

Curso de Laravel 10

Contenido

[Introducción 3](#_Toc157505271)

[Comando para iniciar un proyecto 3](#_Toc157505272)

[Extensiones VSCode 3](#_Toc157505273)

[Rutas en Laravel ¿Qué son y para que sirven? 3](#_Toc157505274)

[Controladores en Laravel ¿Qué son y para que sirven? 5](#_Toc157505275)

[Grupo de rutas 7](#_Toc157505276)

[Vistas en Laravel 7](#_Toc157505277)

[Sistema de Plantillas Blade 9](#_Toc157505278)

[Introducción al manejo de base de datos con Laravel 11](#_Toc157505279)

[Introducción a las migraciones 11](#_Toc157505280)

[php artisan migration 12](#_Toc157505281)

[Cómo crear migraciones en Laravel 12](#_Toc157505282)

[Como modificar tablas con las migraciones de Laravel 13](#_Toc157505283)

[Que pasa si ahora quiero cambiar la estructura de las columnas 15](#_Toc157505284)

[Introducción al manejo de bbdd con eloquent 16](#_Toc157505285)

[Seeders en Laravel ¿Qué son y para que sirven? 18](#_Toc157505286)

[Factories en Laravel ¿Qué son y para que sirven? 18](#_Toc157505287)

[Generador de consultas de eloquent 20](#_Toc157505288)

[Crud 29](#_Toc157505289)

[Cómo listar y leer registros en Laravel 29](#_Toc157505290)

[Cómo agregar y actualizar registros desde un formulario 35](#_Toc157505291)

[Formularios 36](#_Toc157505292)

[Agregar 36](#_Toc157505293)

[Editar Registros 38](#_Toc157505294)

[Modelo Entidad Relación 41](#_Toc157505295)

# Introducción

## Comando para iniciar un proyecto

composer create-project laravel/laravel example-app

## Extensiones VSCode

* Laravel Blade formatter
* Laravel Blade Snippets
* Laravel Goto View
* Laravel Snippets
* PHP Intelephense
* Tailwind CSS IntelliSense
* Alpine.js IntelliSense
* GitHub Copilot

## Rutas en Laravel ¿Qué son y para que sirven?

Las rutas en Laravel son una herramienta fundamental para el desarrollo de aplicaciones web. En el primer capítulo del curso "Laravel desde cero", aprenderás qué son las rutas en Laravel y para qué sirven. Las rutas permiten definir las URLs de una aplicación y mapearlas a controladores que procesan las solicitudes del usuario. En este capítulo, conocerás cómo crear rutas básicas y avanzadas, definir parámetros en las rutas y cómo utilizar middleware para aplicar filtros y restricciones en las solicitudes.

Laravel trabaja con un único punto de entrada que esta en la carpeta public y es el archivo index y laravel usa rutas para ver las rutas o ejecutarlas trabajamos en la carpeta routes / web.php.

Cuando un usuario escriba una url esta se validara aquí.

Ejemplo: Por defaul tenemos esta ruta

Route::get('/', function () {

    return view('welcome');

});

Aquí se define que cuando el usuario entre a la ruta principal del proyecto se mande a la vista o archivo welcome

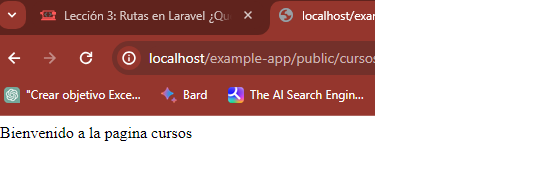
Las view son los archivos del front end que se muestran en nuestro proyecto por el momento no tengo personalizados pero puedo en lugar de una vista mandar un mensaje por ejemplo

Route::get('cursos', function () {

    return 'Bienvenido a la pagina cursos';

});

En esta parte de código determinamos que cuando el usuario Introduzca la url /cursos nos mande un texto que diga “Bienvenidos a la pagina de cursos”



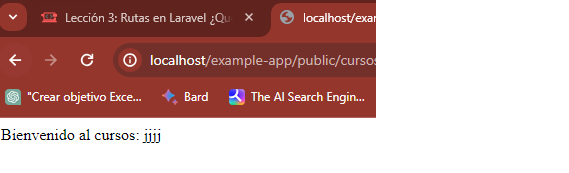
Cuando quieras mandar datos en la url ósea variables, para definir la ruta hacemos lo siguiente

A la ruta le integramos una variable y esta se declaro en la función por ultimo la concatenamos en el return.

Route::get('cursos/{curso}', function ($curso) {

    return "Bienvenido al cursos: $curso";

});



Nota importante las rutas se leen de forma descendente por lo que es importante tenerlo en cuenta por si nuestra referencia no funciona bien.

Ejemplo con valores alternativos ósea que puedan o no tener parámetros y esto se ocupa mucho cuando mostramos parte a parte el proyecto puede ser que tengamos el caso donde mostremos el curso pero no la categoría o que mostremos el curso + la categoría.

Nota por atención a los detalles las variables se ponen entre {} para que fucionen las opcionales se les agrega ?

Route::get('cursos/{curso}/{categoria?}', function($curso, $categoria = null){

    if($categoria){

        return "Bienvenidos al cruso: $curso, de la categoria: $categoria";

    }

    else{

        return "Bienvenido al curso: $curso";

    }

});

## Controladores en Laravel ¿Qué son y para que sirven?

Los controladores son clases que manejan la lógica de la aplicación y se encargan de procesar las solicitudes del usuario. En este capítulo, aprenderás a crear y utilizar controladores en Laravel, así como a definir métodos y rutas para acceder a ellos. Los controladores son una parte fundamental de la arquitectura MVC de Laravel, lo que te permitirá desarrollar aplicaciones de manera estructurada y escalable.

Cuando un usuario hace una petición por una url este se le asigna a un controlador esto ve que necesita y le da la orden a un modelo que le da ciertos datos a la vista y la vista se genera para que el usuario vea en pantalla lo que requiere.

Los controladores se crean en app/Http/Controllers

Para hacerlo lo hacemos desde la terminal ejecutamos el comando:

php artisan make:controller HomeController

ya que lo tenemos creado debemos asignar una ruta para ello en nuestra carpeta routes ponemos el siguiente código:

En la cabecera primero lo importo

use App\Http\Controllers\HomeController;

Después voy a editar la ruta que tenia en welcome

Route::get('/', HomeController::class);

Lo que hice fue quitar la función y llamar al controlador, ahora el contenido de la función lo vamos a programar en el controlador

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller

{

    public function \_\_invoke()

    {

        return "Bienvenido a la pagina principal";

    }

}

Ahora crearemos un controlador para los curso

Nota en el ejemplo anterior usamos \_\_invoke() este método se ocupa cuando gestionamos una sola ruta cuando es mas de una debemos definir sus nombre por ejemplo

class CursoController extends Controller

{

    public function index(){

    }

    public function create(){

    }

    public function show(){

    }

}

Ahora para llamarlo ponemos

Route::get('/', HomeController::class);

Route::get('cursos', [CursoController::class, 'index']);

Route::get('cursos/create', [CursoController::class, 'create']);

Route::get('cursos/{curso}', [CursoController::class, 'show']);

Y en cada función programamos

class CursoController extends Controller

{

    public function index(){

        return 'Bienvenido a la pagina cursos';

    }

    public function create(){

        return "Bienvenido desde este sitio puedes crear un curso";

    }

    public function show($curso){

        return "Bienvenido al curso: $curso";

    }

}

Es simple creamos el controlador lo colocamos en la ruta e invocamos el mensaje con o sin parámetros.

## Grupo de rutas

Los grupos de rutas te permiten agrupar un conjunto de rutas relacionadas y aplicarles filtros y middleware de manera conjunta. De esta forma, puedes definir de manera eficiente rutas con características en común como autenticación, roles de usuario o idiomas. En este capítulo, conocerás cómo crear grupos de rutas, definir middleware para aplicar filtros y restricciones, y cómo utilizar el método "prefix" para establecer prefijos comunes en las rutas de un grupo.

Si queremos limpiar aun mas nuestro código podemos usar grupos esto simplifica mas las rutas que se deriven de una ruta principal como en nuestro ejemplo vamos a agrupar las sub rutas que se derivan de cursos.

Route::controller(CursoController::class)->group(function(){

    Route::get('cursos', 'index');

    Route::get('cursos/create', 'create');

    Route::get('cursos/{course}', 'show');

});

## Vistas en Laravel

Las vistas en Laravel son archivos que contienen código HTML, CSS y JavaScript y se utilizan para presentar la información al usuario. En este capítulo, aprenderás a crear vistas en Laravel y a utilizar el motor de plantillas Blade para generar HTML de manera eficiente. Además, conocerás cómo pasar datos a las vistas desde los controladores y cómo utilizar directivas y componentes de Blade para mejorar la legibilidad y mantenibilidad del código. Las vistas son una parte fundamental del patrón MVC de Laravel, lo que te permitirá separar la lógica de la presentación de tu aplicación y desarrollarla de manera más eficiente.

Ahora crearemos una vista lo cual nos permitirá visualizar un código de html en una ruta, las vistas están en resources/view ahora para crear una vista requerimos de una serie de archivos.

Paso 1 crear los archivos en la carpeta view por ejemplo el home.php

Paso 2 edito el controller con el método view

class HomeController extends Controller

{

    public function \_\_invoke()

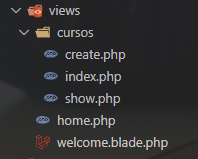
    {

        return view('home');

    }

}

Lo mismo hacemos para los cursos solo que aquí agregamos una carpeta cursos y dentro cada una de las vistas o archivos que llamamos en el controller esto queda así



El código queda así

Index.php

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

<h1>Pagina de cursos</h1>

</body>

</html>

Créate.php

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

<h1>Pagina para crear cursos</h1>

</body>

</html>

Show.php

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

<h1>Pagina del curso: <?php echo $curso;?></h1>

</body>

</html>

El archivo CursoController

class CursoController extends Controller

{

    public function index(){

        return view('cursos.index');

    }

    public function create(){

        return view('cursos.create');

    }

    public function show($curso){

        return view('cursos.show', ['curso'=> $curso]);

    }

}

## Sistema de Plantillas Blade

El sistema de plantillas nos permite crear un estilo general para todas las vistas de nuestro sitio es muy útil si queremos optimizar los cambios generales del proyecto.

Proceso:

Dentro de views creamos una carpeta llamada layouts y dentro nuestro archivo que llamaremos plantilla.blade.php es indispensable que le pongamos .blade.php para que esta funcione el código será el siguiente:

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>@yield('title')</title>

  <link rel="stylesheet" href="style.css">

  <!--favicon-->

  <!--estilos-->

</head>

<body>

    <!--header-->

    <!--nav-->

    @yield('content')

    <!--footer-->

    <!--script-->

</body>

</html>

El componente principal el @yield dentro de el especificamos la etiqueta en este caso ‘title’ y ‘content’

El siguiente paso es editar los archivos de las vistas renombrados con .blade.php por ejemplo [home.blend.php](http://home.blend.php) y editamos nuestro código:

@extends('layouts.plantilla')

@section('title', 'Home')

@section('content')

    <h1>Bienvenidos a la pagina principal</h1>

@endsection()

Hacemos la importación de la plantilla con @extends

Y usamos @section para identificar las secciones donde escribimos ahora la de content tiene @endsection por si colocamos más de una línea de código o no solo un parámetro.

Ahora repetimos el paso para todos los archivos de nuestra vista

Cambiamos el nombre del archivo con .blade.php, Importamos con @extends y colocamos los @sections y @endsections si es necesario los códigos quedan así

Index

@extends('layouts.plantilla')

@section('title', 'Cursos')

@section('content')

    <h1>Bienvenidos a la pagina de Cursos</h1>

@endsection()

Créate

@extends('layouts.plantilla')

@section('title', 'Cursos /  Create')

@section('content')

    <h1>Bienvenidos a la pagina de Cursos / Create</h1>

@endsection()

Show ojo ahora usamos {{}} para agregar la variable esto omite el <?php?>

@extends('layouts.plantilla')

@section('title', 'Cursos' . $curso)

@section('content')

    <h1>Bienvenidos a la pagina de Cursos / {{$curso}}</h1>

@endsection()

# Introducción al manejo de base de datos con Laravel

Laravel ofrece una capa de abstracción de la base de datos que simplifica el manejo de operaciones SQL y mejora la seguridad de las aplicaciones. En este capítulo, conocerás cómo crear y migrar tablas de bases de datos utilizando el sistema de migraciones de Laravel, cómo trabajar con modelos Eloquent para interactuar con la base de datos y cómo utilizar consultas para recuperar, insertar, actualizar y eliminar registros en la base de datos. Además, te enseñaremos cómo utilizar el sistema de semillas de Laravel para poblar la base de datos con datos de prueba y cómo configurar múltiples conexiones de base de datos en tu aplicación. Con estos conocimientos, podrás trabajar con bases de datos de manera efectiva y segura en tus aplicaciones Laravel.

## Introducción a las migraciones

Las migraciones te permiten crear, modificar y eliminar tablas y campos de la base de datos de manera programática y controlada, lo que mejora la calidad y la mantenibilidad de la base de datos. En este capítulo, aprenderás cómo crear y ejecutar migraciones en Laravel, cómo revertir migraciones y cómo trabajar con el sistema de migraciones para administrar cambios en la estructura de la base de datos. Además, conocerás cómo utilizar las migraciones para crear tablas con restricciones de integridad referencial, índices y claves únicas. Con estos conocimientos, podrás manejar de manera efectiva la estructura de la base de datos en tus aplicaciones Laravel.

Las migraciones son la base de datos en laravel se encuentran en database / migration y se ejecutan con el Comando para las migraciones

## php artisan migration

## Cómo crear migraciones en Laravel

Las migraciones son una forma de manejar los cambios en la estructura de la base de datos de manera programática y controlada. En Laravel, las migraciones se escriben en código PHP y se ejecutan utilizando la línea de comandos. En este capítulo, te enseñaremos cómo crear migraciones utilizando el comando "make:migration" de Laravel, cómo definir los campos de la tabla y cómo agregar restricciones de integridad referencial, índices y claves únicas. Además, conocerás cómo utilizar los comandos de migración para crear y revertir migraciones y cómo trabajar con el sistema de migraciones de Laravel para administrar los cambios en la estructura de la base de datos de tu aplicación. Con estos conocimientos, podrás crear migraciones de manera efectiva y controlada en tus aplicaciones Laravel.

Comando php artisan make:migration cursos

Entramos a nuestra migración el código queda así y tiene comentarios que son importantes

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration

{

    /\*\*

     \* Run the migrations.

     \*/

    public function up(): void

    {

        //Usamos la instancia Schema

        //la funcion create y colocamos 2 parametros uno el nombre de la tabla y dos la funcion

        Schema::create('cursos', function(Blueprint $table){

            $table->id(); //Columna id de tipo entero auto incrementable

            $table->string('name'); //Columna string de 255 caracteres

            $table->text('description'); //Columna string de mas de 255 caracteres

            $table->timestamps(); //Crea la columna create\_at y update\_at se almacena el log de los registros

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*/

    public function down(): void

    {

        Schema::dropIfExists('cursos');

    }

};

Ahora migramos nuevamente en la terminal para agregar la

Nota: En laravel para revertir cambios existen los lotes en la tabla migrations se usa un comando y esto es un rollback Comando: php artisan migrate:rollback

Nota: Si al crear la tabla yo ejecuto el nombre de la migración así en automático se me crea el formato de tabla: php artisan make:migrate créate\_cursos\_table

Nota: Como actualizar una migración Agregar por ejemplo un nuevo campo, para eso agregamos el campo pero ahora ejecutamos el comando php artisan migrate:fresh

Esto elimina las tablas con el método down pero las crea nuevamente.

## Como modificar tablas con las migraciones de Laravel

Las migraciones te permiten modificar la estructura de la base de datos de manera programática y controlada, lo que mejora la calidad y la mantenibilidad de la base de datos. En este capítulo, te enseñaremos cómo agregar y eliminar columnas, cómo modificar el tipo de datos y las restricciones de las columnas existentes, y cómo renombrar y eliminar tablas utilizando migraciones de Laravel. Además, conocerás cómo utilizar las migraciones para agregar y eliminar índices y claves foráneas, y cómo trabajar con las migraciones para realizar cambios en cascada en la base de datos. Con estos conocimientos, podrás modificar de manera efectiva la estructura de la base de datos de tus aplicaciones Laravel de manera controlada y segura.

El método Up y Down se ejecutan en el fresh y el refresh cual es la diferencia en apariencia es lo mismo pero no si nosotros configuramos bien el método down entonces programamos como queremos que se lleve acabo el rollback es decir como queremos que se restaure.

**Como modificar una tabla cuando ya tienes registros.**

Numero uno vamos a eliminar la columna avatar y realizaremos la migración con fresh

php artisan migrate:fresh

Ahora voy a agregar un registro en mi base de datos en la tabla user

AHORA PARA EL SIGUIENTE PASO HAY QUE CREAR UN ARCHIVO DE MIGRACIÓN ESTO ES IMPORTANTE SE CREAN ARCHIVOS PARA CADA CAMBIO QUE QUEREMOS

Para agregar un registro creo una nueva migración recuerda que todo se hace mediante archivos

Pongo: php artisan make:migration add\_avatar\_to\_users\_table

Ponemos el siguiente código en este código usamos para agregar el campo a la tabla nullable() este método indica al gestor que este campo puede aceptar valores null, en el down usamos dropColumn por si hacemos un rollback no se elimine mas que el campo que creamos

return new class extends Migration

{

    /\*\*

     \* Run the migrations.

     \*/

    public function up(): void

    {

        Schema::table('users', function (Blueprint $table) {

            //

           $table->string('avatar')->nullable();

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*/

    public function down(): void

    {

        Schema::table('users', function (Blueprint $table) {

            //

            $table->dropColumn('avatar');

        });

    }

};

Otra propuesta es indicar des pues de que campo quieres que se cree el campo para eso usamos after

return new class extends Migration

{

    /\*\*

     \* Run the migrations.

     \*/

    public function up(): void

    {

        Schema::table('users', function (Blueprint $table) {

            //

           $table->string('avatar')->nullable()->after('email');

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*/

    public function down(): void

    {

        Schema::table('users', function (Blueprint $table) {

            //

            $table->dropColumn('avatar');

        });

    }

};

Ahora vamos a ver que se modifica pero no se borra la información.

## Que pasa si ahora quiero cambiar la estructura de las columnas

Lo primercompoo es agregar una dependencia la composer.json con el comando

Composer require doctrine/dbal

Creamos una archivo de migración php artisan make:migration cambiar\_porpiedad\_to\_users\_table

Después el código quedaría

return new class extends Migration

{

    /\*\*

     \* Run the migrations.

     \*/

    public function up(): void

    {

        Schema::table('users', function (Blueprint $table) {

            //

            $table->string('name', 150)->change();

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*/

    public function down(): void

    {

        Schema::table('users', function (Blueprint $table) {

            //

            $table->string('name', 255)->change();

        });

    }

};

# Introducción al manejo de bbdd con eloquent

Eloquent es el ORM (Object-Relational Mapping) incluido en Laravel que simplifica el manejo de operaciones SQL y mejora la seguridad de las aplicaciones. En este capítulo, conocerás cómo crear modelos Eloquent y cómo utilizarlos para interactuar con la base de datos. Aprenderás cómo realizar consultas básicas y avanzadas utilizando el método query builder y cómo utilizar las relaciones Eloquent para acceder a los datos de varias tablas relacionadas. Además, te enseñaremos cómo utilizar el sistema de paginación y cómo realizar operaciones CRUD (crear, leer, actualizar y eliminar) en la base de datos utilizando Eloquent. Con estos conocimientos, podrás trabajar con bases de datos de manera efectiva y segura en tus aplicaciones Laravel utilizando Eloquent ORM.

Entramos a una parte muy importante que son los modelos estos me permiten administrar las tablas que aprendimos a crear anteriormente, seguramente has escuchado que las tablas deberían de tener sus nombre en ingles y esto es cierto por una razón cuando creamos un modelos generalmente lo hacemos apartir de tener ya una tabla laravel considera que los nombres que pones a los modelos son el singular de la relación que hay con las tablas, por ejemplo si tengo una migrate que se llama users su modelo se debería llamar User, si tengo una que se llama categories mi modelo seria Category ok

El comando que se ocupa es php artisan make:model User

Los modelos se crean en la carpeta app / models por defaul tenemos el de User por que hay una migration que se llama users

Ahora que pasa si los nombres están en español pues debemos poner nombres que terminen con s en las migrations y sin s en los modelos por ejemplo tengo migration cursos y mi modelos seria Curso.

Los modelos permiten tratar a los registros como objetos con una tecnología que es Tinker y este se hace ejecutando el comando en consola php artisan tinker ejemplo:

PS C:\laragon\www\example-app> php artisan tinker

Psy Shell v0.11.22 (PHP 8.1.10 — cli) by Justin Hileman

> use App\Models\Curso; //Primero seleciono la tabla o modelo a usar

> $curso = new Curso; //Variable curso instancia de Curso

= App\Models\Curso {#6296} //Resultado

> $curso->name = 'Laravel'; //mandamos a la variable curso la columna name y el dato

= "Laravel" //Salida

> $curso->description = 'El mejor framework de php'; //lo mismo pero en description

= "El mejor framework de php"

> $curso

= App\Models\Curso {#6296

name: "Laravel",

description: "El mejor framework de php",

}

> $curso->save(); //importante aquí guardamos los cambios directo a la bd

= true

> $curso

= App\Models\Curso {#6296

name: "Laravel",

description: "El mejor framework de php",

updated\_at: "2024-01-05 20:28:35",

created\_at: "2024-01-05 20:28:35",

id: 1,

}

>

Como vemos en esta captura de consola vemos que se agrega el registro es una interface del odbc desde la consola. Si quiero sobre escribir el registro solo coloco la asignación del dato al objeto y después el objeto con save.

## Seeders en Laravel ¿Qué son y para que sirven?

Los seeders te permiten definir datos de prueba en código y ejecutarlos de manera automatizada, lo que facilita el proceso de llenado de datos en la base de datos. En este capítulo, te enseñaremos cómo crear y ejecutar seeders en Laravel, cómo definir datos de prueba y cómo utilizar las relaciones Eloquent para crear datos de prueba relacionados. Además, conocerás cómo utilizar las clases Factory en Laravel para crear datos aleatorios y cómo utilizar el comando "db:seed" para ejecutar tus seeders. Con estos conocimientos, podrás llenar la base de datos con datos de prueba de manera fácil y automatizada en tus aplicaciones Laravel.

En resumen son un archivo que migramos para llenar los registros en nuestras tablas.

## Factories en Laravel ¿Qué son y para que sirven?

Las factories te permiten definir patrones para generar datos aleatorios y luego utilizarlos para llenar la base de datos con datos de prueba de manera automatizada. En este capítulo, te enseñaremos cómo crear y utilizar factories en Laravel, cómo definir los campos de las tablas y cómo utilizar las relaciones Eloquent para crear datos de prueba relacionados. Además, conocerás cómo utilizar las factories para generar datos de prueba en formato JSON, cómo utilizar los traits para crear datos aleatorios y cómo utilizar las factories para realizar pruebas automatizadas en Laravel. Con estos conocimientos, podrás generar datos de prueba de manera efectiva y automatizada en tus aplicaciones Laravel utilizando factories.

En resumen nos generan datos aleatorios que usamos en nuestros seeders

Ahora vamos a hacer una fusión entre los seeders y los factory con estos comandos

Voy a editar mi tabla cursos y le agregare una columna de catergorias

Primero creo el archivo Seeder

php artisan make:seeder CursoSeeder

php artisan make:factory CursoFactory –model=Curso

Codigo de Factory es

<?php

namespace Database\Factories;

use App\Models\Curso;

use Illuminate\Database\Eloquent\Factories\Factory;

/\*\*

 \* @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Cruso>

 \*/

class CursoFactory extends Factory

{

    /\*\*

     \* Define the model's default state.

     \*

     \* @return array<string, mixed>

     \*/

     protected $model = Curso::class;

    public function definition(): array

    {

        return [

            'name'=> $this->faker->sentence(),

            'description'=> $this->faker->paragraph(),

            'category'=> $this->faker->randomElement(['Desarrollo web', 'Diseño Web']),

        ];

    }

}

Ahora el código de los Seeders queda así con 50 registros

<?php

namespace Database\Seeders;

use App\Models\Curso;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;

use Illuminate\Database\Seeder;

class CursoSeeder extends Seeder

{

    /\*\*

     \* Run the database seeds.

     \*/

    public function run(): void

    {

        //

        Curso::factory(50)->create();

    }

}

El código para el archivo DatabaseSeeder es para que jale todos los seeder que creemos

Es muy importante poner call para cada seeder $this->call(CursoSeeder::class);

<?php

namespace Database\Seeders;

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

use App\Models\Curso;

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder

{

    /\*\*

     \* Seed the application's database.

     \*/

    public function run(): void

    {

        // \App\Models\User::factory(10)->create();

        // \App\Models\User::factory()->create([

        //     'name' => 'Test User',

        //     'email' => 'test@example.com',

        // ]);

        $this->call(CursoSeeder::class);

    }

}

Ahora los comando a ejecutar son

php artisan migrate:fresh –seed

## Generador de consultas de eloquent

El generador de consultas te permite construir consultas SQL de manera programática utilizando una sintaxis intuitiva y fácil de entender. En este capítulo, te enseñaremos cómo construir consultas básicas y avanzadas utilizando el generador de consultas de Eloquent, cómo utilizar los operadores lógicos y de comparación, cómo realizar consultas con cláusulas "where", "having" y "group by", y cómo utilizar las relaciones Eloquent en las consultas. Además, conocerás cómo utilizar el sistema de paginación de Laravel y cómo realizar operaciones CRUD utilizando el generador de consultas. Con estos conocimientos, podrás construir consultas SQL de manera efectiva y segura en tus aplicaciones Laravel utilizando el generador de consultas de Eloquent.

Usando Tinker

Mostrar todos los registros de la tabla

> use App\Models\Curso

> $cursos = Curso::all();

= Illuminate\Database\Eloquent\Collection {#7302

all: [

App\Models\Curso {#7304

id: 1,

name: "Deserunt autem est harum ut recusandae excepturi.",

description: "Et tempore quibusdam ipsa libero nihil sunt nemo. Impedit cupiditate optio illo occaecati aspernatur mollitia. Fuga est non ut magnam dolore. In rerum aliquid exercitationem consequatur adipisci.",

category: "Desarrollo web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7305

id: 2,

name: "Est aut omnis aliquid quis molestiae.",

description: "Natus nesciunt ut saepe laboriosam. Doloremque repellendus enim magnam nulla fuga sunt quibusdam. Atque voluptatem corporis ut nihil architecto. Voluptas molestiae itaque qui.",

category: "Desarrollo web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7306

id: 3,

name: "Eaque magnam laudantium adipisci atque sequi ducimus.",

description: "Illo ipsum autem ab voluptatibus odit veritatis maiores commodi. Sed quia quae quaerat corporis tempora quia. Est laborum quo omnis consequuntur cum libero rerum.",

category: "Desarrollo web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7307

id: 4,

name: "Qui minima aperiam numquam iusto animi magnam.",

description: "Libero porro ratione excepturi porro occaecati consequatur. Omnis qui officiis dicta. Illo et laudantium similique doloremque totam et.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7308

id: 5,

name: "Aut dolorem molestias fuga.",

description: "Sed omnis qui nemo debitis minima eum. Esse at non unde necessitatibus quam sunt quo. Id ut minus dolor rem. Soluta natus quia ut ex.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7350

id: 47,

name: "Et quisquam et aliquid.",

description: "Consequuntur modi optio dolores eum qui voluptas qui. Nisi quis quisquam aut dolore occaecati rerum. Velit et dolorem et ipsa consequatur sunt similique. Et quidem earum nihil impedit nulla voluptate.",

category: "Desarrollo web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7351

id: 48,

name: "Recusandae optio nesciunt molestias maiores qui rem.",

description: "Et quaerat explicabo commodi autem non veniam voluptate. Ea totam vel dolore ad. Facilis eos dolorum quia similique quaerat nobis. Aut culpa rerum est ducimus similique est suscipit qui.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7352

id: 49,

name: "Ipsum debitis et numquam accusamus.",

description: "Eius et consequuntur fuga et. Officia dolor rerum libero deserunt occaecati omnis. Sint eveniet temporibus maxime occaecati. Cum voluptatem natus sint sapiente provident quam. Eos aut est nihil nihil.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7353

id: 50,

name: "Error sed enim aut et.",

description: "Hic blanditiis voluptatem molestiae culpa. Doloremque omnis rerum deleniti nemo corporis nesciunt repudiandae. Consequatur laboriosam sit sit molestias est amet. Cupiditate totam et velit modi est blanditiis explicabo.",

category: "Desarrollo web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

],

}

>

Mostrando solo algunos registros mediante una consulta en este caso solo los que tengan una category de Diseño Web

$cursos = Curso::where('category','Diseño Web')->get();

= Illuminate\Database\Eloquent\Collection {#7320

all: [

App\Models\Curso {#7318

id: 4,

name: "Qui minima aperiam numquam iusto animi magnam.",

description: "Libero porro ratione excepturi porro occaecati consequatur. Omnis qui officiis dicta. Illo et laudantium similique doloremque totam et.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7317

id: 5,

name: "Aut dolorem molestias fuga.",

description: "Sed omnis qui nemo debitis minima eum. Esse at non unde necessitatibus quam sunt quo. Id ut minus dolor rem. Soluta natus quia ut ex.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7316

id: 7,

name: "Ullam et dolores rem adipisci.",

description: "Id inventore accusamus eius sapiente tempore quia voluptatem ut. Sequi suscipit similique omnis suscipit rerum. Dolorem maxime eos quo.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7315

id: 12,

name: "Dolorem maxime natus ut error.",

description: "Aspernatur amet deserunt et. Sed ut velit veritatis.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7307

id: 39,

name: "Dolorem dolores aut et facere natus ipsam.",

description: "Qui omnis ab qui rerum omnis. Ut ut id quod et. Odit officiis molestiae unde in perferendis. Et consequuntur enim suscipit asperiores accusamus eum.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7306

id: 40,

name: "Rerum inventore inventore et corporis quasi deleniti.",

description: "Est iusto deserunt sunt aut sapiente molestias. Fugit doloremque ut at est aut esse voluptas.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7305

id: 42,

name: "Perferendis blanditiis doloremque repellat asperiores.",

description: "Provident amet voluptatem vel similique in. Eligendi minus nobis incidunt ut ipsam excepturi molestias. Perferendis velit odit hic quia eligendi totam. Enim aliquam facere nam recusandae architecto ipsam.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7304

id: 43,

name: "Quam enim voluptatum delectus excepturi.",

description: "Sunt quia aperiam omnis in culpa in sint. Dolore molestiae aliquid aut laborum cum. Iure repudiandae praesentium perspiciatis.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#6305

id: 44,

name: "Nihil ea saepe culpa est.",

description: "Dolorem iste quia cumque aspernatur laborum et. Beatae dolorem aspernatur dolorum porro est dicta nihil sequi. Aut dolore officiis iure voluptatum non earum vel facere. Ducimus quaerat consequatur ut explicabo.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#6298

id: 45,

name: "Voluptatum iure odio qui fugit.",

description: "Ipsam esse inventore totam accusantium autem fugiat. Accusamus et occaecati voluptatem. Aliquid omnis eveniet dolor magnam. Doloremque ab ex explicabo consectetur dolores rem.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7354

id: 48,

name: "Recusandae optio nesciunt molestias maiores qui rem.",

description: "Et quaerat explicabo commodi autem non veniam voluptate. Ea totam vel dolore ad. Facilis eos dolorum quia similique quaerat nobis. Aut culpa rerum est ducimus similique est suscipit qui.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

App\Models\Curso {#7355

id: 49,

name: "Ipsum debitis et numquam accusamus.",

description: "Eius et consequuntur fuga et. Officia dolor rerum libero deserunt occaecati omnis. Sint eveniet temporibus maxime occaecati. Cum voluptatem natus sint sapiente provident quam. Eos aut est nihil nihil.",

category: "Diseño Web",

created\_at: "2024-01-10 14:31:20",

updated\_at: "2024-01-10 14:31:20",

},

],

}

Ahora vamos a hacer la misma consulta pero que ordene por Id de forma descendente

> $cursos = Curso::where('category','Diseño Web')->orderBy('id', 'desc')->get();

Ahora si queremos que solo me devuelva el primer objeto de la colección usamos el método first

> $cursos = Curso::where('category','Diseño Web')->orderBy('id', 'asc')->first();

Ahora si queremos que me muestre determinados campos ponemos

$cursos = Curso::select('name', 'description')->get();

Ahora puedo seguir usando los métodos aun que use select por ejemplo solo mostrare el name y description orderBy name

$cursos = Curso::select('name', 'description')->orderBy('name', 'asc')->get();

Si queremos podemos usar alias o el famoso as

$cursos = Curso::select('name as Nombre', 'description as Descripcion')->orderBy('name', 'asc')->get();

Si queremos un numero espesifico de registros usamos take()

$cursos = Curso::select('name', 'description')->orderBy('name', 'asc')->take(10)->get();

Ahora una consulta con un registro en particular

$cursos = Curso::where('name', 'Ipsum debitis et numquam accusamus.')->get();

Ahora podemos usar where con un operador relacional colocando 3 parametros

$cursos = Curso::where('id', '>', '48')->get();

# Crud

## Cómo listar y leer registros en Laravel

La lectura y listado de registros es una tarea fundamental en el desarrollo de aplicaciones web. En el decimocuarto capítulo del curso "Laravel desde cero", aprenderás a listar y leer registros en Laravel utilizando la función **all**, **find**, **findOrFail** y **where**. La función **all** te permite recuperar todos los registros de una tabla, mientras que la función **find** te permite recuperar un registro específico en función de su ID. La función **findOrFail** te permite recuperar un registro específico en función de su ID, pero lanzará una excepción si no se encuentra el registro. La función **where** te permite recuperar un conjunto de registros que cumplen una determinada condición. Además, en este capítulo, también aprenderás a utilizar la paginación para listar registros de manera eficiente y cómo realizar búsquedas utilizando la función **where** en combinación con la paginación. Con estos conocimientos, podrás listar y leer registros de manera efectiva y eficiente en tus aplicaciones Laravel.

Bueno lo primero es en el controller de curso ubicado en http devemos poner nuestra consulta tal y como lo usamos en tinker

CursoController.php

<?php

namespace App\Http\Controllers;

use App\Models\Curso;

use Illuminate\Http\Request;

class CursoController extends Controller

{

    public function index(){

        $cursos = Curso::all();

        return view('cursos.index', compact('cursos'));

    }

    public function create(){

        return view('cursos.create');

    }

    public function show($curso){

        return view('cursos.show', ['curso'=> $curso]);

    }

}

En este caso creamos la variable $cursos y con el método conpact se la mandamos a la vista en la url solo que sin $ solo cursos

Ahora en la vista debemos usar el siguiente código

@extends('layouts.plantilla')

@section('title', 'Cursos')

@section('content')

    <h1>Bienvenidos a la pagina de Cursos</h1>

    <ul>

        @foreach ($cursos as $curso)

            <li>

                {{$curso->name}}

            </li>

        @endforeach

    </ul>

@endsection()

Con la ayuda de un ul y un @foreach recorrimos los datos del objeto

Ahora vemos todo el listado pero literalmente son todos los registros lo mas indicado es mostrar los registros usando paginate el código queda así

En el archivo CursosController.php

<?php

namespace App\Http\Controllers;

use App\Models\Curso;

use Illuminate\Http\Request;

class CursoController extends Controller

{

    public function index(){

        //$cursos = Curso::all();

        $cursos = Curso::paginate();

        return view('cursos.index', compact('cursos'));

    }

    public function create(){

        return view('cursos.create');

    }

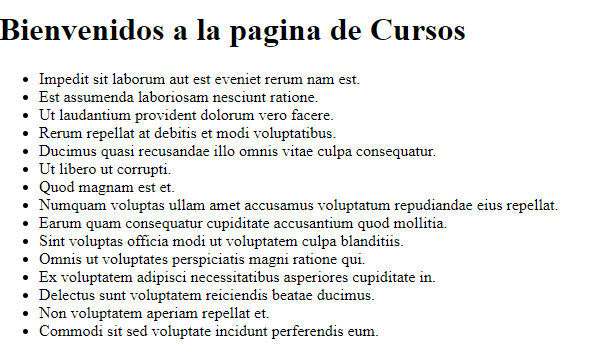
    public function show($curso){

        return view('cursos.show', ['curso'=> $curso]);

    }

}

Ahora solo muestra 15 registros



Si quiero ver la siguiente pagina a mi url le tendría que poner <http://127.0.0.1:8000/cursos?page=2>

Ahora como el usuario no puede hacer esto de la url le apoyamos con una barra de exploración para eso le ponemos el método links() en el view/cursos/index.blade.php

El código queda así

@extends('layouts.plantilla')

@section('title', 'Cursos')

@section('content')

    <h1>Bienvenidos a la pagina de Cursos</h1>

    <ul>

        @foreach ($cursos as $curso)

            <li>

                {{$curso->name}}

            </li>

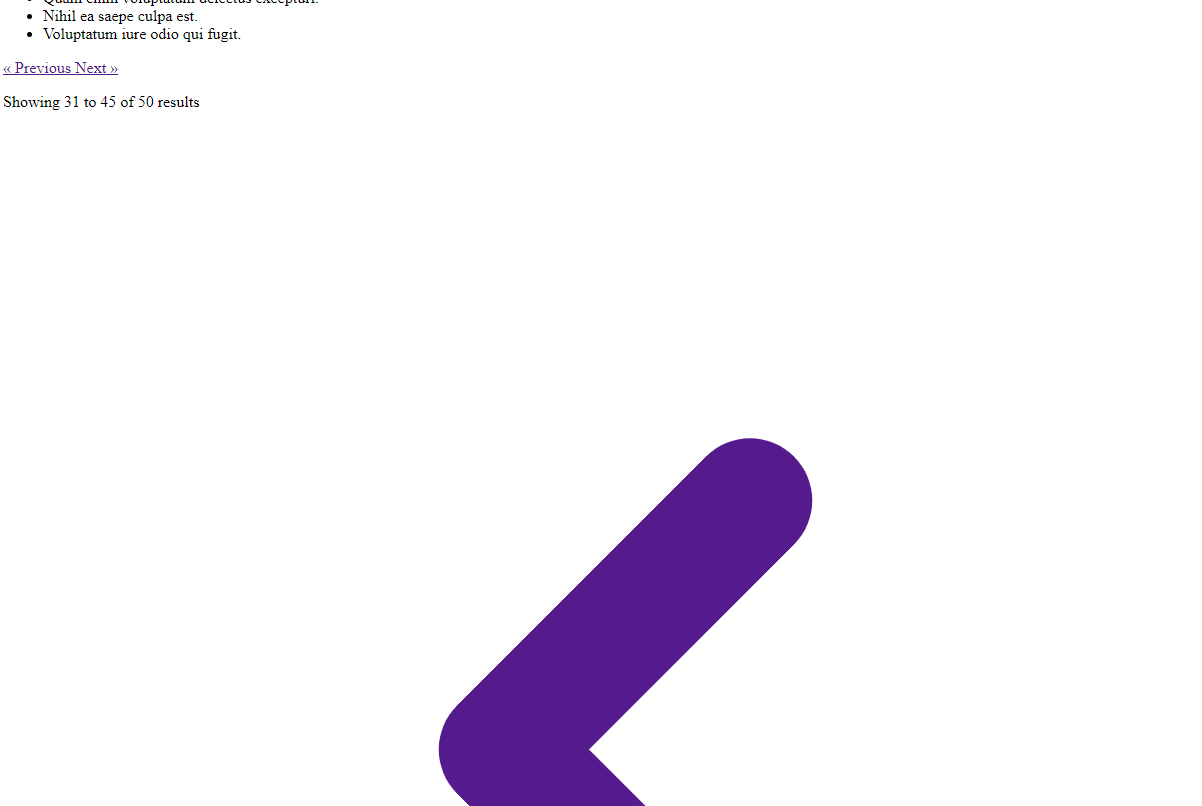
        @endforeach

    </ul>

    {{$cursos->links()}}

@endsection()

Ahora en la salida notamos que funciona pero se ve raro por que hay unos elementos que están con un estilo que no es compatible esto por que laravel utiliza Tailwind CSS



Para resolverlo via cdn incluire tailwind en mi layout que esta en view/layouts/plantilla.blade.php

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>@yield('title')</title>

  <link rel="stylesheet" href="style.css">

  <link rel="stylesheet" href="https://unpkg.com/tailwindcss@^1.0/dist/tailwind.min.css">

  <!--favicon-->

  <!--estilos-->

</head>

<body>

    <!--header-->

    <!--nav-->

    @yield('content')

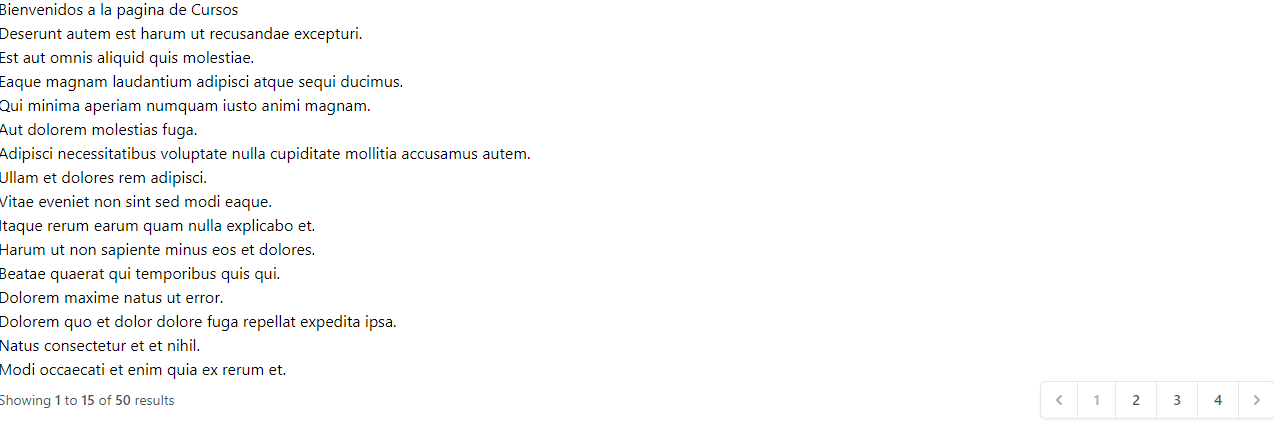
    <!--footer-->

    <!--script-->

</body>

</html>

Y listo ahora se vera así



Pero ahora vemos que nuestro titulo h1 de bienvenidos cambio.

Ahora es momento de empezar a usar Tailwing

Paso siguiente poner una link a crusos/créate para ello en en archivo index.blade.php vamos a crear un hipervínculo pero antes por sugerencia de Laravel debemos no usar la url sino el nombre y la propiedad de la ruta esto es muy simple nos vamos a nuestro archivo web.php que se encuentra en la carpeta routes y agregamos un método llamado name el código queda así

Route::get('/', HomeController::class);

Route::controller(CursoController::class)->group(function(){

    Route::get('cursos', 'index')->name('cursos.index');

    Route::get('cursos/create', 'create')->name('cursos.create');

    Route::get('cursos/{course}', 'show')->name('cursos.show');

});

Ahora vamos a hacer un elance usando una etiqueta y la variable route el código en index.blade.php queda así

@extends('layouts.plantilla')

@section('title', 'Cursos')

@section('content')

    <h1>Bienvenidos a la pagina de Cursos</h1>

    <a href="{{route('cursos.create')}}">Crear curso</a>

    <ul>

        @foreach ($cursos as $curso)

            <li>

                {{$curso->name}}

            </li>

        @endforeach

    </ul>

    {{$cursos->links()}}

@endsection()

Ahora vamos a hacer que los cursos no solo se enlisten sino que sean links que me manden al contenido del curso en la base de datos para ello debemos hacer unos cambios

Primero en el archivo index.blade.php que esta en vistas vamos a crear un link con un id

@extends('layouts.plantilla')

@section('title', 'Cursos')

@section('content')

    <h1>Bienvenidos a la pagina de Cursos</h1>

    <a href="{{route('cursos.create')}}">Crear curso</a>

    <ul>

        @foreach ($cursos as $curso)

            <li>

                <a href="{{route('cursos.show', $curso->id)}}">{{$curso->name}}</a>

            </li>

        @endforeach

    </ul>

    {{$cursos->links()}}

@endsection()

Ahora vamos a modificar nuestro controller en CursoController.php modifico la ruta y creo una variable curso que tome los datos de Curso::find($id) y la imprimo con compact

<?php

namespace App\Http\Controllers;

use App\Models\Curso;

use Illuminate\Http\Request;

class CursoController extends Controller

{

    public function index(){

        //$cursos = Curso::all();

        $cursos = Curso::paginate();

        return view('cursos.index', compact('cursos'));

    }

    public function create(){

        return view('cursos.create');

    }

    public function show($id){

        $curso = Curso::find($id);

        return view('cursos.show', compact('curso'));

    }

}

Ahora vamos a cambiar lo que imprimimos en show.blade.php

Para ver el titulo del curso, su categoría y la descripción además pondré un botón de regresar usando el route

@extends('layouts.plantilla')

@section('title', 'Cursos' . $curso->name)

@section('content')

    <h1><strong>Bienvenido a la pagina del Curso / </strong>{{$curso->name}}</h1>

    <a href="{{route('cursos.index')}}">Volver a Cursos</a>

    <p><strong>Categoria:</strong> {{$curso->category}}</p>

    <p><strong>Descripción:</strong> {{$curso->description}}</p>

@endsection()

## Cómo agregar y actualizar registros desde un formulario

Agregar y actualizar registros desde un formulario es una tarea común en el desarrollo de aplicaciones web con Laravel. Aprenderás a utilizar la función create y update, respectivamente, para agregar y actualizar registros en la base de datos. La función create te permite agregar un nuevo registro a una tabla utilizando los valores proporcionados por el usuario en un formulario, mientras que la función update te permite actualizar un registro existente en función de su ID utilizando los valores proporcionados por el usuario en un formulario. Aprenderás a utilizar el método fill para actualizar los valores de un registro de manera segura y a validar los datos del formulario antes de agregar o actualizar un registro. También conocerás cómo utilizar la función redirect para redirigir al usuario a una página específica después de agregar o actualizar un registro. Con estos conocimientos, podrás agregar y actualizar registros de manera segura y eficiente en tus aplicaciones web desarrolladas en Laravel.

### Formularios

### Agregar

Primero vamos a crear una ruta donde se cree la información para eso nos vamos al archivo web que esta en routes

Route::get('/', HomeController::class);

Route::controller(CursoController::class)->group(function(){

    Route::get('cursos', 'index')->name('cursos.index');

    Route::get('cursos/create', 'create')->name('cursos.create');

    //Crear ruta para crear un registro de tipo post

    Route::post('cursos','store')->name('cursos.store');

    Route::get('cursos/{id}', 'show')->name('cursos.show');

});

Ahora como store no es un método que exista en nuestro CursoController vamos a crearlo dirigiéndonos al archivo CursoController.php

Ponemos el método store

<?php

namespace App\Http\Controllers;

use App\Models\Curso;

use Illuminate\Http\Request;

class CursoController extends Controller

{

    public function index(){

        //$cursos = Curso::all();

        //$cursos = Curso::paginate();

        $cursos = Curso::orderBy('id', 'desc')->paginate();

        return view('cursos.index', compact('cursos'));

    }

    public function create(){

        return view('cursos.create');

    }

    public function store(Request $request){

        $curso = new Curso();

        $curso->name = $request->name;

        $curso->description = $request->description;

        $curso->category = $request->category;

        $curso->save();

        //Redireccionar al finalizar

        return redirect()->route('cursos.show');

    }

    public function show($id){

        $curso = Curso::find($id);

        return view('cursos.show', compact('curso'));

    }

}

Ahora editamos el créate.blade.php nota usamos en los comentario un token verlos

@extends('layouts.plantilla')

@section('title', 'Cursos /  Create')

@section('content')

    <h1>Bienvenidos a la pagina de Cursos / Create</h1>

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

        {{--En laravel cada que mandamos un post debemos anexar un token--}}

        @csrf

        <label>

            Nombre:

            <br>

            <input type="text" name="name"/>

        </label>

        <br>

        <label>

            Descripción:

            <br>

            <textarea name="description" rows="5"></textarea>

        </label>

        <br>

        <label>

            Categoria:

            <br>

            <input type="text" name="category"/>

        </label>

        <br>

        <input type="submit" value="Enviar Formulario">

    </form>

@endsection()

Así ya podemos agregar registros

### Editar Registros

Primero vamos a editar colocando un link en nuestro archivo show.blade.php

@extends('layouts.plantilla')

@section('title', 'Cursos' . $curso->name)

@section('content')

    <h1><strong>Bienvenido a la pagina del Curso / </strong>{{$curso->name}}</h1>

    {{--Navegación por el crud--}}

    <a href="{{route('cursos.index')}}">Volver a Cursos</a>

    <br>

    <a href="#">Editar curso</a>

    <p><strong>Categoria:</strong> {{$curso->category}}</p>

    <p><strong>Descripción:</strong> {{$curso->description}}</p>

@endsection()

después tenemos que crear la ruta para eso editamos el archivo web.php y después regresamos a editar el likn de show.blade

Creamos la ruta

Route::get('cursos/{id}/edit', 'edit')->name('cursos.edit');

Si ejecutamos y probamos nuestro link nos da como resultado <http://127.0.0.1:8000/cursos/53/edit>

Ahora debemos tomar ese 53 para mostrar nuestro formulario de edición primero como anteriormente lo hicimos nos colocamos en CursosController y editamos nuestro método

Creamos en el CursoController el método edit

public function edit(id){

    }

Después hacemos el hipervínculo en show.blade

    public function edit($id){

        $curso = Curso::find($id);

        return $curso;

    }

Aquí retornamos los datos del registro ahora hay que dar formato

{"id":53,"name":"Red Cola","description":"zxczxc","category":"Borrar","created\_at":"2024-01-12T19:33:08.000000Z","updated\_at":"2024-01-12T19:33:08.000000Z"}

Para ello debemos cambiar nuestro código para que nos retorne una vista que crearemos llamada edit.blade.php esto en la carpeta vistas/cursos el código de este archivo prácticamente es el mismo que el de créate solo adaptaremos unos pequeños cambios, sabemos que en los crud créate y update básicamente son los mismos.

@extends('layouts.plantilla')

@section('title', 'Cursos /  Edit')

@section('content')

    <h1>Bienvenidos a la pagina de Edición de Curso</h1>

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

        {{--En laravel cada que mandamos un post debemos anexar un token--}}

        @csrf

        <label>

            Nombre:

            <br>

            <input type="text" name="name" value="{{$curso->name}}"/>

        </label>

        <br>

        <label>

            Descripción:

            <br>

            <textarea name="description" rows="5">{{$curso->description}}</textarea>

        </label>

        <br>

        <label>

            Categoria:

            <br>

            <input type="text" name="category" value="{{$curso->category}}"/>

        </label>

        <br>

        <input type="submit" value="Actualizar Formulario">

    </form>

@endsection()

Ya solo nos falta dar la instrucción del post para que nos actualice el registro para ello como lo hicimos en el créate debemos hacer una nueva ruta en nuestro archivo web.php la cual la mandaremos en el action de nuestro formulario solo que usaremos el método put en lugar de post

    Route::get('cursos/{id}/edit', 'edit')->name('cursos.edit');

    //Crear ruta para el update

    Route::put('cursos/(curso)', 'update')->name('cursos.update');

Ahora voy a apuntar a la nueva ruta en el action del formulario pero aquí tenemos un detalle nativamente html no conoce el método put por lo que debemos integrarlo y no modificar el que establece nuestro formulario esto queda así

<h1>Bienvenidos a la pagina de Edición de Curso</h1>

    <form action="{{route('cursos.update')}}" method="POST">

        {{--En laravel cada que mandamos un post debemos anexar un token--}}

        @csrf

        {{--importamos el metodo put--}}

        @method('put');

        <label>

            Nombre:

            <br>

            <input type="text" name="name" value="{{$curso->name}}"/>

        </label>

        <br>

Ahora en el CursoController nos falta crear el método update.

# Modelo Entidad Relación

