

Contenido

Instalacion de COMPOSER	2
Instalacion de NODEJS.....	2
Instalacion de LARAVEL 6.0.....	2
AUTENTICACION EN LARAVEL 6.0(estos es opcional)	2
RUTAS(ROUTE) que es el mismo que urls.py en django	3
PLANTILLAS(BLADE) que es el mismo que templates en django	4
ESTRUCTURA DE CONTROL EN LAS PLANTILLAS(BLADE) que es el mismo que templates en django.....	5
CONTROLLERS que es el mismo que views en django (creo).....	6
BASE DE DATOS (para las primeras 3 tablas de laravel).....	7
CREANDO MODELOS (tablas en la base de datos) y ELOQUENT	9
ELOQUENT (operaciones con la base de datos).....	11
CONSULTAR UN DATO A LA BASE DE DATOS	12
AGREGAR DATOS A LA BASE DE DATOS	13
VALIDAR INPUT	15
ACTUALIZAR DATOS	16
ELIMINAR DATOS.....	17
PAGINACION.....	18
MODIFICAR TABLAS YA EXISTENTES CON LAS MIGRACIONES DE LARAVEL (https://www.youtube.com/watch?v=ft9is7r6QoM)	20
2.- Debemos de crear otra nueva migración para agregar el campo usuario a la tabla nota, el nombre de la migración debe ser lo mas descriptiva posible en nuestro ejemplo se llamara: add_usuario_to_notas y debemos de ejecutar el comando php artisan make:migration quedando de la siguiente forma la línea de comando: php artisan make:migration add_usuario_to_notas	21
3.- Modificaremos el contenido de la nueva migración creada que nuestro caso fue: 2021_01_11_135416_add_usuario_to_notas.php, en la funciones up() y down().....	21
6.- Ejecutamos en la linea de comando: php artisan migrate:rollback.....	22
7.- Por ultimo ejecutamos en la linea de comando: php artisan migrate.....	23
AUTENTICACION Y PROTEGIENDO RUTAS	23
RUTAS CON RECURSO (RESOURCE CONTROLLERS)	24
BASE DE DATOS RELACIONALES	24
RUTA PARA COPIAR LOS ARCHIVOS JS	26
RUTA PARA COPIAR LOS ARCHIVOS CSS	26
POSIBLES SOLUCIONES PARA EL ESTADO DE LA PAGINAS 419 PAGE EXPIRED	26
MODIFICAR LAS PLANTILLAS DE ERROR EN LARAVEL 6: 401, 403, 404, 419, 429, 500, 503.	26
RELACION ENTRE TABLAS MUCHOS A MUCHOS.....	27
VALIDACION DE CAMPOS.....	29
AGREGAR ROL AL USUARIO AL MOMENTO DE REGISTRARSE.	31
AGREGAR VARIOS ROLES A UN USUARIO y EDITARLOS.....	32

Instalacion de COMPOSER

- 1.- Instalar **COMPOSER** descargarlo en: <https://getcomposer.org/download/>
- 2.- Al instalar seleccionar la ruta donde esta el archivo **php.exe** en nuestra carpeta en mi caso del proyecto esta en **xampp 7**

Instalacion de NODEJS

- 1.- Descargar **NODEJS** en: <https://nodejs.org/es/download/>

Instalacion de LARAVEL 6.0

- 1.- Entrar con el símbolo del sistema a la carpeta donde se creara el proyecto en mi caso seria en la ruta: **xampp7/htdocs/**
- 2.- después que entramos en la ruta donde estará nuestro proyecto tecleamos en el símbolo del sistema: **composer create-project --prefer-dist laravel/laravel NOMBREDELPROYECTO "6.0.*"**

```
Simbolo del sistema
Microsoft Windows [Versión 10.0.18363.1256]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\monchis>cd ..
C:\Users>cd ..
C:\>cd xampp7
C:\xampp7>cd htdocs

C:\xampp7\htdocs>composer create-project --prefer-dist laravel/laravel hmvideo "6.0.*"
Creating a "laravel/laravel" project at "./hmvideo"
Installing laravel/laravel (v6.0.2)
- Downloading laravel/laravel (v6.0.2)
- Installing laravel/laravel (v6.0.2): Extracting archive
Created project in C:\xampp7\htdocs\hmvideo
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 90 installs, 0 updates, 0 removals
- Locking dnoegel/php-xdg-base-dir (v0.1.1)
- Locking doctrine/inflector (2.0.3)
- Locking doctrine/instantiator (1.4.0)
- Locking doctrine/lexer (1.2.1)
- Locking dragonmantank/cron-expression (v2.3.1)
- Locking egulias/email-validator (2.1.25)
- Locking facade/flare-client-php (1.3.7)
- Locking facade/ignition (1.16.4)
- Locking facade/ignition-contracts (1.0.2)
```

De ahí comenzara instalarse **laravel 6.0** junto con la carpeta del proyecto.

AUTENTICACION EN LARAVEL 6.0(estos es opcional)

- 1.- En el símbolo del sistema dentro de la carpeta del proyecto y ejecutar lo siguiente:

```
composer require laravel/ui "*"^1.0" --dev
```

- 2.- Después en el símbolo del sistema dentro de la carpeta del proyecto ejecutar lo siguiente:

```
php artisan ui vue --auth
```

3.- Despues en el símbolo del sistema dentro de la carpeta del proyecto ejecutar lo siguiente:

npm i ←--- ESTO YA NO SE HACE

```
C:\xampp7\htdocs\hmvideo>npm i
npm WARN deprecated axios@0.19.2: Critical security vulnerability fixed in v0.21.1. For more information, see https://github.com/axios/axios/pull/3410
npm WARN deprecated unix@0.1.0: Please see https://github.com/lydell/unix#deprecated
npm WARN deprecated popper.js@1.16.1: You can find the new Popper v2 at @popperjs/core, this package is dedicated to the legacy v1
npm WARN deprecated chokidar@2.1.8: Chokidar 2 will break on node v14+. Upgrade to chokidar 3 with 15x less dependencies
npm WARN deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated
npm WARN deprecated fsevents@1.2.12: fsevents 1 will break on node v14+ and could be using insecure binaries. Upgrade to fsevents 2
```

RUTAS(ROUTE) que es el mismo que urls.py en django

Estas se encuentran en: **Raízdelproyecto/routes/**

```
13
14 Route::get('/', function () {
15     return view('welcome');
16 });
17
18 /* Ruta de prueba */
19 Route::get('bienvenido', function () {
20     return 'Hola';
21 });
22
23 /* Ruta paso de parametros */
24 /* el signo(?) despues de numero significa que el parametro es opcional, el = 'sin numero' indica que es
   opcional el parametro*/
25 /* el ->where('numero', '[0-9]+') indica que solo recibira como parametros numeros si se manda un texto
   enviar un 404 Not Found */
26
27 Route::get('fotos/{numero?}',function($numero = 'sin numero'){
28     return 'Estas en la galeria de fotos'.$numero;
29 })->where('numero', '[0-9]+');
30
31
32 /* OTRA FORMA DE DECLARAR LAS RUTAS con pase de parametros */
33 Route::view('/galeria', 'fotos', ['name' => 'Monchis']);
34
35 Auth::routes();
36
37 Route::get('/home', 'HomeController@index')->name('home');
38
```


ESTRUCTURA DE CONTROL EN LAS PLANTILLAS(BLADE) que es el mismo que templates en django

Archivo de las rutas (**web.php**) que se encuentra en **routes/web.php**

```
42
43 Route::get('nosotros/{nombre?}',function($nombre=null){
44     $equipo = ['Ignacio','Juanito','Pedrito'];
45     // return view('nosotros',['equipo'=>$equipo,'nombre'=>$nombre]);
46     return view('nosotros',compact('equipo','nombre')); // esto es lo mismo que la línea de arriba
47 }->name('nosotros');
```

Plantilla (**nosotros.blade.php**) que se encuentra en **resources/views/nosotros.blade.php**

```
1  @extends('plantilla')
2
3  @section('seccion')
4      <h1>Este es mi equipo de trabajo</h1>
5      @foreach($equipo as $item)
6          <a href="{{ route('nosotros',$item) }}" class="h4 text-danger">{{ $item }}</a>
7      @endforeach
8
9      @if(!empty($nombre))
10         @switch($nombre)
11             @case($nombre=='Ignacio')
12                 <h2>El nombre de nacho es: {{ $nombre }}</h2>
13             @break
14             @case($nombre=='Juanito')
15                 <h2>El nombre de juanote es: {{ $nombre }}</h2>
16             @break
17             @case($nombre=='Pedrito')
18                 <h2>El nombre de pedro es: {{ $nombre }}</h2>
19             @break
20         @endswitch
21     @endif
22 @endsection
```

CONTROLLERS que es el mismo que views en django (creo)

1.- En el símbolo del sistema tecleamos: `php artisan make:controller`

NOMBREDELCONTROLADOR

```
C:\xampp7\htdocs\hmvideo>php artisan make:controller PageController
Controller created successfully.
```

Los controladores se ubican en: **raízdelproyecto/app/Http/Controllers**

Como quedaría nuestro archivo de rutas (**web.php**)

```
14 Route::get('/', 'PageController@inicio');
15
16 /* Ruta de prueba */
17 Route::get('bienvenido', function () {
18     return 'Hola';
19 });
20
21 /* Ruta paso de parametros */
22 /* el signo(?) despues de numero significa que el parametro es opcional, e
23    opcional el parametro */
24 /* el ->where('numero', '[0-9]+') indica que solo recibira como parametros
25    enviar un 404 Not Found */
26
27 Route::get('fotos/{numero?}', function($numero = 'sin numero'){
28     return 'Estas en la galeria de fotos'.$numero;
29 })->where('numero', '[0-9]+');
30
31 /* OTRA FORMA DE DECLARAR LAS RUTAS con pase de parametros */
32 Route::view('/galeria', 'fotos', ['name' => 'Monchis']);
33
34 Route::get('nuevafotos/', 'PageController@fotos')->name('foto');
35
36 Route::get('blog/', 'PageController@noticias')->name('noticias');
37
38 Route::get('nosotros/{nombre?}', 'PageController@nosotros')->name('nosotros');
```

Como quedaría nuestro archivo donde esta el controlador (**PageController.php**)

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class PageController extends Controller
8 {
9     public function inicio(){
10         return view('welcome');
11     }
12
13     public function fotos(){
14         return view('nuevafotos');
15     }
16
17     public function nosotros($nombre=null){
18         $equipo = ['Ignacio','Juanito','Pedrito'];
19         // return view('nosotros',['equipo'=>$equipo,'nombre'=>$nombre]);
20         return view('nosotros',compact('equipo','nombre')); // esto es lo mismo que la línea de arriba
21     }
22
23     public function noticias(){
24         return view('blog');
25     }
26 }
27
```


BASE DE DATOS (para las primeras 3 tablas de laravel)

- 1.- Creamos nuestra base de datos en **phpmyadmin** con el cotejamiento **utf8_general_ci**
- 2.- Abrimos el archivo **.env** que se encuentra en la raíz del proyecto y cambiamos los datos de conexión a la base de datos:

```
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:jrzYIMZH/mtfCzqjBakEw/Xs9BflhNXrHBfJqG/6ooc=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8
9 DB_CONNECTION=mysql
10 DB_HOST=127.0.0.1
11 DB_PORT=3306
12 DB_DATABASE=hmvideo
13 DB_USERNAME=root
14 DB_PASSWORD=
15
16 BROADCAST_DRIVER=log
17 CACHE_DRIVER=file
18 QUEUE_CONNECTION=sync
19 SESSION_DRIVER=file
20 SESSION_LIFETIME=120
21
```

- 3.- Si al momento de realizar el comando **php artisan migrate** manda este error:

```
Símbolo del sistema
C:\xampp7\htdocs\hmvideo>php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table

Illuminate\Database\QueryException : SQLSTATE[42000]: Syntax error or access violation: 1071 Specified key was too long; max key length is 767 bytes (SQL: alter table `users` add unique `users_email_unique`(`email`))

at C:\xampp7\htdocs\hmvideo\vendor\laravel\framework\src\Illuminate\Database\Connection.php:669
665|         // If an exception occurs when attempting to run a query, we'll format the error
666|         // message to include the bindings with SQL, which will make this exception a
667|         // lot more helpful to the developer instead of just the database's errors.
668|         catch (Exception $e) {
669|             throw new QueryException(
670|                 $query, $this->prepareBindings($bindings), $e
671|             );
672|         }
673|

Exception trace:
1 PDOException::("SQLSTATE[42000]: Syntax error or access violation: 1071 Specified key was too long; max key length is 767 bytes")
  C:\xampp7\htdocs\hmvideo\vendor\laravel\framework\src\Illuminate\Database\Connection.php:463
2 PDOStatement::execute()
  C:\xampp7\htdocs\hmvideo\vendor\laravel\framework\src\Illuminate\Database\Connection.php:463

Please use the argument -v to see more details.
```

4.- Debemos de agregar **2 líneas de código** al archivo **AppServiceProvider.php** que se encuentra en **App/Providers/**

```
use Illuminate\Support\Facades\Schema;
```

```
Schema::defaultStringLength(191);
```

```
1 <?php
2
3 namespace App\Providers;
4
5 use Illuminate\Support\ServiceProvider;
6 use Illuminate\Support\Facades\Schema; // AGREGAMOS ESTA LINEA SI AL MOMENTO DEL MIGRATE NOS MANDA ERROR EL
  MYSQL, POR SU VERSION AGREGADO POR MONCHIS
7
8 class AppServiceProvider extends ServiceProvider
9 {
10     /**
11      * Register any application services.
12      *
13      * @return void
14      */
15     public function register()
16     {
17         //
18     }
19
20     /**
21      * Bootstrap any application services.
22      *
23      * @return void
24      */
25     public function boot()
26     {
27         Schema::defaultStringLength(191); // AGREGAMOS ESTA LINEA SI AL MOMENTO DEL MIGRATE NOS MANDA ERROR
  EL MYSQL, POR SU VERSION AGREGADO POR MONCHIS
28     }
29 }
30
```

5.- Debemos eliminar la **base de datos del proyecto**



Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño	Resid
migrations	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	16 KB	
users	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	16 KB	
test			InnoDB	utf8_general_ci	32 KB	

6.- Volvemos a ejecutar en la terminal del ms-dos: **php artisan migrate** y con eso quedaría:

```
C:\xampp7\htdocs\hmvideo>php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (0.33 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (0.29 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.18 seconds)
```

CREANDO MODELOS (tablas en la base de datos) y ELOQUENT

1.- Para el ejemplo crearemos el modelo **Nota**, en el símbolo del sistema debemos de teclear:
php artisan make:model Nota -m

Lo que hace esto es que **crea el modelo** y **crea la migración**

Los modelos se ubican en: **App/**

Las migraciones se ubicacn en: **Database/migrations/**

```
C:\xampp7\htdocs\hmvideo>php artisan make:model Nota -m
Model created successfully.
Created Migration: 2021_01_08_144000_create_notas_table
```

2.- Debemos abrir el archivo de la migración en este caso sería el **2021_01_08_144000_create_notas_table**

```
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  class CreateNotasTable extends Migration
8  {
9      /**
10       * Run the migrations.
11       *
12       * @return void
13       */
14     public function up()
15     {
16         Schema::create('notas', function (Blueprint $table) {
17             $table->bigIncrements('id');
18             $table->string('nombre'); // campo agregado monchis
19             $table->text('descripcion'); // campo agregado monchis
20             $table->timestamps();
21         });
22     }
23
24     /**
25      * Reverse the migrations.
26      *
27      * @return void
28      */
29     public function down()
30     {
31         Schema::dropIfExists('notas');
32     }
33 }
34
```

3.- En el símbolo del sistema tecleamos: **php artisan migrate**, y se crean los demás campos que agregamos en nuestra migración

ELOQUENT (operaciones con la base de datos)

Como ejemplos modificaremos la plantilla de inicio (**welcome.blade.php**) para mostrar los registros de la tabla **notas**.

Antes que nada debemos de modificar nuestro **controlador** que ya hemos hecho (**PageController.php**)

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App; // ESTO SE AGREGA PARA ACCEDER A LOS MODELOS
7
8 class PageController extends Controller
9 {
10     public function inicio(){
11         $notas = App\Nota::all(); // CONSULTA TODOS LOS REGISTROS A LA TABLA NOTA DE LA BASE DE DATOS
12         return view('welcome',compact('notas'));
13     }
14
15     public function fotos(){
16         return view('nuevafotos');
17     }
18
19     public function nosotros($nombre=null){
20         $equipo = ['Ignacio','Juanito','Pedrito'];
21         // return view('nosotros',['equipo'=>$equipo,'nombre'=>$nombre]);
22         return view('nosotros',compact('equipo','nombre')); // esto es lo mismo que la linea de arriba
23     }
24
25     public function noticias(){
26         return view('blog');
27     }
28 }
```

```

1 <!doctype html>
2 <html lang="en">
3 <head>
4 <!-- Required meta tags -->
5 <meta charset="utf-8">
6 <meta name="viewport" content="width=device-width, initial-scale=1">
7
8 <!-- Bootstrap CSS -->
9 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css" rel="stylesheet"
  integrity="sha384-giJF6kkoqN00vy+hMDP7azOul0xtbficaT9wjKhr8RbDVddVHyTfAAsrekwKmp1" crossorigin="
  anonymous">
10
11 <title>Hello, world!</title>
12 </head>
13 <body>
14 <div class="container my-4">
15 <h1 class="display-4">Notas</h1>
16 <table class="table">
17 <thead>
18 <tr>
19 <th scope="col">#id</th>
20 <th scope="col">Nombre</th>
21 <th scope="col">Descripcion</th>
22 <th scope="col">Handle</th>
23 </tr>
24 </thead>
25 <tbody>
26 @foreach($notas as $nota)
27 <tr>
28 <th scope="row">{{ $nota->id }}</th>
29 <td>{{ $nota->nombre }}</td>
30 <td>{{ $nota->descripcion }}</td>
31 <td>{{ $nota->handle }}</td>
32 </tr>
33 @endforeach
34 </tbody>
35 </table>
36 </div>
37

```

1.- En nuestra plantilla en este ejemplo usaremos (**welcome.blade.php**), mandamos el id y la ruta para consultar el dato:

12

2.- En nuestro controlador (**PageController.php**) creamos la función de la ruta:

```
public function detalle($id){
    $nota = App\Nota::findOrFail($id);
    return view('notas.detalle',compact('nota')); // notas.detalle es como si tuvieramos notas/detalle/
    que es una carpeta que se crea para que este organizado el proyecto
}
```

3.- En nuestro archivo de rutas (**web.php**) creamos la ruta con el parámetro:

```
Route::get('/detalle/{id}', 'PageController@detalle')->name('notas.detalle');
```

4.- Creamos nuestra plantilla (**detalle.blade.php**)

```
@extends('plantilla')

@section('seccion')

    <h1>Detalle Nota</h1>
    <h4>id: {{ $nota->id }}</h4>
    <h4>Nombre: {{ $nota->nombre }}</h4>
    <h4>Detalle: {{ $nota->descripcion }}</h4>

@endsection
```

AGREGAR DATOS A LA BASE DE DATOS

1.- Archivos de ruta (**web.php**)

```
Route::post('/', 'PageController@crear')->name('notas.crear');
```

2.- Nuestro controlador (PageController.php)

```
public function crear(Request $request){  
    // return $request->all(); <--- esta linea manda los datos del formulario  
    $notaNueva = new App\Nota;  
    $notaNueva->nombre = $request->nombre;  
    $notaNueva->descripcion = $request->descripcion;  
  
    $notaNueva->save();  
  
    return back()->with('mensaje','Nota agregada!'); // back() nos regresa a la ventana anterior y manda  
    un mensaje en la plantilla  
}
```

3.- Nuestra plantilla (welcome.blade.php)

```
<div class="container my-4">  
    <h1 class="display-4">Notas</h1>  
  
    @if(session('mensaje')) <!-- aqui es para mostrar los mensajes los alert en bootstrap -->  
        <div class="alert alert-success"> <!-- aqui es para mostrar los mensajes los alert en bootstrap  
        -->  
            {{ session('mensaje') }}<!-- aqui es para mostrar los mensajes los alert en bootstrap -->  
        </div><!-- aqui es para mostrar los mensajes los alert en bootstrap -->  
    @endif<!-- aqui es para mostrar los mensajes los alert en bootstrap -->  
  
    <form action="{{ route('notas.crear') }}" method="POST">  
        @csrf  
        <input type="text" name="nombre" placeholder="Nombre" class="form-control mb-2">  
        <input type="text" name="descripcion" placeholder="Descripcion" class="form-control mb-2">  
        <button class="btn btn-primary btn-block" type="submit">Agregar</button>  
    </form>
```


VALIDAR INPUT

1.- En nuestro controlador (**PageController.php**)

```
public function crear(Request $request){
    // return $request->all(); <--- esta línea manda los datos del formulario

    $request->validate([
        'nombre' => 'required', //<-- aquí se hacen las validaciones de los campos
        'descripcion' => 'required' //<-- aquí se hacen las validaciones de los campos
    ]);

    $notaNueva = new App\Nota;
    $notaNueva->nombre = $request->nombre;
    $notaNueva->descripcion = $request->descripcion;

    $notaNueva->save();

    return back()->with('mensaje','Nota agregada!'); // back() nos regresa a la ventana anterior y manda
    un mensaje en la plantilla
}
```

2.- Nuestra plantilla (welcome.blade.php)

```
<div class="container my-4">
  <h1 class="display-4">Notas</h1>

  @if(session('mensaje'))
    <div class="alert alert-success">
      {{ session('mensaje') }}
    </div>
  @endif

  <form action="{{ route('notas.crear') }}" method="POST">
    @csrf

    @error('nombre')
      <div class="alert alert-danger">
        El nombre es obligatorio
      </div>
    @enderror

    @error('descripcion')
      <div class="alert alert-danger">
        El descripcion es obligatorio
      </div>
    @enderror

    <input type="text" name="nombre" placeholder="Nombre" class="form-control mb-2" value="{{ old('nombre') }}"> <!-- el value="{{ old('nombre') }}" sirve para al momento de mandar el error de validacion los valores que tenga el input no desaparezcan -->
    <input type="text" name="descripcion" placeholder="Descripcion" class="form-control mb-2" value="{{ old('descripcion') }}">
    <button class="btn btn-primary btn-block" type="submit">Agregar</button>
  </form>
```

ACTUALIZAR DATOS

1.- Nuestro archivo de rutas (web.php)

```
Route::get('/editar/{id}', 'PageController@editar')->name('notas.editar'); // ruta para mostrar el formulario de edicion

Route::put('/editar/{id}', 'PageController@update')->name('notas.update'); // ruta para el controlador donde actualiza la informacion en la base de datos
```

2.- Nuestro controlador (PageController.php)

```
public function editar($id){ /// para mostrar el formulario de edicion
    $nota = App\Nota::findOrFail($id);
    return view('notas.editar',compact('nota'));
}

public function update(Request $request, $id){ /// para actualizar la informacion en la base de datos

    $notaUpdate = App\Nota::findOrFail($id);

    $notaUpdate->nombre = $request->nombre;
    $notaUpdate->descripcion = $request->descripcion;

    $notaUpdate->save();

    return back()->with('mensaje','Nota actualizada');
}
```

3.- Plantilla o formulario para editar (**editar.blade.php**)

```
@extends('plantilla')

@if(session('mensaje'))
    <div class="alert alert-success">{{ session('mensaje') }}</div>
@endif

@section('seccion')
    <h1>Editar Nota: {{ $nota->id }}</h1>
    <form action="{{ route('notas.update', $nota->id) }}" method="POST">
        @method('PUT')
        @csrf

        @error('nombre')
            <div class="alert alert-danger">
                El nombre es obligatorio
            </div>
        @enderror

        @error('descripcion')
            <div class="alert alert-danger">
                El descripcion es obligatorio
            </div>
        @enderror

        <input type="text" name="nombre" placeholder="Nombre" class="form-control mb-2" value="{{ $nota->nombre }}">
        <input type="text" name="descripcion" placeholder="Descripcion" class="form-control mb-2" value="{{ $nota->descripcion }}">
        <button class="btn btn-warning btn-block" type="submit">Editar</button>
    </form>
@endsection
```

ELIMINAR DATOS

1.- Nuestro archivo de rutas (**web.php**)

```
Route::delete('/eliminar/{id}', 'PageController@eliminar')->name('notas.eliminar'); // ruta para el controlador donde eliminara la informacion en la base de datos
```

```
public function eliminar($id){    /// para eliminar la informacion en la base de datos

    $notaEliminar = App\Nota::findOrFail($id);

    $notaEliminar->delete();

    return back()->with('mensaje','Nota eliminada');
}
```

2.- Nuestro controlador (**PageController.php**)

3.- En la plantillas del listado (**welcome.blade.php**) en el botón **eliminar** debemos de agregarlo con un formulario

```
<table class="table">
<thead>
<tr>
<th scope="col">#id</th>
<th scope="col">Nombre</th>
<th scope="col">Descripcion</th>
<th scope="col">Acciones</th>
</tr>
</thead>
<tbody>
@foreach($notas as $item)
<tr>
<th scope="row">{{ $item->id }}</th>
<td>
<a href="{{ route('notas.detalle', $item) }}" --> aqui se manda el id y la ruta del id -->
{{ $item->nombre }}
</a>
</td>
<td>{{ $item->descripcion }}</td>
<td>
<a href="{{ route('notas.editar', $item) }}" class="btn btn-warning btn-sm">
Editar
</a>
<form action="{{ route('notas.eliminar', $item) }}" method="POST" class="d-inline">
@method('DELETE')
@csrf
<button class="btn btn-danger btn-sm" type="submit">Eliminar</button>
</form>
</td>
</tr>
</tbody>
@endforeach
</table>
```

PAGINACION

1.- En nuestro controlador (**PageController.php**) agregamos **paginate** en la función inicio:

```
class PageController extends Controller
{
    public function inicio(){
        //$notas = App\Nota::all(); // CONSULTA TODOS LOS REGISTROS A LA TABLA NOTA DE LA BASE DE DATOS
        $notas = App\Nota::paginate(2); // paginate(2) muestra unicamente 2 elementos
        return view('welcome',compact('notas'));
    }
}
```

2.- En nuestra plantilla (**welcome.blade.php**) al final de nuestra tabla agregamos el objeto que mandamos desde el controlador y el método **links()**.

```
        Editar
    </a>

    <form action="{{ route('notas.eliminar', $item) }}" method="POST" class="d-inline">
        @method('DELETE')
        @csrf
        <button class="btn btn-danger btn-sm" type="submit">Eliminar</button>
    </form>

    </td>
</tr>
@endforeach
</tbody>
</table>
{{ $notas->links() }} <!-- con esto se agrega la paginacion -->
</div>
```

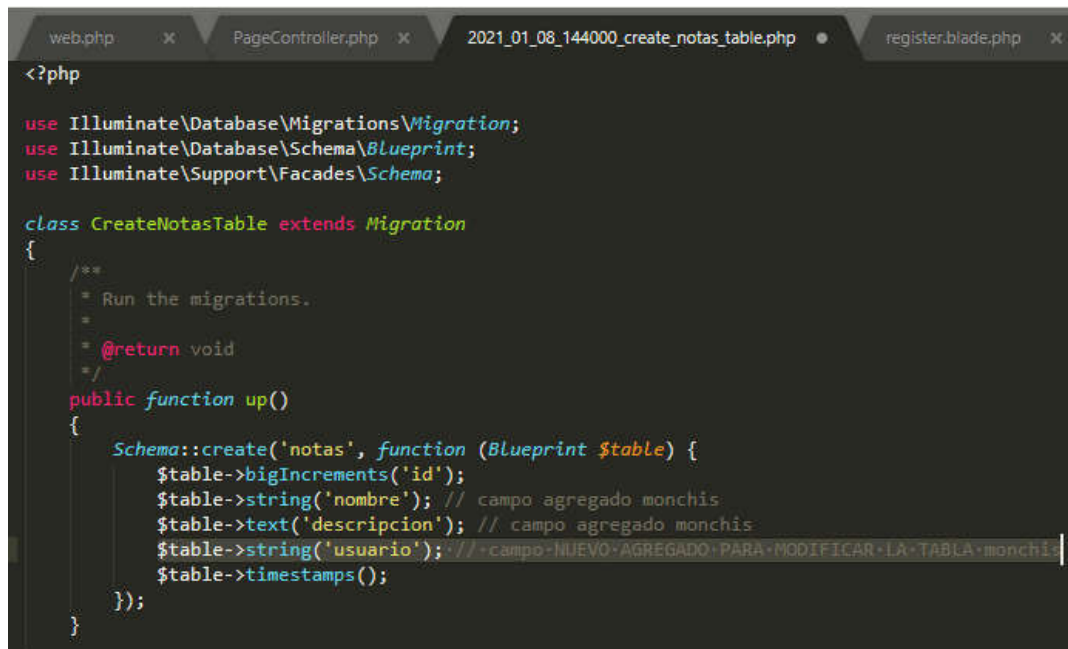
MODIFICAR TABLAS YA EXISTENTES CON LAS MIGRACIONES DE LARAVEL

(<https://www.youtube.com/watch?v=fT9is7r6QoM>)

Cuando queremos agregar un nuevo campo un modelo ya existente se tiene que realizar lo siguiente:

1.- En la migración ya creada, como ejemplo le agregaremos el campo usuario a la tabla notas donde ese campo tendrá el correo electrónico del usuario, el campo se llamara usuario, como inicio debemos agregar el nuevo campo a nuestra migration que en este caso seria la

2021_01_08_144000_create_notas_table.php



```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateNotasTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('notas', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('nombre'); // campo agregado monchis
            $table->text('descripcion'); // campo agregado monchis
            $table->string('usuario'); // campo NUEVO AGREGADO PARA MODIFICAR LA TABLA monchis
            $table->timestamps();
        });
    }
}
```


2.- Debemos de crear otra nueva migración para agregar el campo usuario a la tabla nota, el nombre de la migración debe ser lo mas descriptiva posible en nuestro ejemplo se llamara: **add_usuario_to_notas** y debemos de ejecutar el comando **php artisan make:migration add_usuario_to_notas** quedando de la siguiente forma la línea de comando: **php artisan make:migration add_usuario_to_notas**

```
C:\xampp7\htdocs\hmvideo>php artisan make:migration add_usuario_to_notas
Created Migration: 2021_01_11_135416_add_usuario_to_notas
```

3.- Modificaremos el contenido de la nueva migración creada que nuestro caso fue: **2021_01_11_135416_add_usuario_to_notas.php**, en la funciones **up()** y **down()**

```
web.php PageController.php x 2021_01_08_144000_create_notas_table.php x
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class AddUsuarioToNotas extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('notas', function (Blueprint $table) {
            $table->string('usuario',250)->nullable();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('notas', function (Blueprint $table) {
            $table->dropColumn('usuario');
        });
    }
}
```

4.- En el símbolo del sistema ejecutamos: **php artisan migrate**

```
C:\xampp7\htdocs\hmvideo>php artisan migrate
Migrating: 2021_01_11_135416_add_usuario_to_notas
Migrated: 2021_01_11_135416_add_usuario_to_notas (0.43 seconds)
```

5.- **Esto es opcional** si nosotros queremos insertar el nuevo campo **usuario** después de un campo y no al final de la tabla debemos de modificar la ultima migración que se creo y colocar la siguiente línea de código:

```
web.php PageController.php x 2021_01_08_144000_create_notas_table.php x 2021_01_11_135416_add_usuario_to_notas.php x
* Run the migrations.
*
* @return void
*/
public function up()
{
    Schema::table('notas', function (Blueprint $table) {
        $table->string('usuario',250)->nullable()->after('descripcion'); // ->after('nombre') aqui
        indicamos que el campo usuario se agregara despues del campo descripcion
    });
}
```

6.- Ejecutamos en la linea de comando: **php artisan migrate:rollback**

```
C:\xampp7\htdocs\hmvideo>php artisan migrate:rollback
Rolling back: 2021_01_11_135416_add_usuario_to_notas
Rolled back: 2021_01_11_135416_add_usuario_to_notas (0.36 seconds)
```

7.- Por ultimo ejecutamos en la linea de comando: **php artisan migrate**

```
C:\xampp7\htdocs\hmvideo>php artisan migrate
Migrating: 2021_01_11_135416_add_usuario_to_notas
Migrated: 2021_01_11_135416_add_usuario_to_notas (0.25 seconds)
```

AUTENTIFICACION Y PROTEGIENDO RUTAS

1.- Despues hacer lo de la primera hoja del manual si no jalaran los estilos entonces podemos sustituir la hoja de estilos del archivo **app.blade.php** que se encuentra en **resources/views/layouts/** la **línea 20** y podemos sustituirlo por la ruta de nuestro **css de preferencia** en mi caso lo sustitui por la de **bootstrap**.

2.- Registramos el nuevo usuarios en la ruta **register/**

3.- Para proteger las rutas es decir si el usuario quiere acceder a formulario sin loguearse se puede proteger de la siguiente forma agregando el **middleware** en nuestro controlador (**PageController.php**)



```
<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App; // ESTO SE AGREGA PARA ACCEDER A LOS MODELOS

class PageController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth'); // ESTO PROTEGE A TODAS LAS RUTAS DEL CONTROLAR PARA QUE NINGUN USUARIO
        PUEDA ACCEDER A ALGUN FORMULARIO SIN ANTES LOGUEARSE
    }
}
```

RUTAS CON RECURSO (RESOURCE CONTROLLERS)

- 1.- Despues de crear el modelo con: **php artisan make:model NOMBREDELMODELO --m**
- 2.- Tecleamos en el símbolo del sistema: **php artisan make:controller CosaController --resource**

```
C:\xampp7\htdocs\hmvideo>php artisan make:controller CosaController --resource
Controller created successfully.
```

- 3.- Agregamos la ruta del controlador en nuestro archivo de rutas (**web.php**).
- 4.- En nuestro archivo controller para nuestro ejemplo seria (**CosasController.php**) de manera automática se crean las rutas para listar registros, agregar, modificar, eliminar, donde debemos incluir los códigos ahí para que hagan las operaciones que nosotros deseamos.

BASE DE DATOS RELACIONALES

- 1.- Las bases de datos relacionales se trabajan en el **controlador de cada tabla, ubicados en app/Http/Controllers/**

Ejemplo:

Modelo Libro:

```
public function up()
{
    Schema::create('libros', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->string('titulo');
        $table->mediumText('descripcion');
        $table->text('contenido');
        $table->timestamp('fecha')->nullable();
        $table->unsignedInteger('categoria_id'); // Relación con categorias
        $table->timestamps();
    });
}
```

2.- Archivo **Libro.php** ubicado en **app/**

```
2019_06_15_142452_create_etiqueta_libro_table.php
Libro.php > Libro > etiquetas

namespace App;

use Illuminate\Database\Eloquent\Model;

class Libro extends Model
{
    public function categoria(){ //$Libro->categoria->nombre
        return $this->belongsTo(Categoria::class); //Pertenece a una
        categoría.
    }

    public function etiquetas(){
        return $this->belongsToMany(Etiqueta::class); // Muchos a muchos
    }
}
```

<https://www.youtube.com/watch?v=61evfkWG2tg&list=PLPI81lqbj-4KHPEGngoy5PSjjxcwnpCdb&index=19>

ruta para copiar los archivos JS

En la plantilla para agregar un script JS de tiene que agregar de la siguiente forma:

<script src="{{ asset('js/bootstrap.bundle.js') }}"></script> y copiar dicho archivo en la siguiente ubicación: **Public/js/**

ruta para copiar los archivos CSS

En la plantilla para agregar un script CSS de tiene que agregar de la siguiente forma:

<script src="{{ asset('css/bootstrap.css') }}"></script> y copiar dicho archivo en la siguiente ubicación: **Public/css/**

POSIBLES SOLUCIONES PARA EL ESTADO DE LA PAGINAS 419 | PAGE EXPIRED

- 1.- Colocarle todos los permisos de control total a las carpetas **storage, vendor y bootstrap**
- 2.- Ejecutar la limpia de cache en laravel con **php artisan cache:clear**

MODIFICAR LAS PLANTILLAS DE ERROR EN LARAVEL 6: 401, 403, 404, 419, 429, 500, 503.

- 1.- Copiar las plantillas blade que se encuentran en la siguiente ruta:
\vendor\laravel\framework\src\Illuminate\Foundation\Exceptions\views
- 2.- Crear una carpeta en: **resources/views/** con el nombre de **errors**
- 3.- Pegar los archivos antes copiados en esa carpeta y modificarlos.

RELACION ENTRE TABLAS MUCHOS A MUCHOS.

En el ejemplo vamos a relacionar **1 Usuario con 1 Rol**.

1.- Creamos el modelo **Role**: en nuestro símbolo el sistema tecleamos:

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

2.- En la migración que nos creamos **2021_01_22_152005_create_roles_table** colocamos los demás campos que contendrá la tabla **Role**.

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateRolesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('roles', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('descripcion');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('roles');
    }
}
```

3.- Crearemos únicamente la migración de la tabla pivote que en nuestro caso seria la tabla **role_user_table** de la siguiente manera:

```
php artisan make:migration create_role_user_table
```

4.- Abrimos la migración que nos crea que en este caso sería:

2021_01_23_003700_create_role_user_table y colocamos los demás campos que contendrá en este caso sería los campos: **user_id** y **role_id**.

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateRoleUserTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('role_user', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->integer('user_id')->unsigned();
            $table->integer('role_id')->unsigned();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('role_user');
    }
}
```

5.- En el modelo **User.php** que se encuentra dentro de la carpeta **app** indicamos la relación muchos a muchos creando una función llamada **roles**.

```
User.php
Role.php
2021_01_23_003700_create_role_user_table.php
app.blade

<?php

namespace App;

use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
    use Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'email', 'password',
    ];

    public function roles(){
        //ESTA ES LA FUNCION PARA INDICAR LA RELACION
        return $this->belongsToMany('App\Role');
        //belongsToMany significa la relacion
        //muchos a muchos con la tabla role (App\Role)
    }
}
```

6.- En el modelo **Role.php** que se encuentra dentro de la carpeta **app** indicamos la relación muchos a muchos creando una función llamada **users**.

```
User.php x Role.php 2021_01_23_003700_create_role_user_table.php x app.blade.php
1 <?php
2
3 namespace App;
4
5 use Illuminate\Database\Eloquent\Model;
6
7 class Role extends Model
8 {
9     //
10     public function users(){ ///ESTA ES LA FUNCION PARA INDICAR LA RELACION
11         .....return $this->belongsToMany('App\User'); ///belongsToMany significa la relacion
12         .....muchos a muchos con la tabla user (App\User)
13     }
14 }
```

7.- Ejecutamos **migrate** y se nos creara las tablas y la relación entre ellas.

php artisan migrate

VALIDACION DE CAMPOS.

1.- La validación de campos en un formulario para indicar si va ser el registro **único**, o es **requerido** u otras validaciones se realizan en el **controlador dentro de la función**, en nuestro ejemplo vamos a validar los campos de nuestro controlador **PeliculaController.php** dentro de la función **store** que se encuentra ubicado en **app/http/controllers**

```
/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    //AQUI SE REALIZA LA validaciones de campos.
    $validaciones = $request->validate([
        'nombre' => 'required|unique:peliculas', ///unique:peliculas es campo unico y se menciona la
        'nombretraducido' => 'required|unique:peliculas',
        'sinopsis' => 'required',
        'directores' => 'required',
        'actores' => 'required',
        'duracion' => 'required',
        'anioestreno' => 'required',
        'calidad' => 'required',
        'tiporecurso' => 'required',
        'generoid' => 'required',
        'origen' => 'required',
        'trailer' => 'required'
    ]);

    //
    $hoy = Carbon::now()->format('Y-m-d');

    $pelicula = new Pelicula();
    $pelicula->nombre = $request->nombre;
    $pelicula->nombretraducido = $request->nombretraducido;
    $pelicula->sinopsis = $request->sinopsis;
    $pelicula->directores = $request->directores;
    $pelicula->actores = $request->actores;
    $pelicula->duracion = $request->duracion;
    $pelicula->anioestreno = $request->anioestreno;
    $pelicula->calidad = $request->calidad;
    $pelicula->fechaacreado = $hoy;
    $pelicula->fechaactualizado = $hoy;
    $pelicula->visible = 1;
    $pelicula->usuarioid = 1;
    $pelicula->tiporecurso = $request->tiporecurso;
    $pelicula->generoid = $request->generoid;
    $pelicula->temporada = $request->temporada;
    $pelicula->capitulo = $request->capitulo;
    $pelicula->origen = $request->origen;
    $pelicula->trailer = $request->trailer;
    $pelicula->numvisualizaciones = 1;
    $pelicula->ranking = 1;
}
```

2.- En nuestra plantilla **blade o html** del formulario colocamos el bloque de código para mostrar errores.

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

@section('content')
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-8">
            @if($errors->any())<!-- AQUÍ SE MUESTRA LOS ERRORES DE VALIDACIÓN -->
            <div class="alert alert-primary" role="alert">
                @foreach($errors->all() as $error)
                <p>{{ $error }}</p>
                @endforeach
            </div>
            @endif<!-- AQUÍ SE MUESTRA LOS ERRORES DE VALIDACIÓN -->
            <div class="card">
                <div class="card-header d-flex justify-content-between align-items-center">
                    <span>Agregar Pelicula</span>
                    <a href="/peliculas" class="btn btn-primary btn-sm">Volver a lista de peliculas</a>
                </div>
                <div class="card-body">
                    @if ( session('mensaje') )
                    <div class="alert alert-success">{{ session('mensaje') }}</div>
                    @endif
                    <form method="POST" action="/peliculas" enctype="multipart/form-data">
                        @csrf
                        <input type="text" name="nombre" placeholder="Nombre" class="form-control mb-2" />
                        <input type="text" name="nombretraducido" placeholder="Nombre traducido" class="form-control mb-2" />
                        <label for="file">Seleccione la imagen de la pelicula</label>
                        <input type="file" name="foto" placeholder="foto" class="form-control mb-2" />
                        <textarea placeholder="Sinopsis" type="text" id="sinopsis" name="sinopsis" class="form-control mb-2">
                        </textarea>
                        <input type="text" name="directores" placeholder="Directores" class="form-control mb-2" />
                        <input type="text" name="actores" placeholder="Actores" class="form-control mb-2" />
                        <input type="text" name="duracion" placeholder="Duracion ejemplo: 01:30" class="form-control mb-2" />
                        <input type="text" name="anioestreno" placeholder="Año de estreno ejemplo: 2007" class="form-control mb-2" />
                        <select id="calidad" name="calidad" class="form-control mb-2">
                            <option value="0">Seleccione la calidad de la pelicula</option>
                            @foreach ($calidadpeliculas as $itemcalidadpelicula)
                            <option value="{{ $itemcalidadpelicula->id }}">{{ $itemcalidadpelicula->descripcion }}</option>
                            @endforeach
                        </select>
                        <select id="tiporecurso" name="tiporecurso" class="form-control mb-2">
                            <option value="0">Pelicula, Serie o documental</option>
                            @foreach ($tipopeliculas as $itemtipopelicula)
```

Para modificar los mensajes en **español los errores de validación** se tendría que modificar el script **validation.php** que se encuentra en **resources/lang/en/**

Las demás reglas de validación lo podemos checar en:

<https://laravel.com/docs/6.x/validation#available-validation-rules>

AGREGAR ROL AL USUARIO AL MOMENTO DE REGISTRARSE.

1.- En la clase **User** de la aplicación que se encuentra en: **app/User.php** se agregaran **3 funciones**, estas funciones sirven para buscar si al usuario se le asignara **1 ROL o mas de 2 ROLES**.

```
<?php
namespace App;

use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
    use Notifiable;

    public function roles(){ /// ESTA ES LA FUNCION PARA INDICAR LA RELACION
        return $this->belongsToMany('App\Role'); /// belongsToMany significa la relacion muchos a muchos con la tabla role
        (App\Role)
    }

    /// DESDE LA LINEA 17 HASTA LA LINEA 48 SE BUSCA SI EXISTE UN ROL
    public function authorizeRoles($roles)
    {
        abort_unless($this->hasAnyRole($roles), 401);
        return true;
    }

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

    public function hasRole($role)
    {
        if ($this->roles()->where('descripcion', $role)->first()) {
            return true;
        }
        return false;
    }

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'email', 'password',
    ];
}
```

2.- En el controlador **RegisterController.php** ubicado en **app/http/controller/** en el encabezado se tiene que agregar el controlador **Role.php** y al finalizar buscamos el **rol**.

```
<?php
namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;
use App\Providers\RouteServiceProvider;
use App\Role; /// Se agrega el Controller ROLE MONCHI
use App\User;
use Illuminate\Foundation\Auth\RegistersUsers;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Validator;
```



```

/**
 * Create a new user instance after a valid registration.
 *
 * @param array $data
 * @return \App\User
 */
protected function create(array $data)
{
    //return User::create([
    //    'name' => $data['name'],
    //    'email' => $data['email'],
    //    'password' => Hash::make($data['password']),
    //]);

    $user = User::create([
        //aquí inicia el código para agregar un rol al registrarse
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => Hash::make($data['password']),
    ]);

    $user->roles()->attach(Role::where('descripcion', 'Suscriptor')->first()); //después del WHERE
    //descripcion es el nombre del campo de la tabla ROLES y 'Suscriptor' es el rol que se le asignará
    return $user;
}

```

AGREGAR VARIOS ROLES A UN USUARIO y EDITARLOS.

1.- Creamos el modelo **User**.

php artisan make:model User

2.- Creamos el controlador **UserController** se creará un archivo en **app/http/controllers/UserController.php** no hay que hacer la migración porque ya existe la table users.

php artisan make:controller UserController --resource

3.- En el archivo de nuestro modelo **User.php** agregamos estas 2 funciones:

```

<?php

namespace App;

use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
    use Notifiable;

    public function roles(){ /// ESTA ES LA FUNCION PARA INDICAR LA RELACION
        return $this->belongsToMany('App\Role'); /// belongsToMany significa la relacion muchos a muchos con la tabla role (App\Role)
    }

    public function asignarRol($role) /// ESTA ES LA FUNCION PARA ASIGNAR ROL AL USUARIO
    {
        $this->roles()->sync($role,false);
    }

    public function tieneRol($role) /// ESTA ES LA FUNCION PARA SABER SI TIENE UN ROL EL USUARIO
    {
        return $this->roles->flatten()->pluck('descripcion')->unique();
    }
}

```


4.- En nuestro archivo de rutas **web.php** creamos la ruta para acceder:

```
Route::get('/', function () {
    return view('welcome');
});

Route::resource('/generos', 'GeneroController');
Route::resource('/tipopeliculas', 'TipopeliculaController');
Route::resource('/peliculas', 'PeliculaController');
Route::resource('/calidadpeliculas', 'CalidadpeliculasController');
Route::resource('/usuarios', 'UserController'); // rutas para el controlador de usuarios-MONCHIS
Route::get('/ver/{id}', 'PageController@verpelicula')->name('ver.verpelicula'); // ruta para
Auth::routes();
Route::get('/home', 'HomeController@index')->name('home');
```

5.- En nuestro archivo del controlador **UserController.php** en la parte de arriba usaremos el modelo **Role** y usamos la librería **DB** (esta nos servirá para ejecutar el query para eliminar los roles del usuario al momento de actualizar).

6.- En la función **public function store(Request \$request)** agregamos los roles de usuario del archivo **UserController.php**

```
public function store(Request $request)
{
    //
    //if ($request->input('roles')!=null)
    {
        $vroles = implode(' ', $request->input('roles'));
        $todosroles = explode(' ', $vroles);
    }

    return back()->with('mensaje', $todosroles[1]); //

    $validaciones = $request->validate([
        'name' => 'required',
        'email' => 'required|unique',
        'password' => 'required',
        'roles[]' => 'required'
    ]);

    $usuario = new User();
    $usuario->name = $request->name;
    $usuario->email = $request->email;
    $usuario->password = bcrypt($request['password']);

    $usuario->save();

    if ($request->input('roles')!=null) //Estas lineas de codigo antes del foreach realizan la
    funcion que recogen los valores del select multiple y los concatena con , linea 73 y despues yo lo
    separe como un array con explode linea 74
    {
        $vroles = implode(' ', $request->input('roles'));
        $todosroles = explode(' ', $vroles);
    }

    foreach ((array)$todosroles as $vrol){
        $usuario->asignarRol($vrol); //Esta linea agrega un rol al usuario
    }

    return back()->with('mensaje', 'Usuario Agregado Correctamente!');
}
```

Activar Windows

7.- En la función **public function update(Request \$request)** actualizamos los roles de usuario del archivo **UserController.php**

```
public function update(Request $request, $id)
{
    // $nrd=DB::delete('SQL-QUERY-HERE');
    $validaciones = $request->validate([
        'name' => 'required',
        'email' => 'required',
        'password' => 'required'
    ]);

    $idusuario=$id;
    $usuarioupdate = User::findOrFail($id);
    $usuarioupdate->name = $request->name;
    $usuarioupdate->email = $request->email;
    $usuarioupdate->password = bcrypt($request['password']);

    $usuarioupdate->update();

    $roleeliminado=DB::delete('delete from role_user where user_id='.$idusuario);

    if ($request->input('roles')!=null) {//////Estas líneas de código antes del foreach realizan la
    función que recogen los valores del select multiple y los concatenan con ',' línea 138 y después yo
    lo separo como un array con explode línea 139
    {
        $vroles = implode(',', $request->input('roles'));
        $todosroles = explode(',', $vroles);
    }

    foreach((array)$todosroles as $vrol){
        $usuarioupdate->asignarRol($vrol); /// En esta línea asignamos los roles al usuario
    }

    return back()->with('mensaje', 'Usuario Editado Correctamente!');
}
```

8.- Creamos la carpeta **usuarios en resources/views** donde agregaremos nuestras plantillas blade.

CONSUMO DE UNA API.

1.- Instalamos la librería **Guzzle 7** en su versión 7

<https://www.youtube.com/watch?v=13i6V3r3K34>

<https://docs.guzzlephp.org/en/stable/overview.html>

En el símbolo del sistema dentro de la carpeta del proyecto tecleamos lo siguiente:

composer require guzzlehttp/guzzle:^7.0

```
C:\xampp7\htdocs\hmpeliculas>composer require guzzlehttp/guzzle:^6.0
./composer.json has been updated
Running composer update guzzlehttp/guzzle
Loading composer repositories with package information
Updating dependencies
Lock file operations: 5 installs, 0 updates, 0 removals
- Locking guzzlehttp/guzzle (6.0.0)
- Locking guzzlehttp/promises (1.4.0)
- Locking guzzlehttp/psr7 (1.7.0)
- Locking psr/http-message (1.0.1)
- Locking ralouphie/getallheaders (3.0.3)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 5 installs, 0 updates, 0 removals
- Downloading ralouphie/getallheaders (3.0.3)
- Downloading psr/http-message (1.0.1)
- Downloading guzzlehttp/psr7 (1.7.0)
- Downloading guzzlehttp/promises (1.4.0)
- Downloading guzzlehttp/guzzle (6.0.0)
- Installing ralouphie/getallheaders (3.0.3): Extracting archive
- Installing psr/http-message (1.0.1): Extracting archive
- Installing guzzlehttp/psr7 (1.7.0): Extracting archive
- Installing guzzlehttp/promises (1.4.0): Extracting archive
- Installing guzzlehttp/guzzle (6.0.0): Extracting archive
1 package suggestions were added by new dependencies, use `composer suggest` to see details.
Package jakub-ondarka/php-console-color is abandoned, you should avoid using it. Use php-parallel-lint/php-console-color instead.
Package jakub-ondarka/php-console-highlighter is abandoned, you should avoid using it. Use php-parallel-lint/php-console-highlighter instead.
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: facade/ignition
Discovered Package: fideloper/proxy
Discovered Package: laravel/tinker
Discovered Package: laravel/ui
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Package manifest generated successfully.
66 packages you are using are looking for funding.
Use the `composer fund` command to find out more!

C:\xampp7\htdocs\hmpeliculas>
```