  category_id: 1,
  slug: "my-family-post",
  title: "My Family Post",
  excerpt: "Excerpt for my post",
  body: "Lorem ipsum dolar sit amet.",
  created_at: "2023-10-19 22:41:37",
  updated_at: "2023-10-19 22:41:37",
  published_at: null,
}

> $post->category();
= Illuminate\Database\Eloquent\Relations\BelongsTo {#6108}

> $post->category;
= App\Models\Category {#6114}
  id: 1,
  name: "Personal",
  slug: "personal",
  created_at: "2023-10-19 22:32:59",
  updated_at: "2023-10-19 22:34:45",
}

> $post
= App\Models\Post {#6814}
  id: 1,
  category_id: 1,
  slug: "my-family-post",
  title: "My Family Post",
  excerpt: "Excerpt for my post",
  body: "Lorem ipsum dolar sit amet.",
  created_at: "2023-10-19 22:41:37",
  updated_at: "2023-10-19 22:41:37",
  published_at: null,
  category: App\Models\Category {#6114}
    id: 1,
    name: "Personal",
    slug: "personal",
    created_at: "2023-10-19 22:32:59",
    updated_at: "2023-10-19 22:34:45",
  },
}

```

Editamos el archivo post.blade.php para que tambien muestre en la vista la cterogria de post

```

@extends('layouts.layout')

@section('content')
    @foreach ($posts as $post)
        <article>
            <h1>
                <a href="/posts/{{ $post->slug }}"> {{ $post->title }} </a>
            </h1>

            <p>
                <a href="#"> {{ $post->category->name }} </a>
            </p>

            <div>

```

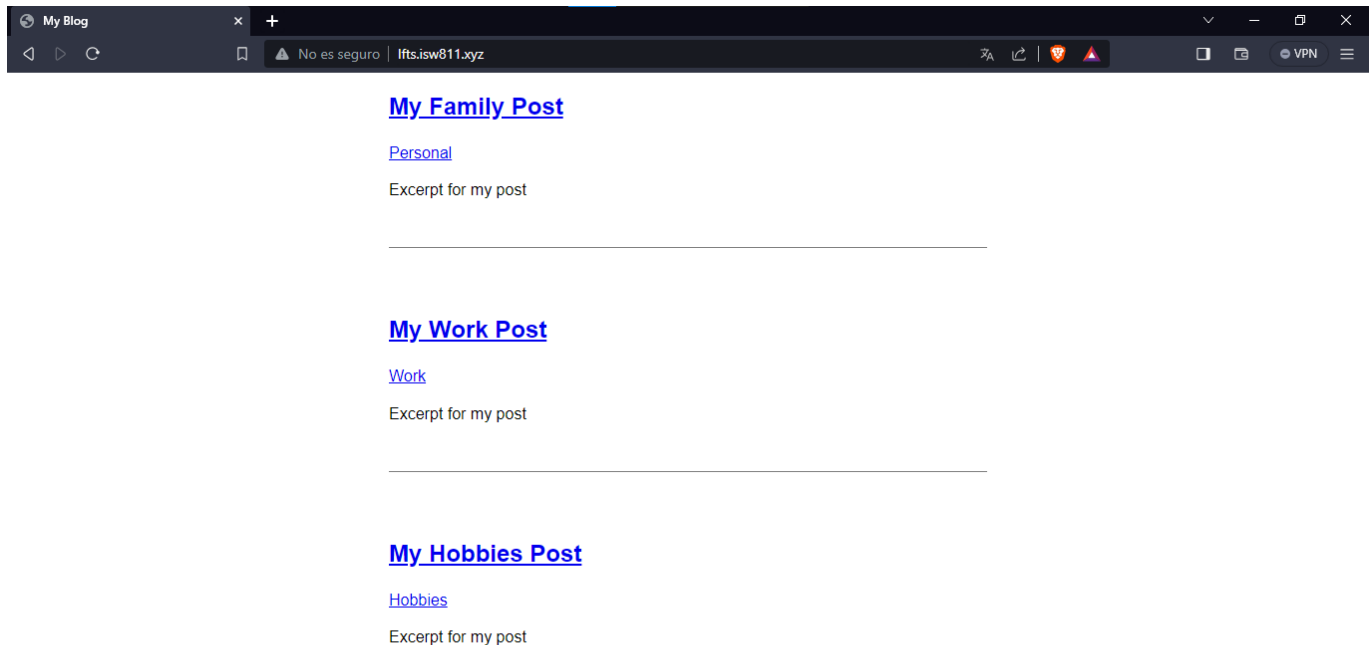


```

        {{$post->excerpt}}
    </div>
</article>
@endforeach
@endsection

```

Visualizamos la pagina



Y como vemos los posts se cargan son su respectiva categoría.

Mostrar todas las publicaciones asociadas con una categoría / Show All Posts Associated With a Category

Creamos una nueva ruta en web.php

```

Route::get('categories/{category}', function (Category $category) {
    return view('posts', [
        'posts' => $category->posts
    ]);
});

```

Creamos una nueva funcion en la clase Category para llamar todos los post que tienen esa misma categoría

```

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

```

Revisamos por en la terminal como funciona esta nueva función

```
> App\Models\Category::first();
= Illuminate\Database\Eloquent\Collection {#6819
  id: 1,
  name: "Personal",
  slug: "personal",
  created_at: "2023-10-19 22:32:59",
  updated_at: "2023-10-19 22:34:45",
}

> App\Models\Category::first()->posts;
= Illuminate\Database\Eloquent\Collection {#6862
  all: [
    App\Models\Post {#7077
      id: 1,
      category_id: 1,
      slug: "my-family-post",
      title: "My Family Post",
      excerpt: "Excerpt for my post",
      body: "Lorem ipsum dolar sit amet.",
      created_at: "2023-10-19 22:41:37",
      updated_at: "2023-10-19 22:41:37",
      published_at: null,
    },
  ],
}
```

Ahora modificamos las vistas para poder cargar las categorias por medio del slug y no del id

```
//show.blade.php
@extends('layouts.layout')

@section('content')
    <article>

        <h1> {{ $post->title }} </h1>

        <p>
            <a href="/categories/{{ $post->category->slug }}"> {{ $post->category-
>name }} </a>
        </p>

        <div>
            {!! $post->body !!}
        </div>

    </article>

    <a href="/">Volver</a>
@endsection
```

```
//index.blade.php
@extends('layouts.layout')
```

```

@section('content')
    @foreach ($posts as $post)
        <article>
            <h1>
                <a href="/posts/{{ $post->slug }}"> {{ $post->title }} </a>
            </h1>

            <p>
                <a href="/categories/{{ $post->category->slug }}"> {{ $post->category->name }} </a>
            </p>

            <div>
                {{ $post->excerpt }}
            </div>
        </article>
    @endforeach
@endsection

```

Editamos la funcion de la ruta en web.php para que busque la categoria por el slug

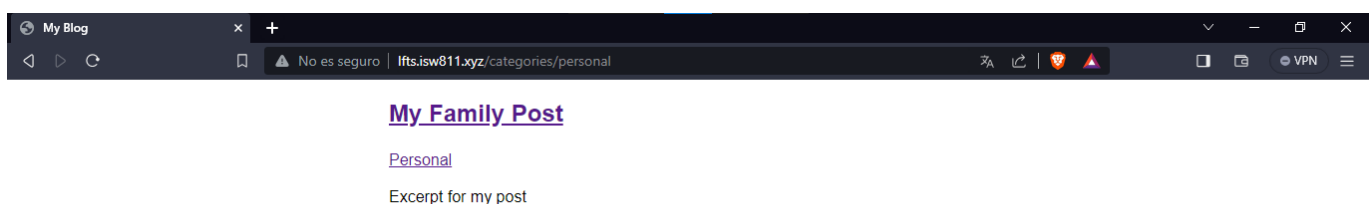
```

Route::get('categories/{category:slug}', function (Category $category) {

    return view('posts', [
        'posts' => $category->posts
    ]);
});

```

Y visualizamos la pagina de la categoría donde se cargan todos los posts asociados a esa categoría



El mecanismo de relojería y el problema N+1 / Clockwork, and the N+1 Problem

Tenemos un problema y es que cuando se estan cargando los posts en la main page se esta corriendo el mismo query 3 veces y eso ralentiza pa pagina, por lo que lo idean sería que solo se cargue una vez.

Vamos a utilizar una herramienta llamada clockwork que instalaremos dentro de nuestra VM webserver con el siguiente comando

```
$ composer require itsgoingd/clockwork
```

```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ composer require itsgoingd/clockwork
./composer.json has been updated
Running composer update itsgoingd/clockwork
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
  - Locking itsgoingd/clockwork (v5.1.12)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Downloading itsgoingd/clockwork (v5.1.12)
  - Installing itsgoingd/clockwork (v5.1.12): Extracting archive
Package fruitcake/laravel-cors is abandoned, you should avoid using it. No replacement was suggested.
Package swiftmailer/swiftmailer is abandoned, you should avoid using it. Use symfony/mailer instead.
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: facade/ignition
Discovered Package: fruitcake/laravel-cors
Discovered Package: itsgoingd/clockwork
Discovered Package: laravel/sail
Discovered Package: laravel/sanctum
Discovered Package: laravel/tinker
Discovered Package: laravel/ui
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Package manifest generated successfully.
79 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force
No publishable resources for tag [laravel-assets].
Publishing complete.
No security vulnerability advisories found.
Using version ^5.1 for itsgoingd/clockwork
```

Ademas necesitaremos la extension del navegador que encontramos en el siguiente repositorio de github

[-Clockwork](#)

Con esta extension podemos ver desde los dev tools del navegador los querys que se estan corriendo al momento de cargar la pagina

The screenshot shows a web browser with the URL `lfts.isw811.xyz` displaying a page titled "My Family Post" with a "Personal" link. Below the browser, the Clockwork development tool is open, showing the "Database" tab. The "Queries" section displays a list of database queries executed during the page load.

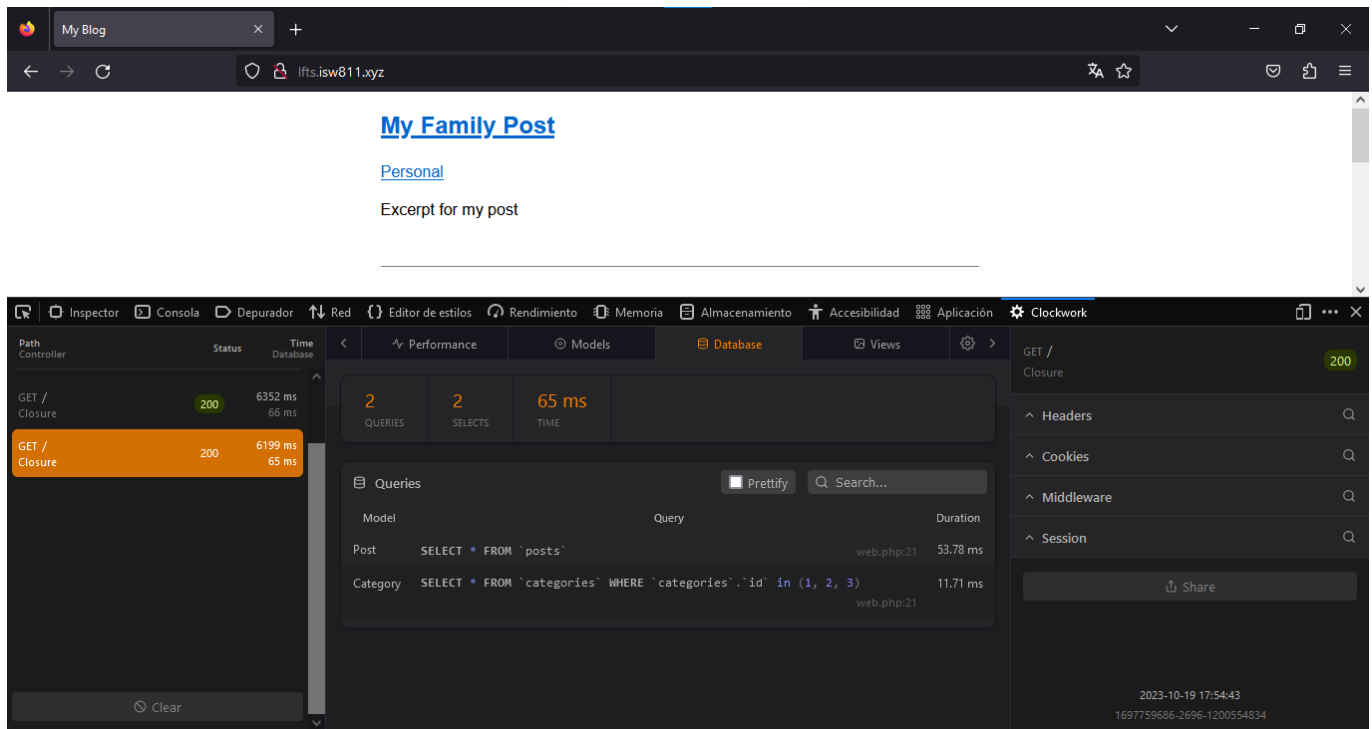
Model	Query	Duration
Post	<code>SELECT * FROM `posts`</code>	54.97 ms
Category	<code>SELECT * FROM `categories` WHERE `categories`.`id` = 1 LIMIT 1</code>	3.71 ms
Category	<code>SELECT * FROM `categories` WHERE `categories`.`id` = 2 LIMIT 1</code>	3.77 ms
Category	<code>SELECT * FROM `categories` WHERE `categories`.`id` = 3 LIMIT 1</code>	3.45 ms

The tool also shows a summary of 4 queries, 4 selects, and a total time of 66 ms. The right sidebar shows the "GET / Closure" route with a status of 200 and a response size of 200 bytes. The bottom right corner displays the date and time: 2023-10-19 17:50:34.

Entonces para resolver esto vamos a editar la ruta en el archivo `web.php`

```
Route::get('/', function () {  
    return view('posts', [  
        'posts' => Post::with('category')->get()  
    ]);  
});
```

Ahora visualizamos la web y veremos que solo se cargan dos queries pero la pagina sigue funcionando de la misma manera



La siembra de bases de datos ahorra tiempo / Database Seeding Saves Time

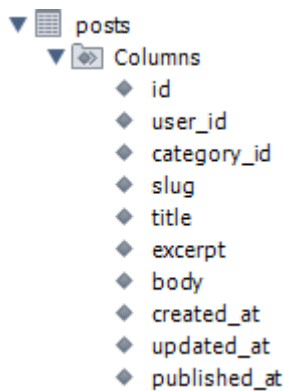
Vamos a crear una coneccion entre los post y el usuario que los crea, por lo que en la migracion de posts vamos a crear otro foreign key

```
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->foreignId('user_id');
        $table->foreignId('category_id');
        $table->string('slug')->unique();
        $table->string('title');
        $table->text('excerpt');
        $table->text('body');
        $table->timestamps();
        $table->timestamp('published_at')->nullable();
    });
}
```

Y ahora refrescamos la base de datos

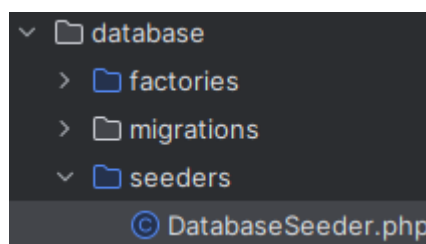
```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ php artisan migrate:fresh
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (95.55ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (47.80ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (45.86ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated: 2019_12_14_000001_create_personal_access_tokens_table (70.23ms)
Migrating: 2023_10_19_074650_create_posts_table
Migrated: 2023_10_19_074650_create_posts_table (54.96ms)
Migrating: 2023_10_19_220940_create_categories_table
Migrated: 2023_10_19_220940_create_categories_table (17.78ms)
```

Y ahora vemos como dentro de la base de datos en la tabla posts tenemos la columna de user_id



Cada vez que refrescamos la base de datos con el comando migration:fresh perdemos toda la informacion de prueba que introducimos, por lo que para arreglar esto haremos lo siguiente

Nos dirigimos al archivo `DatabaseSeeder.php`



Dentro del archivo creamos lo siguiente

```
public function run()
{
    \App\Models\User::factory(10)->create();
}
```

Luego nos movemos a la terminal de la Vm webserver y corremos el siguiente comando

```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ php artisan db:seed
Database seeding completed successfully.
```

Este comando junto con el codigo anterior lo que hace es crearnos 10 usuarios como informacion de prueba dentro de la base de datos.

id	name	email	email_verified_at	password	remember_token
1	Hellen Kemmer	ullrich.brigitte@example.org	2023-10-20 00:09:36	\$2y\$10\$92IXUNpkj00rOQ5byMi.Ye4oKoEa3Ro...	Ubtv5...
2	Kaia VonRueden	brielle.zulauf@example.net	2023-10-20 00:09:36	\$2y\$10\$92IXUNpkj00rOQ5byMi.Ye4oKoEa3Ro...	gnJRC...
3	Mr. Ken Donnelly PhD	meaghan.fahey@example.com	2023-10-20 00:09:36	\$2y\$10\$92IXUNpkj00rOQ5byMi.Ye4oKoEa3Ro...	Iv3Jj5...
4	Ms. Jennyfer Hoeger III	dwight.lindgren@example.org	2023-10-20 00:09:36	\$2y\$10\$92IXUNpkj00rOQ5byMi.Ye4oKoEa3Ro...	Y5coC...
5	Dr. Helmer Simonis	haskell.rempel@example.org	2023-10-20 00:09:36	\$2y\$10\$92IXUNpkj00rOQ5byMi.Ye4oKoEa3Ro...	259m2...
6	Waylon Crona	buddy.robel@example.com	2023-10-20 00:09:36	\$2y\$10\$92IXUNpkj00rOQ5byMi.Ye4oKoEa3Ro...	kuon6...
7	Prof. Conor Hintz MD	mante.tomasa@example.com	2023-10-20 00:09:36	\$2y\$10\$92IXUNpkj00rOQ5byMi.Ye4oKoEa3Ro...	n8nnv...
8	Macie Wiegand DVM	emerson86@example.org	2023-10-20 00:09:36	\$2y\$10\$92IXUNpkj00rOQ5byMi.Ye4oKoEa3Ro...	2Cr17C...
9	Angelica Rolfson	ellen.schinner@example.com	2023-10-20 00:09:36	\$2y\$10\$92IXUNpkj00rOQ5byMi.Ye4oKoEa3Ro...	qTnYR...
10	Miss Onie Hansen	mya.reynolds@example.com	2023-10-20 00:09:36	\$2y\$10\$92IXUNpkj00rOQ5byMi.Ye4oKoEa3Ro...	Lz819f...

Ahora vamos a cambiar en la migracion de categories para que el nombre y el slug sean unicos

```
public function up()
{
    Schema::create('categories', function (Blueprint $table) {
        $table->id();
        $table->string('name')->unique();
        $table->string('slug')->unique();
        $table->timestamps();
    });
}
```

Ahora vamos a hacer cambios en el DataSeeder para que siembre información e ignore información que no pueda repetir

```
public function run()
{
    User::truncate();
    Category::truncate();
    Post::truncate();

    $user = User::factory()->create();

    $personal = Category::create([
        'name' => 'Personal',
        'slug' => 'personal',
    ]);

    $family = Category::create([
        'name' => 'Family',
        'slug' => 'family',
    ]);

    $work = Category::create([
        'name' => 'Work',
        'slug' => 'work',
    ]);
}
```



```

    });

    Post::create([
        'user_id' => $user->id,
        'category_id' => $family->id,
        'title' => 'My Family Post',
        'slug' => 'my-family-post',
        'excerpt' => 'Lorem ipsum dolor sit amet.',
        'body' => 'Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.'
    ]);

    Post::create([
        'user_id' => $user->id,
        'category_id' => $work->id,
        'title' => 'My Work Post',
        'slug' => 'my-work-post',
        'excerpt' => 'Lorem ipsum dolor sit amet.',
        'body' => 'Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.'
    ]);
}

```

Con este código ahora creamos un solo usuario, 3 categorías y 3 posts, y adaptamos el código para que si alguna categoría o usuario ya existe este la ignore

id	name	email	email_verified_at	password	remember_token
1	Mr. Andre Hirthe	wcronin@example.com	2023-10-20 00:22:54	\$2y\$10\$92IXUNpkjO0rOQ5byMi.Ye4oKoEa3Ro...	GsvzifqtNA

id	name	slug	created_at	updated_at
1	Personal	personal	2023-10-20 00:22:54	2023-10-20 00:22:54
2	Family	family	2023-10-20 00:22:54	2023-10-20 00:22:54
3	Work	work	2023-10-20 00:22:54	2023-10-20 00:22:54

id	user_id	category_id	slug	title	excerpt	body
1	1	2	my-family-post	My Family Post	Lorem ipsum dolor sit amet.	Lorem ipsum dolor sit amet, consectetur adipisci...
2	1	3	my-work-post	My Work Post	Lorem ipsum dolor sit amet.	Lorem ipsum dolor sit amet, consectetur adipisci...

Ahora cada vez que refresquemos la base de datos sembraremos información de prueba con el comando

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

Ahora creamos las relaciones elocuentes entre post y user

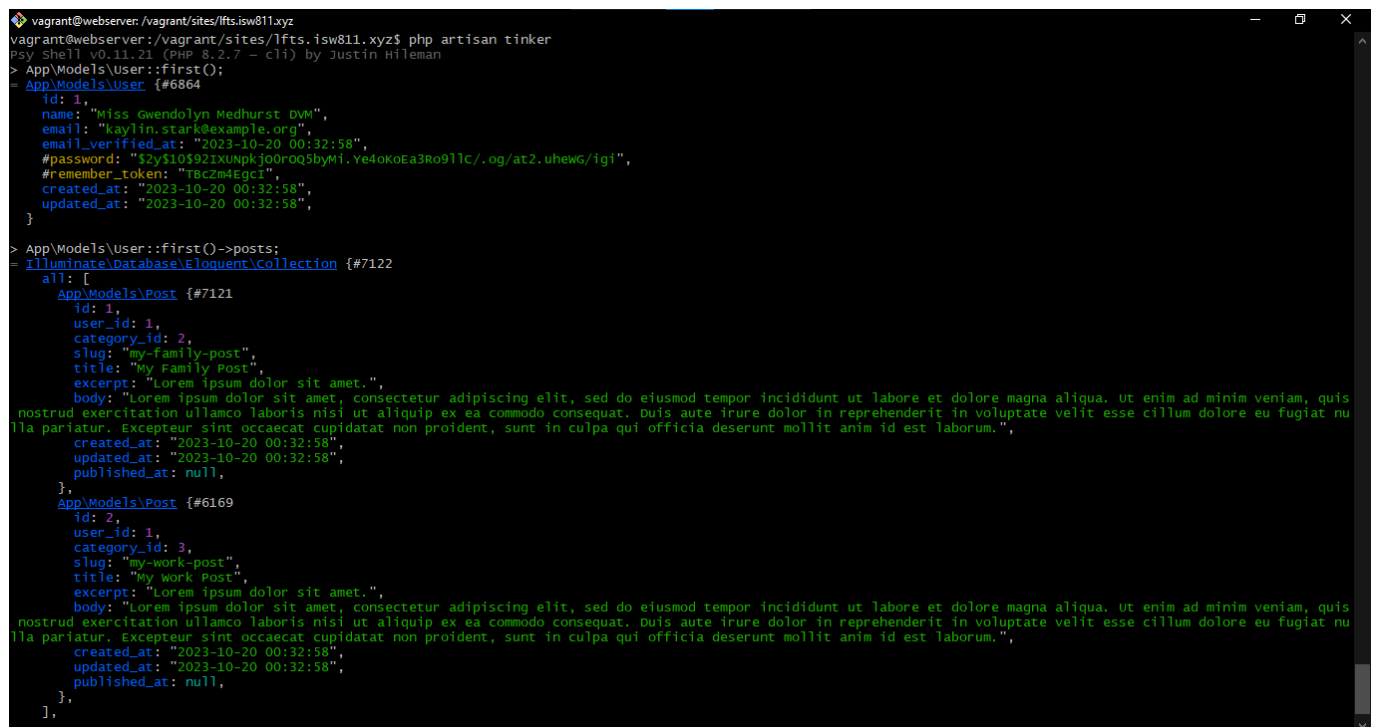
Añadimos esta funcion a la clase Post

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

Y en la clase User añadimos esta funcion

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

Ahora dentro de la terminal de nuestra VM webserver podremos buscar un usuario y acceder a los posts de este



```
vagrant@webserver: /vagrant/sites/lfts.isw811.xyz
vagrant@webserver: /vagrant/sites/lfts.isw811.xyz$ php artisan tinker
Psy Shell v0.11.21 (PHP 8.2.7 - cli) by Justin Hileman
> App\Models\User::first();
= App\Models\User {#6864
  id: 1,
  name: "Miss Gwendolyn Medhurst DVM",
  email: "kaylin.stark@example.org",
  email_verified_at: "2023-10-20 00:32:58",
  #password: "$2y$10$92IXunpkj00r0Q3byM1.Ye4oKoEa3Ro9llC/.og/at2.uhewG/igi",
  #remember_token: "tbcZm4EgcI",
  created_at: "2023-10-20 00:32:58",
  updated_at: "2023-10-20 00:32:58",
}
> App\Models\User::first()->posts;
= Illuminate\Database\Eloquent\Collection {#7122
  all: [
    App\Models\Post {#7121
      id: 1,
      user_id: 1,
      category_id: 2,
      slug: "my-family-post",
      title: "My Family Post",
      excerpt: "Lorem ipsum dolor sit amet.",
      body: "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.",
      created_at: "2023-10-20 00:32:58",
      updated_at: "2023-10-20 00:32:58",
      published_at: null,
    },
    App\Models\Post {#6169
      id: 2,
      user_id: 1,
      category_id: 3,
      slug: "my-work-post",
      title: "My work Post",
      excerpt: "Lorem ipsum dolor sit amet.",
      body: "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.",
      created_at: "2023-10-20 00:32:58",
      updated_at: "2023-10-20 00:32:58",
      published_at: null,
    },
  ],
}
```

Ahora haremos que en nuestro post se muestre el usuario dueño del post dinamicamente

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

@section('content')
<article>

<h1> {!! $post->title !!} </h1>

<p>
<a href="#">{{$post->user->name}}</a>
in
<a href="/categories/{{ $post->category->slug }}"> {{ $post->category->
```

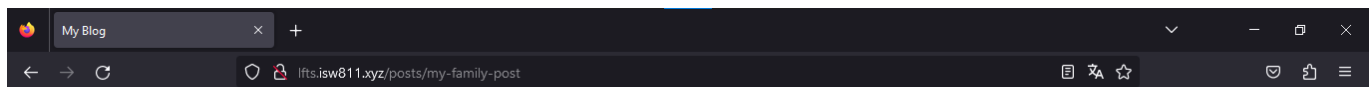
```
>name }} </a>
    </p>

    <div>
        {!! $post->body !!}
    </div>

</article>

<a href="/">Volver</a>
@endsection
```

Y vemos como nuestro post ahora tiene el nombre de su usuario



My Family Post

[Miss Gwendolyn Medhurst DVM in Family](#)

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

[Volver](#)

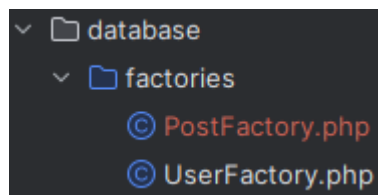
Turbo Boost con fábricas / Turbo Boost With Factories

Ahora vamos a crear un PostFactory para poder crear posts con sus respectivos atributos aleatoriamente dentro del DataSeeder

Nos vamos a nuestra terminal y corremos el comando

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

Esto nos creará este archivo



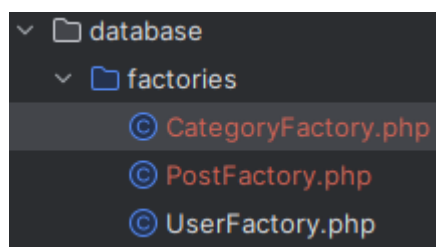
Dentro del folder **PostFactory**

Vamos a añadir el siguiente código

```
public function definition()
{
    return [
        'user_id' => User::factory(),
        'category_id' => Category::factory(),
        'title' => $this->faker->sentence,
        'slug' => $this->faker->slug,
        'excerpt' => $this->faker->sentence,
        'body' => $this->faker->sentence,
    ];
}
```

Hacemos el mismo proceso para factory

```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ php artisan make:factory CategoryFactory
Factory created successfully.
```



```
public function definition()
{
    return [
        'name' => $this->faker->word,
        'slug' => $this->faker->slug()
    ];
}
```

Ahora probamos a crear un post utilizando factory()

```
vagrant@webserver:/vagrant/sites/lfts.isw811.xyz$ php artisan tinker
Psy Shell v0.11.21 (PHP 8.2.7 - cli) by Justin Hileman
> App\Models\Post::factory()->create();
= App\Models\Post {#6225
  user_id: 1,
  category_id: 1,
  title: "Doloremque exercitationem ipsum a officia alias.",
  slug: "mollitia-architecto-omnis-mollitia-voluptatibus-repellendus-architecto-ipsam",
  excerpt: "Vel totam voluptatem iusto voluptas harum fugiat.",
  body: "Qui sint molestiae minima eos laboriosam.",
  updated_at: "2023-10-20 01:09:08",
  created_at: "2023-10-20 01:09:08",
  id: 1,
}
```

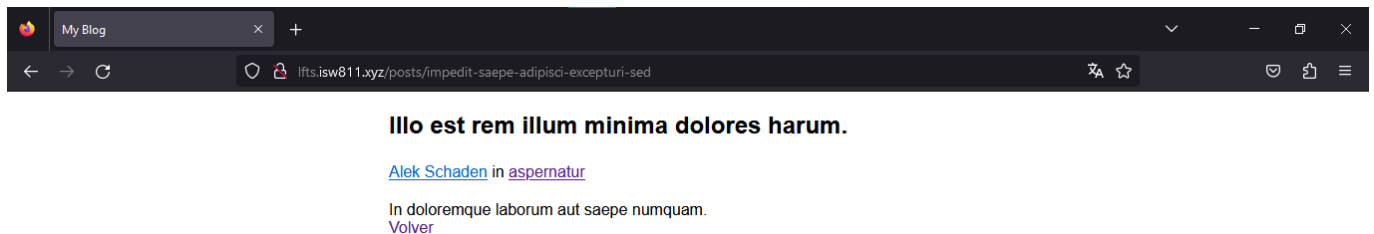
Como podemos ver funciona a la perfeccion

Ahora vamos a limpiar el codigo en el DataSeeder para ahora utilizar factory()

```
public function run()
{
    User::truncate();
    Category::truncate();
    Post::truncate();

    Post::factory()->create();
}
```

Ahora cada vez que creamos un post este va a crear un categoría generica y un usuario generico



Ver todos los post de un autor / View All Posts by An Author

Cada vez que un post es creado este se va al final de la pagina, para corregir esto y tener los posts mas recientes primero, hacemos esta pequeña modificación en la ruta en `web.php`

```
Route::get('/', function () {
    return view('posts', [
        'posts' => Post::latest()->with('category')->get()
    ]);
});
```

Ahora cada post nuevo irá al principio de la pagina y no al final.

Vamos a cambiar el codigo en `post.blade.php` para que el usuario no sea referenciado como tal si no como author entonces agregamos

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

@section('content')
    <article>

        <h1> {!! $post->title !!} </h1>

        <p>
            By
            <a href="#">{{$post->author->name}}</a>
            in
            <a href="/categories/{{ $post->category->slug }}"> {{ $post->category-
>name }} </a>
        </p>

        <div>
            {!! $post->body !!}
        </div>

    </article>

    <a href="/">Volver</a>
@endsection
```

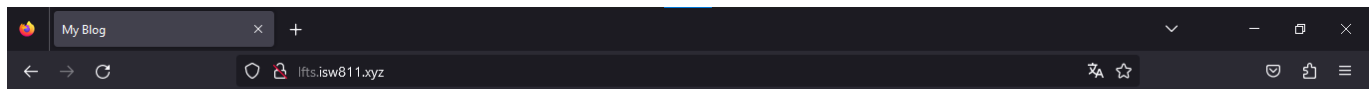
Y ahora nos dirigimos a la clase Post y cambiamos el codigo

```
public function author() {
    return $this->belongsTo(User::class, 'user_id');
}
```

Ahora vamos a agregar que se pueda observar el autor del post en la pagina principal, por lo que unicamente vamos a copiar este trozo de codigo de `post.blade.php` a `posts.blade.php`

```
<p>
  By
  <a href="#">{{ $post->author->name }}</a>
  in
  <a href="/categories/{{ $post->category->slug }}"> {{ $post->category->name }}
</a>
</p>
```

Ahora visualizamos la pagina principal



[Illo est rem illum minima dolores harum.](#)

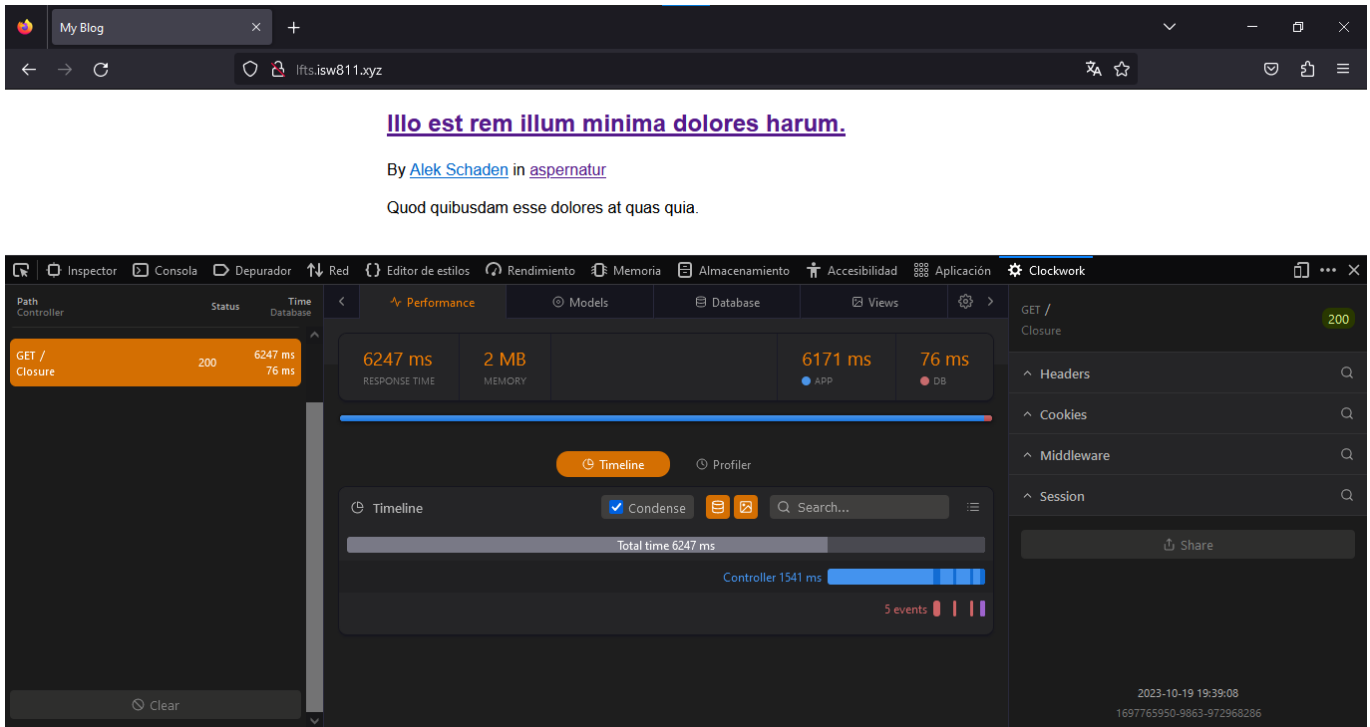
By [Alek Schaden](#) in [aspernatur](#)

Quod quibusdam esse dolores at quas quia.

El problema es que por cada post que tenemos que hace un select para llamar al autor y eso ralentiza la pagina, por lo que lo solucionamos con lo siguiente en web.php

```
Route::get('/', function () {
    return view('posts', [
        'posts' => Post::latest()->with(['category', 'author'])->get()
    ]);
});
```

Y podemos visualizar que solo se utilizan 3 queries al momento de cargar la pagina principal



Vamos a agregar una nueva columna en la migracion de user para poder tener username

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

Luego modificamos el UserFactory para que al crear usuarios se cree el username

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

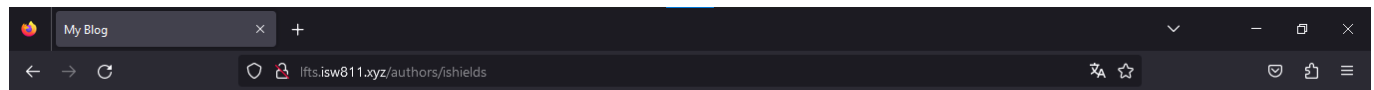

Ahora vamos a agregar una nueva ruta en web.php para poder acceder a los posts pertenecientes al mismo autor por medio del username

```
Route::get('authors/{author:username}', function (User $author) {  
  
    return view('posts', [  
        'posts' => $author->posts  
    ]);  
});
```

Y modificamos el código en posts y post para que al darle click al autor este nos redirija a la página donde se encuentran todos sus posts

```
@extends('layouts.layout')  
  
@section('content')  
    <article>  
  
        <h1> {!! $post->title !!} </h1>  
  
        <p>  
            By  
            <a href="/authors/{{ $post->author->username }}">{{ $post->author->name }}  
        </a>  
            in  
            <a href="/categories/{{ $post->category->slug }}">{{ $post->category->name }} </a>  
        </p>  
  
        <div>  
            {!! $post->body !!}  
        </div>  
  
    </article>  
  
    <a href="/">Volver</a>  
@endsection
```

Ahora podemos visualizar los posts de un mismo autor por medio de su username en la URL



Relaciones de carga ansiosas en un modelo existente / Eager Load Relationships on an Existing Model

Vamos a crear 10 posts con el id de categoría 1

```
> App\Models\Post::Factory(10)->create(['category_id' => 1]);
= Illuminate\Database\Eloquent\Collection {#6203}
  all: [
    App\Models\Post {#6229}
      user_id: 2,
      category_id: 1,
      title: "Similique qui velit maxime in et sed.",
      slug: "ut-et-error-esse-cumque-exercitationem-sunt-impedit",
      excerpt: "Et sequi quia et iusto aut id eaque.",
      body: "Accusantium architecto aliquam expedita aliquam ex officiis aliquid.",
      updated_at: "2023-10-20 02:05:53",
      created_at: "2023-10-20 02:05:53",
      id: 2,
    },
    App\Models\Post {#6565}
      user_id: 3,
      category_id: 1,
      title: "Molestias qui repellendus qui quos delectus illo ex et.",
      slug: "voluptatem-animi-nihil-molestiae-temporibus-expedita-nisi",
      excerpt: "Esse non aut pariat neque id quos deleniti.",
      body: "Minima aut quidem ea quo aut non et.",
      updated_at: "2023-10-20 02:05:53",
      created_at: "2023-10-20 02:05:53",
      id: 3,
    },
    App\Models\Post {#6972}
      user_id: 4,
      category_id: 1,
      title: "Est deleniti possimus placeat ipsa consequuntur.",
      slug: "non-omnis-blanditiis-laborum-at-voluptatem-nihil",
      excerpt: "Assumenda vitae blanditiis voluptas iusto ullam officia.",
      body: "Quis sint dolore et quis tenetur quam.",
      updated_at: "2023-10-20 02:05:53",
      created_at: "2023-10-20 02:05:53",
      id: 4,
    },
    App\Models\Post {#6230}
      user_id: 5,
      category_id: 1,
      title: "Deleniti est cupiditate eligendi nulla fuga cumque.",
```

Ahora vamos a visualizar el clockwork al cargar estos posts en la categoría

The screenshot shows a web browser displaying a blog post titled "Esse rerum officiis odio." by Wallace Boehm Jr. Below the browser, a development tool (likely Chrome DevTools) is open, showing the "Database" tab. The "Queries" panel lists several SQL queries executed during the page load:

Model	Query	Duration
Category	<code>SELECT * FROM `categories` WHERE `slug` = 'illum-eius-voluptatem-nihil-aperiam' LIMIT 1</code>	55.45 ms
Post	<code>SELECT * FROM `posts` WHERE `posts`.`category_id` = 1 and `posts`.`category_id` IS not NULL</code>	4.49 ms
User	<code>SELECT * FROM `users` WHERE `users`.`id` = 1 LIMIT 1</code>	5.5 ms
Category	<code>SELECT * FROM `categories` WHERE `categories`.`id` = 1 LIMIT 1</code>	3.11 ms
User	<code>SELECT * FROM `users` WHERE `users`.`id` = 2 LIMIT 1</code>	3.35 ms

The "Performance" tab shows a total of 24 queries, 24 selects, and a total time of 136 ms. The "Path Controller" tab shows a GET request to `/categories/illum-eius-voluptatem-nihil-aperiam` with a status of 200 and a response time of 6444 ms.

Podemos ver que de nuevo hay muchos querys corriendose al mismo tiempo, al cargar todos los posts de un mismo autor pasa lo mismo, por lo que modificaremos las rutas en `web.php`

```
Route::get('categories/{category:slug}', function (Category $category) {
    return view('posts', [
        'posts' => $category->posts->load(['category', 'author'])
    ]);
});

Route::get('authors/{author:username}', function (User $author) {
    return view('posts', [
        'posts' => $author->posts->load(['category', 'author'])
    ]);
});
```

Ahora vemos como se corren muchos menos querys y la pagina tiene mejor respuesta

The screenshot shows a web browser displaying a blog post titled "Esse rerum officiis odio." by Wallace Boehm Jr. Below the title is a Latin quote: "Laudantium ut reiciendis doloribus itaque laboriosam occaecati impedit." The browser's address bar shows the URL "lfts.isw811.xyz/categories/illum-eius-voluptatem-nihil-aperiam".

Below the browser, a development tool (likely Laravel Telescope) displays the database queries executed for the request. The tool shows a summary of 4 queries, 4 selects, and a total time of 63 ms. The queries are listed in a table:

Model	Query	Duration
Category	SELECT * FROM `categories` WHERE `slug` = 'illum-eius-voluptatem-nihil-aperiam' LIMIT 1	50.86 ms
Post	SELECT * FROM `posts` WHERE `posts`.`category_id` = 1 and `posts`.`category_id` IS not NULL	4.98 ms
Category	SELECT * FROM `categories` WHERE `categories`.`id` in (1)	3.23 ms
User	SELECT * FROM `users` WHERE `users`.`id` in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)	4.22 ms

Ahora vamos a ver una alternativa a este metodo que acabamos de utilizar

Nos dirigimos a la clase Post y añadimos lo siguiente

```
protected $with = ['category', 'author'];
```

Por lo que ahora no sería necesario el cambio que hicimos en web.php por lo que estas funciones quedarían así

```
Route::get('/', function () {
    return view('posts', [
        'posts' => Post::latest()->get()
    ]);
});

Route::get('categories/{category:slug}', function (Category $category) {
    return view('posts', [
        'posts' => $category->posts
    ]);
});

Route::get('authors/{author:username}', function (User $author) {
    return view('posts', [
        'posts' => $author->posts
    ]);
});
```

Y vemos como igualmente se utilizan pocos queries

My Blog

lfts.isw811.xyz/categories/illum-eius-voluptatem-nihil-aperiam

Esse rerum officiis odio.

By [Wallace Boehm Jr.](#) in [ab](#)

Laudantium ut reiciendis doloribus itaque laboriosam occaecati impedit.

moz-extension://a831ef1c-bb09-49f9-974c-24db927b3df4/index.html#

Inspector

Consola

Depurador

Red

Editor de estilos

Rendimiento

Memoria

Almacenamiento

Accesibilidad

Aplicación

Clockwork

Path Controller

Status

Time Database

GET /categories/illum-eius-vc Closure

200

6444 ms

136 ms

GET /categories/illum-eius-vc Closure

200

6184 ms

63 ms

GET /categories/illum-eius-vc Closure

200

6289 ms

64 ms

Clear

Performance

Models

Database

Views

4

4

64 ms

QUERIES

SELECTS

TIME

Queries

Prettify

Search...

Model

Query

Duration

Category

SELECT * FROM `categories` WHERE `slug` = 'illum-eius-voluptatem-nihil-aperiam' LIMIT 1

52.77 ms

Post

SELECT * FROM `posts` WHERE `posts`.`category_id` = 1 and `posts`.`category_id` IS not NULL

3.94 ms

Category

SELECT * FROM `categories` WHERE `categories`.`id` in (1)

3.86 ms

User

SELECT * FROM `users` WHERE `users`.`id` in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)

3.69 ms

GET /categories/illum-eius-voluptatem-nihil-aperiam Closure

200

Headers

Cookies

Middleware

Session

Share

2023-10-19 20:18:05

1697768288-4769-645274942